SOS: SELF-ORGANIZING SUBSTRATES

THÈSE N^{O} 3615 (2006)

PRÉSENTÉE LE 21 AOÛT 2006

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

Laboratoire de système d'information répartis

SECTION DES SYSTÈMES DE COMMUNICATION

ÉCOLE POLYTECHNIQUE FÉDERALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Anwitaman DATTA

Bachelor of Technology in Electrical Engineering, IIT Kanpur, Inde
de nationalité indienne

acceptée sur proposition du jury:

Prof. M. Hasler, président du jury
Prof. K. Aberer, directeur de thèse
Prof. M. A. Shokrollahi, rapporteur
Prof. P. Felber, rapporteur
Prof. E. Aurell, rapporteur

EPFL
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Lausanne, EPFL
2006

SoS: Self-organizing Substrates

Thèse de doctorat
présentée à
l'Ecole Polytechnique Fédérale de Lausanne

par

Anwitaman Datta

ॐ

Dedicated to my mother Smt. Sipra Datta,
In loving memory of my father Late Amarnath Datta.

ACKNOWLEDGEMENT

I have learnt more than just computer science from Prof. Karl Aberer, who has been my friend, philosopher and guide. I will be ever grateful for the independence I have enjoyed in determining the course of my research, even while he has shepherded me with his wisdom, kindness and patience.

The brain-storming sessions with Dr. Manfred Hauswirth have been crucial at many junctures of my research and the results therefrom are integral part of this thesis. I thank Roman Schmidt and Renault John for the collaboration we had in developing the P-Grid system from a conceptual embryo to the actual functional software, and also for permitting me to include the experimental results based on the deployed software's evaluation.

Working with Prof. Wolfgang Nejdl and Prof. Martin Hasler have not only been fun, but also, the results contributed to this dissertation. I am also thankful to have Prof. Martin Hasler as the president of my thesis committee.

Prof. Pascal Felber's experience with peer-to-peer systems has been of great value to the committee. Prof. Amin Shokrollahi not only facilitated the use of Digital Fountain codes for some of my ongoing work, but also provided insightful advises during my research, and I am glad to have him in my thesis committee. Having a physicist with expertise in looking into distributed systems as complex systems has provided the opportunity to scrutinize my work from diverse perspectives, and I am thankful to Prof. Erik Aurell for his participation in the committee.

I am thankful to Prof. Ion Stoica for hosting me at UC Berkeley and giving me the time and opportunity to work with and learn from him.

Chantal Menghini has gone out of her way to make my move to Lausanne and the stay since, comfortable. The opportunity to work with some other colleagues from LSIR including Fabius, Mohit, Oana, Philippe, Sarunas and Zoran have been rewarding and learning experiences. I'll like to thank Jean-Pierre Depertuis, Philippe Chammartin and Marc-André Lüthi for their superb system administration.

These past years in Switzerland, my life rested on a tripod.

Life in company of friends from LSIR and the extended network (all names not enumerated because of scalability reasons) has been fun. Sharing my office with Roman, desserts and discussions with Manfred and Zoran, movies chez Phil's and barbeques at Karl's garden; I'll miss. Friends from India have filled part of the vacuum of a life away from home, occasionally fulfilling the urge for the taste of Indian food in fun-filled evenings. Prasenjit's jokes, Debjani, Usha and Vish's cooking, travels with Rajesh and Partha, Subhash's witticism and trying to ski at Gayathri's insistence are but few of the many nice memories I'd not mind reliving.

The experience of living in Lausanne would never have been complete without the company and friendship of Frédérique to enjoy the evenings together, watching movies while not trying cuisines from all over the world or relishing the excellent cakes she bakes. Je lui suis reconnaissant de sa présence dans ma vie, toujours - le printemps, l'été, l'automne, l'hiver ... et le printemps. I can not imagine how I would have finished my thesis without her continuous encouragements and support. Thanks for being there, even when not here ever since you moved across an ocean and some more!

My family is the third leg, firmly supporting me from India, wherever I have been. If without my mother's immense help I would not have finished my under-graduation, without her patience and encouragements, finishing the PhD. would have been impossible. Without my brother Angshuman's timely help, I'd have failed to appear in a crucial examination which set the wheels of my academic life into motion. My departed father's attempts to instill some concepts from quantum physics when I was barely starting to grasp the structure of atoms among other things has hopefully not gone completely in vain, and memory of his enthusiasm to pursue knowledge has encouraged me throughout in my pursuit of the same.

Even while I hope to continue to learn new things in life, the time to finish my thesis is also a time to move forever from being formally a student, and I'll like to thank some of the teachers I had the privilege to learn from. My mother being a science and mathematics teacher herself, I had the opportunity to be taught by her until my secondary school, and my father occasionally exposed me to concepts and ideas (far) beyond the curriculum. The attempts to reduce my ignorance have since been ably helped by many, some of whom stand out. I'll like to particularly thank Prof. Jagdish Prasad, Prof. Vijay A. Singh, Dr. Arup Banerjee, Prof. R.K. Bansal and Prof. A.K. Chaturvedi, apart of course my thesis advisor Prof. Karl Aberer for influencing significantly the course of my life.

Living in Switzerland has been great fun, thanks to the opportunity provided by the hospitable people in general and EPFL in particular. I'll like to thank them, and the funding agencies for my research. This thesis was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322 and was (partly) carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Funding Agency OFES as part of the European project Evergrow No 001935.

Abstract

Large-scale networked systems often, both by design or chance exhibit self-organizing properties. Understanding self-organization using tools from cybernetics, particularly modeling them as Markov processes is a first step towards a formal framework which can be used in (decentralized) systems research and design. Interesting aspects to look for include the time evolution of a system and to investigate if and when a system converges to some absorbing states or stabilizes into a dynamic (and stable) equilibrium and how it performs under such an equilibrium state. Such a formal framework brings in objectivity in systems research, helping discern facts from artefacts as well as providing tools for quantitative evaluation of such systems.

This thesis introduces such formalism in analyzing and evaluating peer-to-peer (P2P) systems in order to better understand the dynamics of such systems which in turn helps in better designs.

In particular this thesis develops and studies the fundamental building blocks for a P2P storage system. In the process the design and evaluation methodology we pursue illustrate the typical methodological approaches in studying and designing self-organizing systems, and how the analysis methodology influences the design of the algorithms themselves to meet system design goals (preferably with quantifiable guarantees). These goals include efficiency, availability and durability, load-balance, high fault-tolerance and self-maintenance even in adversarial conditions like arbitrarily skewed and dynamic load and high membership dynamics (churn), apart of-course the specific functionalities that the system is supposed to provide.

The functionalities we study here are some of the fundamental building blocks for various P2P applications and systems including P2P storage systems, and hence we call them substrates or base infrastructure. These elemental functionalities include: (i) Reliable and efficient discovery of resources distributed over the network in a decentralized manner; (ii) Communication among participants in an address independent manner, i.e., even when peers change their physical addresses; (iii) Availability and persistence of stored objects in the network, irrespective of availability or departure of individual participants from the system at any time; and (iv) Freshness of the objects/resources' (up-to-date replicas).

Internet-scale distributed index structures (often termed as structured overlays) are used for discovery and access of resources in a decentralized setting. We propose a rapid construction from scratch and maintenance of the P-Grid overlay network in a self-organized manner so as to provide efficient search of both individual keys as well as a whole range of keys, doing so providing good load-balancing characteristics for diverse kind of arbitrarily skewed loads - storage and replication, query forwarding and query answering loads. For fast overlay construction we employ recursive partitioning of the key-space so that the resulting partitions are balanced with respect to storage load and replication. The proper algorithmic parameters for such partitioning is derived from a transient analysis of the partitioning process which has Markov property. Preservation of ordering information in P-Grid such that queries other than exact queries, like range queries can be efficiently and rather trivially handled makes P-Grid suitable for data-oriented applications. Fast overlay construction is analogous to building an index on a new set of keys making P-Grid suitable as the underlying indexing mechanism for peer-to-peer information retrieval applications among other potential applications which may require frequent indexing of new attributes apart regular updates to an existing index.

In order to deal with membership dynamics, in particular changing physical address of peers across sessions, the overlay itself is used as a (self-referential) directory service for maintaining the participating peers' physical addresses across sessions. Exploiting this self-referential directory, a family of overlay maintenance scheme has been designed with lower communication overhead than other overlay maintenance strategies. The notion of dynamic equilibrium study for overlays under continuous churn and repairs, modeled as a Markov process, was introduced in order to evaluate and compare the overlay maintenance schemes.

While the self-referential directory was originally invented to realize overlay maintenance schemes with lower overheads than existing overlay maintenance schemes, the self-referential directory is generic in nature and can be used for various other purposes, e.g., as a decentralized public key infrastructure. Persistence of peer identity across sessions, in spite of changes in physical address, provides a logical independence of the overlay network from the underlying physical network. This has many other potential usages, for example, efficient maintenance mechanisms for P2P storage systems and P2P trust and reputation management. We specifically look into the dynamics of maintaining redundancy for storage systems and design a novel lazy maintenance strategy. This strategy is algorithmically a simple variant of existing maintenance strategies which adapts to the system dynamics. This randomized lazy maintenance strategy thus explores the cost-performance trade-offs of the storage maintenance operations in a self-organizing manner. We model the storage system (redundancy), under churn and maintenance, as a Markov process. We perform an equilibrium study to show that the system operates in a more stable dynamic equilibrium with our strategy than for the existing maintenance scheme for comparable overheads. Particularly, we show that our maintenance scheme provides substantial performance gains in terms of maintenance overhead and system's resilience in presence of churn and correlated failures.

Finally, we propose a gossip mechanism which works with lower communication overhead than existing approaches for communication among a relatively large set of unreliable peers without assuming any specific structure for their mutual connectivity. We use such a communication primitive for propagating replica updates in P2P systems, facilitating management of mutable content in P2P systems. The peer population affected by a gossip can be modeled as a Markov process. Studying the transient spread of gossips help in choosing proper algorithm parameters to reduce communication overhead while guaranteeing coverage of online peers.

Each of these substrates in themselves were developed to find practical solutions for real problems. Put together, these can be used in other applications, including a P2P storage system with support for efficient lookup and inserts, membership dynamics, content mutation and updates, persistence and availability. Many of the ideas have already been implemented in real systems and several others are in the way to be integrated into the implementations.

There are two principal contributions of this dissertation. It provides design of the P2P systems which are useful for end-users as well as other application developers who can build upon these existing systems. Secondly, it adapts and introduces the methodology of analysis of a system's time-evolution (tools typically used in diverse domains including physics and cybernetics) to study the long run behavior of P2P systems, and uses this methodology to (re-)design appropriate algorithms and evaluate them.

We observed that studying P2P systems from the perspective of complex systems reveals their inner dynamics and hence ways to exploit such dynamics for suitable or better algorithms. In other words, the analysis methodology in itself strongly influences and inspires the way we design such systems. We believe that such an approach of orchestrating self-organization in internet-scale systems, where the algorithms and the analysis methodology have strong mutual influence will significantly change the way future such systems are developed and evaluated. We envision that such an approach will particularly serve as an important tool for the nascent but fast moving P2P systems research and development community.

Keywords: *Peer-to-peer (P2P), Randomized algorithms, Self-organization, Markov model.*

Version Abrégée

Les systèmes de réseaux à large échelle font souvent montre, par construction ou par hasard, de propriétés d'auto-organisation. Comprendre cette auto-organisation fait appel à des techniques provenant de la cybernétique, comme leur modélisation en tant que processus markoviens, qui est le premier pas vers un contexte formel, utilisable dans la conception et la recherche de systèmes (décentralisés). Parmi les aspects intéressants souhaitables sont compris l'évolution du système en fonction du temps, ainsi qu'étudier si et quand un système converge vers des états absorbants ou se stabilise en un équilibre dynamique (et stable), et comment il se comporte dans un tel état d'équilibre. Un tel contexte formel apporte de l'objectivité dans la recherche en systèmes, aidant ainsi à discerner les faits des artefacts, ainsi qu'à fournir des outils en vue d'une évaluation quantitative de tels systèmes.

Cette thèse introduit un tel formalisme en analysant des systèmes pair-à-pair (P2P, "Peer-to-Peer") afin de mieux comprendre leur dynamisme, ce qui à son tour aide à obtenir une meilleure conception.

En particulier, cette thèse développe et étudie les blocs fondamentaux à la construction de systèmes P2P. Ce faisant, notre méthodologie d'évaluation et de conception illustre les approches méthodologiques typiques de l'étude et de la conception de systèmes s'auto-organisant, ainsi que comment la méthodologie d'analyse influence la conception d'algorithmes afin d'atteindre les objectifs de conceptions de systèmes (de préférence avec des garanties quantifiables). Ces objectifs comprennent l'efficacité, la disponibilité, et la durée, l'équilibre des charges, la tolérance aux erreurs, et l'auto-maintenance même lors de conditions adverses, comme des charges arbitrairement déséquilibrées et dynamiques, un fort dynamisme d'adhésion ("churn"), ainsi que, clairement, les fonctionalités spéciales que le système est sensé fournir.

Les fonctionalités que nous étudions font partie des blocs fondamentaux de nombreuses applications P2P ainsi que de systèmes incluant des systèmes de stockage P2P, nous les appelons donc substrat ou infrastructure de base. Ces fonctionalités essentielles comprennent: (i) découverte fiable et efficace des resources distribuées dans le réseau de façon décentralisée; (ii) Communication parmi les pairs indépendamment de l'adressage, i.e., même lors de changement d'adresse physique; (iii) Disponibilité et persistance des objets stockés dans le réseau, indépendamment de la disponibilité ou du départ de participants individuels à un moment donné; et (iv) Fraîcheur des objets/resources (copies mises-à-jour).

Les structures d'indexage distribué à l'échelle d'internet (souvent appelées "overlays" structurés) sont utilisées pour la découverte et l'accès à des resources décentralisées. Nous proposons une construction rapide (partant de zéro) et une maintenance auto-organisées du réseau P-Grid pour une recherche efficace des clefs individuelles et d'un ensemble de clefs, fournissant ainsi de bonnes caractéristiques d'équilibre des charges pour différentes sortes de charges arbitrairement déséquilibrées - stockage et copies, retransmission de requêtes et charges de réponse aux requêtes. Pour une construction rapide d'overlay, nous utilisons une partition récursive de l'espace des clefs, afin que les partitions résultantes soient équilibrées par rapport à la charge de stockage et de copies. Les paramètres algorithmiques propres à de telles partitions sont dérivés d'une analyse transiente du processus de partition, qui est markovien. La préservation d'un ordre de l'information dans P-Grid telle que des requêtes autres qu'exactes, comme des requêtes sur des ensembles, puissent être gérées efficacement et presque trivialement, rend P-Grid adapté aux applications orientées données ("data-oriented"). La construction rapide d'overlay est analogue à construire un index sur un nouvel ensemble de clefs, faisant de P-Grid un méchanisme potentiel d'indexage sous-jacent pour des applications P2P de récupération d'information parmi d'autres applications potentielles demandant un indexage fréquent de nouveaux attributs, et des mises-à-jour d'un index existant.

Pour gérer la dynamique d'adhésion lors des sessions, en particulier changer l'adresse physique des pairs, l'overlay lui-même est utilisé comme un service d'annuaire (auto-référentiel) pour maintenir l'adresse physique des participants. Exploitant cet annuaire auto-référentiel, une famille de schémas pour la maintenance d'overlays est conçue, à coût de communication moindre que les stratégies existantes. La notion d'étude d'équilibre dynamique, sous "churn" continu et réparation, modelisé comme processus markovien, est introduite pour évaluer et comparer les schémas de maintenance d'overlays.

Alors qu'il était originalement inventé pour réaliser des schémas de maintenance d'overlays à coût moindre que les schémas existants, l'annuaire auto-référentiel est générique de nature et peut être utilisé pour différentes autres applications, e.g., comme une infrastructure à clé publique décentralisée. La persistance de l'identité des clefs lors des sessions, en dépit des changements dans l'adresse physique, fournit une indépendance logique du réseau overlay par rapport au réseau physique sous-jacent. Cela offre de nombreuses autres utilisations, comme des méchanismes efficaces de maintenance pour des systèmes de stockage, et un moyen de gérer la confiance et la réputation dans les réseaux P2P. Nous étudions en particulier la dynamique de maintenance de redondance pour les systèmes de stockage, et avons conçu une nouvelles stratégie "paresseuse" de maintenance. Celle-ci est algorithmiquement une simple variante de stratégies de maintenance existantes, qui s'adapte à la dynamique du système. Cette stratégie "paresseuse" randomisée explore donc les compromis coût-performance des opérations de maintenance nécessaires au stockage d'une manière auto-organisée. Nous modélisons le système de stockage (redondance), sous "churn" et maintenance, comme un processus markovien. Nous faisons une étude d'équilibre afin de montrer que le système opère dans un équilibre dynamique plus stable avec notre stratégie qu'avec les schémas de maintenance existants pour des coûts comparables. En particulier, nous montrons que notre schéma de maintenance fournit des gains substantiels de performance en terme de coût de maintenance et de résilience du système en présence de "churn" et de défaillances corrélées.

Finalement, nous proposons un méchanisme de bavardage (gossip), qui fonctionne avec un coût de communication inférieur à ceux existants, pour établir une communication parmi un ensemble relativement large de pairs non-fiables, sans supposer de structure spécifique sur leur connectivité mutuelle. Nous utilisons cette primitive de communication pour propager des mises-à-jour de copies dans des systèmes P2P, rendant plus facile de gérer les contenus changeants. La population de pairs affectée par le bavardage est modélisée comme un processus markovien. Etudiant la propagation transiente du bavardage aide à choisir les paramètres de l'algorithme afin de réduire le coût de communication tout en garantissant la couverture des pairs online. Chacun de ces substrats ont été développés pour trouver des solutions pratiques à des problèmes réels. Mis ensemble, ceux-ci peuvent être utilisés dans d'autres applications incluant un système de stockage P2P supportant des recherches et des insertions efficaces, de la dynamique d'adhésion, de la mutation dans les contenus, des mises-à-jour, de la persistance et de la disponibilité. Plusieurs de ces idées ont déjà été implémentées dans des systèmes réels.

Cette dissertation a deux principales contributions. Elle fournit une conception de systèmes P2P, qui sont utiles à des utilisateurs comme à d'autres développeurs, qui peuvent construire d'autres systèmes sur ceux existants. Elle adapte et introduit la méthodologie d'analyse de l'évolution du temps d'un système (outil typiquement utilisés en physique et cybernétique) afin d'étudier le comportement à long terme des systèmes P2P, et utilise cette méthodologie pour (re-)concevoir des algorithmes appropriés et les évaluer.

Nous avons observé qu'étudier les systèmes P2P du point de vue des systèmes complexes révèle leur dynamique intérieure et donc des moyens d'exploiter celle-ci pour des algorithmes adaptés ou meilleurs. Autrement dit, la méthodologie d'analyse en elle-même influence fortement et inspire la façon dont nous concevons de tels systèmes. Nous croyons qu'une telle approche dans l'orchestration de l'auto-organisation dans les systèmes à l'échelle d'internet, où les algorithmes et les méthodologies d'analyse ont une forte influence mutuelle, va changer significativement la façon dont de tels systèmes seront développés et évalués dans le futur. Nous pensons qu'une telle approche va servir d'outil important pour la communauté, naissante mais déjà grandissante, de recherche et développement dans les systèmes P2P.

Mots-clés: *Pair-à-pair (P2P), Algorithmes randomisés, Auto-organisation, Modèle markovien.*

Table of Contents

Part I. Background

Part II. Self-organizing overlay substrate

Part III. Content management in internet-scale systems

Part IV. Conclusion

1. Preamble

"Anything you build on a large scale or with intense passion invites chaos" — Francis Ford Coppola

1.1 The peer-to-peer (P2P) paradigm

The recent years have witnessed a paradigm shift in the usage of the internet, with widespread proliferation of peer-to-peer technologies for diverse applications and services - including content storage and sharing (file-sharing, content distribution, backup storage) and communication (voice, anonymous, instant messages, multicast, internet indirection) to name a few.

But what is the peer-to-peer (P2P) paradigm?

Since the year 1999-2000, when the first incarnation of Napster was beginning to be incinerated with lawsuits, making the phrase peer-to-peer (P2P) not only well-known but also unfortunately synonymous to pirate-to-pirate sharing of files violating copyrights, and such a shadowy image further augmented by propaganda on the potential misuses of anonymized content distribution networks like Freenet [39, 38], a holistic, precise and unanimously agreed definition of peer-to-peer is yet to be found!

Over the years however, with the advent of many other legitimate business models using peer-to-peer technology, the perception about the peer-to-peer paradigm has changed positively. Even at what is the nascence of this born-again[1] paradigm of computing, P2P's potential is being better appreciated as more than one established industry is forced to rethink their business models - be it the entertainment industry or the telephony sector.

Peer-to-peer systems are large, networked, distributed systems.

From the application perspective, the P2P paradigm is essentially about utilization of otherwise unutilized resources available from across the world - for the end-users from the end-users. It is about performing processing, bandwidth, storage or human-resource intensive tasks and value additions - which would otherwise have had incurred dedicated and expensive infrastructure, were it to be done by central service providers.

[1] The internet was originally designed to specifically facilitate peer-to-peer interactions, as we'll recapitulate in Section 2.1.

From the systems perspective, the peer-to-peer paradigm is about management of such autonomous and individually unreliable resources which come together dynamically in order to provide (albeit often probabilistic) guarantees to meet whichever desired objectives the system is meant to meet, without global knowledge, central control or dedicated infrastructure.

One most debated topic while defining a peer-to-peer system is whether such a system comprises of any centralized component or not. Looking at some of the classic peer-to-peer systems like Napster [170] (for file sharing), Seti@Home [18] (for grid computing), PlanetLab [37] (testbed for distributed systems) or Skype [160] (for VoIP communication), we see that some part of these systems are centralized - Napster's index of files being shared by users, Seti@Home's management of distribution of task and aggregation of results, PlanetLab's management of slices and Skype's management of user accounts. These components are purely client-server. On the other hand other components of these systems, like the actual storage and bandwidth for file transfer in Napster, computational infrastructure in Seti@Home and PlanetLab, and discovery of the latest physical address (IP address) and the voice communication among Skype users involve only the resources from the users themselves. Thus to say, a hybrid architecture with some centralized component may still have many other functions performed in a peer-to-peer manner.

However centralization (physical or logical) can become a single point of failure or attack (physical and/or legal), a bottleneck, incur high administrative overhead or be a point of censorship or monopoly or may simply not scale.

To that end, completely decentralized systems are desirable. A point in case is the advent of the Gnutella (unstructured overlay) network among others to fill in the vacuum left by Napster's closure.

The Gnutella story also exposes the other side of the story of decentralization. Gnutella's originally flat structure meant that locating resources required flooding the whole network which is bandwidth intensive, in contrast to a single message exchange that was required in the client/server paradigm used in Napster. A partial flooding on the other hand results in incomplete results, that is, a resource present in the network may well not be found (in information retrieval terminology, $recall < 1$). Super-peer based hierarchical architectures alleviate the problems to certain extent.

At this point of the story, a nagging question remains - does decentralization help or hinder scaling, and what about the service guarantees?

Napster provided a centralized index, making resource discovery very efficient and complete, but becoming a single point of attack/failure. Gnutella did away with the central server, and at the same time lost the index, making the resource discovery process both imprecise and expensive and slow.

A third way, a middle path of sorts is however to build a decentralized index (structured overlays). Structured overlays can facilitate resource discovery completeness efficiently[2] based on communication with only a very small fraction of the peer population - typically logarithmic.

[2] Purely in terms of bandwidth efficiency, a centralized system will of-course be more efficient.

This example shows that a properly designed decentralized system can indeed scale and provide performance guarantees - since with larger number of participants the available resources in the system also grow. Thus, in this dissertation, we consciously focus on design of completely decentralized systems. In fact a significant part of this dissertation is dedicated to the design of a structured overlay network. More on the objectives and contributions of this dissertation will follow later.

Intuitively, nature prefers disorder than order. The fundamental questions to ask about decentralized systems are - how then do such decentralized systems with desirable properties come into being, and how do such systems sustain these desirable properties over a prolonged period of time.

Many large-scale systems, even those which have some central components managing it partially, by design or coincidence - exhibit what can be termed as *self-organization* and *self-maintenance*. Such self-organization allows individual or small units of participants to participate autonomously and act locally such that a global behavior emerges. Examples as diverse as ant colonies to the internet may be cited as instances of such self-organization.

We rely heavily on this potential of large scale systems to self-organize and self-maintain, in order to design what we call *self-organizing substrates*, that in turn provides some fundamental efficient and reliable building blocks for other large-scale systems and applications judiciously using (for example to provide load-balancing) resources spread across the edge of the network, even if the participants are individually unreliable and autonomous.

Before venturing further and describing the specifics of the substrates/services we endeavor to provide, a word of admonition regarding the scope of this dissertation is necessary. We consider that participants of the system (peers/nodes) are cooperative but nonetheless unreliable and autonomous. We consider a bimodal unreliability - either the participant provides the resources at its disposal, or its away from the system (offline/crashed). The whole process of self-organization and self-maintenance is then to utilize the resources available in the system at any time, which varies over time, but with full cooperation of these online participants.

Issues like trust, incentives, selfish, non-cooperative, or intently malicious participants (launching distributed denial of service - DDoS - attacks) are not considered in this work. As more and more peer-to-peer systems and applications proliferate our daily lives, such security concerns and enforcement of cooperation will become even more critical, and is clearly a big challenge for the P2P research community in general. These are active research topics, and some of the existing ideas to deal with these problems may be used in conjunction with the systems designed here. But these security issues are out of the scope of this dissertation.

1.2 Self-organizing Substrates

Most peer-to-peer applications require one or several of the following elemental functionalities: (i) Reliable and efficient discovery of resources distributed over the network, (ii) Communication among participants (preferably in an address independent manner) even when peers change their physical addresses (logical mobility), (iii) Availability and persistence of stored objects in the network, irrespective of availability or departure of individual participants from the system at any given time and (iv) Freshness of the objects/resources' (up-to-date replicas).

These functionalities are in themselves realized by building full-fledged systems. These are also used as building blocks to build other P2P systems and functionalities and hence we call these as substrates. Literally, a substrate means an underlying layer; a substratum. It is not only that other P2P systems need some of these elemental functionalities to build upon, but also these substrates themselves have mutual interdependencies.

We will point out some potential uses of these substrates when we study each of them in detail, and we'll also point out the interdependencies among these substrates at relevant junctures. We'll summarize these dependencies and interdependencies later in Table 10.1.

The principal contribution of this dissertation is to look into the self-organizational aspects of these substrates - dealing with the dynamics of the system arising from membership changes (churn) and judiciously using the resources (load-balancing) adaptive to the workload.

1.2.1 Structured overlay networks

Structured overlay networks have become a standard technique to provide efficient resource discovery over the network. A structured overlay is essentially a distributed index structure. Distributed Hash Tables (DHTs) like Chord [163], Pastry [154] and CAN [140] were some of the earliest structured overlay networks. DHTs however are not suitable for data-oriented applications (e.g., to do range queries). We propose the construction and maintenance of the P-Grid overlay network in a manner so as to provide efficient search of both individual keys as well as a whole range of keys [52], and doing so providing good load-balancing characteristics [6, 8].

In addition, we propose a parallelized mechanism [8] to construct (such load-balanced) overlay networks - analogous to re/construction of an index in databases - as a departure from the traditional approaches which looked into a (quasi-)sequential network membership change (churn).

In order to support range queries, it is necessary to retain ordering information while creating the keys to be indexed. Doing so leads potentially to non-uniform distribution of keys over the key-space. Our overlay construction mechanism thus deals with unknown and arbitrary such distributions in order to build load-balanced overlays. Moreover, we can build such an overlay in a rapid manner. Such fast overlay construction is analogous to (re-)indexing, useful for example to construct a new index based on a new attribute. These two properties - preservation of ordering information and fast construction of a new index, and doing so in a load-balanced manner makes P-Grid suitable for more data-oriented applications including using it as the underlying indexing mechanism for peer-to-peer information retrieval applications.

Apart dealing with skewed distribution of keys, we deal with a gamut of other load-balancing issues encountered in overlay networks - including the determination of an optimal caching scheme (how many and where to cache for a limited storage capacity) to deal with skews in usage load of individual keys.

1.2.2 Managing peers' identity and logical mobility

A directory service providing meta-information about peers can be of immense use for various purposes. This meta-information can be diverse and derived based on diverse ways, ranging from for instance the latest physical address of peers or some relatively abstract notion like the trustworthiness of peers. The overlay index itself is essentially a directory. So the overlay can as well be used as a directory service for peers participating in the overlay itself.

We propose the use of such a self-referential directory [5] using the P-Grid overlay itself in order to preserve securely (resistant against impersonation attacks) peers' identity (ID) over multiple sessions over which peers' physical addresses (IP) change. Based on such a self-referential directory storing peers' ID-to-IP mapping, a family of efficient lazy route repair schemes have also been proposed (Correction on Use and Correction on Failure) which is efficient and effective for a wide range of churn levels. In fact such repair is achieved by a small alteration of the greedy routing in the overlay, which makes the routing process to recursively trigger new queries in the network leading to self-healing. We validate the feasibility of such a self-healing overlay maintenance mechanism based on a self-referential directory in the context of the P-Grid network.

1.2.3 Persistent and available storage

Collaborative storage systems spread over the network can be an economical way to store and back-up data. Such a mechanism is not only cost-effective, eliminating the need of expensive backup tapes or RAID systems, but also can provide better protection against geographically restricted physical damages like a fire in a building or a hurricane in a city, in which case, unless the back-up is in a different physical location, it is of no use. A peer-to-peer storage system can thus cater to diverse kind of end users - from individuals to big multinational corporations. In P2P storage systems, redundancy is used for persistence and high availability of content stored in the system irrespective of availability of individual participants. Over time, as the peer population changes and some members leave the system either temporarily or for ever, it becomes necessary to maintain a certain level of redundancy. Traditionally, any loss (potentially temporary) of redundancy was compensated immediately. Such a proactive approach however turns out to be expensive since it does not exploit the returning peers which bring back the temporarily unavailable content - particularly for large objects. A recent deterministic lazier approach [25] proposed to trigger repairs when a given threshold of the redundancy is lost. Such an approach makes redundancy maintenance efficient - but we prove it to be actually vulnerable - with resistance to only low level of churn, and very little or no resistance against correlated failures. We provide an alternative randomized lazy repair strategy - which samples a smaller random subset of the redundantly stored content and replaces immediately the lost redundancy. By making the size of the random subset adaptive to the current availability of the object - so that not all the losses are immediately detected - in effect we device a hybrid of the existing proactive and lazy mechanism, and it performs gracefully in terms of the maintenance cost as well as resilience. In fact, the maintenance cost of our scheme averaged over time is comparable (and often lower) than the existing deterministic lazy approach, while our scheme has better resilience against both regular churn as well as correlated failures (than the deterministic lazy approach). These approaches were analyzed and the results validated for erasure-code based redundancy, however the underlying principles hold for other redundancy mechanisms, (namely - pure replication or rateless erasure codes) and hence our maintenance strategy can be used for better exploration of resilience-maintenance cost trade-offs for redundancy maintenance in storage systems using any of the redundancy mechanisms.

1.2.4 A gossiping primitive

Redundantly stored content need to be updated. Such update is required to introduce either a new version of an existing object or to introduce a new object at multiple sites (replicas). Depending on the application or object popularity, the size of the replica population may be large and even changing - such that a point-to-point communication directly from the originator of the update to each and every replica may not scale. Moreover all replicas may not be online when an update operation is performed, and later the originator may as well go offline. To deal with the challenges of scale and membership dynamics, we proposed a push/pull based gossip communication primitive to propagate updates within an unstructured (replica) sub-networks. This update mechanism is integrated in the P-Grid system for overlay replica maintenance. Though designed for update propagation, the gossip mechanism can as well be used for purposes like probabilistic communication in large dynamic groups.

1.3 The philosophy and practice of self-organization

Both by design or chance, large-scale systems exhibit self-organizational properties. The above mentioned contributions to algorithms and system design rely on better understanding and in turn exploitation of the dynamics of self-organization in such large scale systems.

Incidentally, just like “peer-to-peer”, “self-organization” is in-fact an even more difficult to define buzz word and there are very many subjective interpretations, even as researchers from domains as diverse as physics, biology/ecology, social science and computer science (cybernetics) study self-organization in diverse systems - from the domain alignment in magnets, ant colonies, small-world phenomenon in social or communication networks to give some well-known examples.

Informally speaking, self-organization is about distribution of control (decentralization), where individual participants act locally and autonomously, based on local stimulus or information, and still global properties emerge. E.g., preferential attachment [22] leading to power-law connectivity of the internet (routing infrastructure), world wide web (hyperlink connections), Gnutella network.

One interpretation of self-organization [84] is as follows. *"Self-organization is the process of evolution of a complex system with local interaction of system components only, resulting in system states with certain observed or intended global properties. A self-organizing process is driven by randomized local variations. These "fluctuations" or "noise" as they are also called, lead to a continuous perturbation of the system and allow the system to explore a global state space until it enters into (dynamic) equilibrium states. These states correspond to the global, emergent structures."*

A mathematical interpretation of the above definition is to study the system as a probabilistic system with Markov property, and the equilibrium or absorbing state(s) for the corresponding Markov chain provides then the emergent state of the system. We briefly look into the general framework for modeling and studying self-organization under this premise.

1.3.1 Probabilistic systems

Traditionally system properties are described in terms of *observables*. The full description of the properties of the system - represented by a state vector s determines the *state* of the system. Typically, the state is time dependent, and hence can be represented as $s(t)$. A typical example will be from classical physics - that of the position of a moving object.

Often in a complex system, the exact state of the system can not be known for certain, and instead the *probability distribution of the states* - P is definable and is of greater interest. Thus to say, if the system resides in state s_i with probability P_i,[3] then the *probabilistic system*'s state can be represented as $P = \sum_{\forall i} P_i(s_i)$. A typical example will be the outcome of a coin-tossing experiment.

The dynamics of the system is then described by the time evolution of the probability distribution function $P(t)$.

1.3.2 Markov model for self-organization

Given a set of possible states S, which usually is very large, the evolution of a complex system can be described deterministically by a function $f_T : S \rightarrow S$. In practice, the lack of information about the precise state will make a deterministic description of the system evolution infeasible. Thus a more realistic way to describe the system's evolution is by a stochastic process following the Markov property.

[3] Using this notation implicitly assumes a discrete state space, though an extension to a continuous space is natural.

For each given state s_j we can give the probability that a state s_i is reached, i.e. $P_i(s_i|s_j) = M_{ji} \in [0, 1]$, where M is the transition matrix of a Markov process. Given the probability distribution of states $P_j(s_j, t)$ at time t it is thus possible to calculate the time evolution of the system as

$$P_i(s_i, t+1) = \sum_i M_{ji} \, P_j(s_j, t).$$

The emergent behavior of the self-organizing system correspond to the equilibrium or absorbing states of the Markov chain, determined by the distribution $P = \sum_{\forall i} P_i(s_i)$. This distribution should be time-independent once the system has reached its equilibrium or absorbing states.

This is the principal mathematical framework we will use in this dissertation to study the behavior of the algorithms, and hence the systems we'll design. Whenever applicable, the analysis results have been validated against simulations. We used such a two pronged approach of using both analysis and simulations because the simulation based validation helped in sanity checking the analytical model, and in turn the theory helped discern facts from implementation artifacts.

Even though the emphasis has been to develop algorithms based on a fundamental understanding of the system's behavior based on appropriate models, we'd like to admit that since our primary objective is to build practical systems we have not shied away from heuristics either. For what are heuristics in systems design would be called art (or may be black art) in other trades. We also evaluate the performance of algorithms and systems built on such heuristics based on simulation experiments.

1.4 Thesis organization and main contributions

This thesis is divided in four parts.

In **Part I** we provide general background on peer-to-peer systems. Chapter 2 starts with a brief history of the peer-to-peer paradigm followed by a closer look at structured overlays. Structured overlays are distributed index structures storing key-value pairs, where the value is the resource (or pointers to the resource) which is discovered by looking for the key. Thus structured overlays primary role is to support resource discovery in a decentralized setting. Structured overlays can also be used for substrate for other applications (e.g., broadcast [58]) and other overlays built on top (e.g., Scribe [34] for multicast trees, GridVine [3] semantic overlay networks), and is a fundamental building block for many peer-to-peer systems.

In **Part II** we look into the various aspects of structured overlay network design. While many of the ideas are general, some others are specific to the P-Grid (www.p-grid.org) system [1, 7, 8]. The ideas are almost always validated with P-Grid unless otherwise stated.

In Chapter 3 we first introduce the P-Grid structured overlay network. The P-Grid routing network is motivated by prior works on prefix based routing [132] and scalable distributed data structures (SDDS) [108, 109] and was introduced [1] prior to the commencement of this dissertation. Thus the contributions of this dissertation start from Section 3.3.

(i) We introduce and analyze the algorithms for performing range queries in P-Grid. In Chapter 4 we introduce various mechanisms to construct a load-balanced P-Grid network.

(ii) Apart sequential overlay construction, as has been traditionally studied, we propose a mechanism to construct the overlay rapidly in a highly parallelized manner.

(iii) The overlay construction mechanism ensures (storage) load-balance at each peer even in presence of skew in the load-distribution (over the key-space). Such load-skews typically occur when lexicographic ordering is preserved, which is necessary to perform efficient range queries.

(iv) We propose stochastic mechanisms to balance the replication of different keys in the structured overlay. In contrast to related works, we do not impose a globally predetermined replication factor. Instead we allow peers to decide on the absolute load they are willing to bear, i.e., absolute amount of resources they are willing to contribute to the overlay. Based on that the replication (of the index) is adapted to the available resources in the system.

In Chapter 5 we concentrate on query-load balancing, particularly when different keys are queried with different frequencies.

(v) We show that replication proportional to the query load not only provides a first-order balancing[4] of query-answering load (which is intuitive), but also optimizes the search latency in structured overlays with logarithmic search-cost. This simultaneous optimization of load-balancing and search cost is not intuitive since in various contexts like unstructured overlays [111] and mobile broadcast disks [12], replication proportional to the square-root of the popularity has been shown to be optimal.

Having dealt with the issues of overlay construction and maintenance to achieve diverse sorts of load-balance, in Chapter 6 we focus on the maintenance of the overlay under churn, particularly focusing on the issue of peers' logical mobility (change of physical/IP address).

(vi) We propose a self-referential directory. The self-referential directory in itself is general purpose.

(vii) We propose a family of self-healing routing mechanism, which uses the self-referential directory in order to discover peers' latest ID-to-IP mappings when such mappings change.

Incidentally Skype [160] claims to have achieved the same functionalities independently in a super-peer architecture, and asserts that having outsourced this critical and resource gobbling activity to the users, they

[4] If there is skew in the distribution of load (query load in this case), different peers are likely to be subjected to different load. By first-order balancing we mean that the load is redistributed such that each peer is likely to get the same load - i.e., have a random uniform distribution of load among the peers. This in itself however does not automatically mean that in reality each peer will have exactly the same load - because of statistical variation. To reduce such statistical variation, second-order balancing mechanisms are also required in practice to improve the quality of load-balancing. By second-order balancing we mean mechanisms to reduce the variation as is observed based on solely first-order balancing.

could improve their quality of service. This also exemplifies the practical problems we try to solve in this dissertation.

(viii) We adapt tools from cybernetics to model the dynamics of continuous loss of and repairs of routing information to determine the system's performance. Given a fixed rate of churn (membership dynamics), the overall system operates at a dynamic equilibrium. In the context of peer-to-peer systems, we thus introduce the concept of dynamic equilibrium, which extends the previous analytical tools to understand the system behavior under churn. These previous tools included the notion of static resilience (how the system behaves before any repairs) [75] and half-life (lower bound on repair costs to reach a fully consistent state) [106]. The notion of dynamic equilibrium is necessary - since it explicitly models the fact that the system will never arrive at a fully consistent state, and instead operate at the equilibrium state. This also leads way to determine the maintenance cost for any specific maintenance mechanism and the system's performance and resilience when the specific maintenance mechanism is used.

In Chapter 7 we provide experiment results based on deployment of the actual Java based P-Grid software in the PlanetLab testbed. The implementation of the software is a larger collaborative effort, and these experimental results are included primarily in order to validate the theory. The P-Grid implementation exclusively (but not exhaustively) uses algorithms developed in this thesis, and some of the more recent algorithms proposed here are still in the pipeline to be integrated and/or benchmarked.

In **Part III** we look into some aspects of content and storage management in peer-to-peer systems.

Chapter 8 looks into the dynamics of peer-to-peer storage systems under churn.

(ix) We demonstrate the general applicability of using the notion of dynamic equilibrium (earlier developed to model the behavior of the overlay network) in peer-to-peer settings by studying storage systems under churn.

(x) We propose a simple randomized lazy redundancy maintenance algorithm, which can better explore the cost-resilience tradeoffs and has both lower overhead and better resilience (against churn as well as correlated failures) than existing lazy maintenance mechanism [25] for storage systems.

In Chapter 9 we propose a generic gossip based communication primitive for unstructured subnetworks.

(xi) We introduce a push/pull gossiping primitive. This mechanism was originally devised to propagate update messages among replica sub-groups efficiently and effectively even in presence of churn. The mechanism is efficient since it reduces duplicate messages and effective since it reaches all online replicas (probabilistically).

We conclude in Chapter 10 (**Part IV**). There we look at the interdependence of the substrates themselves, as well as how some other P2P systems are using some of these building blocks.

This dissertation comprises of ideas developed over the last few years which have seen vigorous activity in the peer-to-peer research community. There have thus come about alternative, and sometimes better mechanisms to deal with certain problems. New results, small or big, keeps pouring in almost daily. We try, to the best of our knowledge, to provide and put in context our work with not only what existed prior to our contributions but also to what have come about since.

We have particularly tried to design systems that function efficiently for a wide range of environment in a self-adaptive manner avoiding the need for manual configurations and predetermined hard-coded global parameters - which can be serious impediments to deploy and manage the systems in diverse and changing environments. To that end, the analysis of these systems as complex systems has not only helped in revealing the dynamics of such large-scale systems (post-design analysis), but also to exploit the same in order to design improved and adaptive algorithms.

For instance, the algorithms developed for both overlay route maintenance as well as redundancy maintenance in storage system have inherent adaptivity to the environment, such that the maintenance cost gracefully increases with increased dynamicity, and is marginal in a stable environment, unlike other existing approaches which heavily rely on global predetermined parameters in their design, and often incurs a constant cost irrespective of the dynamics. Philosophically similar adaptivity may be observed in various other aspects of various other algorithms we develop. For example, for simultaneous load-balancing of storage and replication, our system automatically adapts the replication factor depending on available resources available in the whole system, based on the storage space each peer is willing to devote (which is easier to determine locally) in comparison to other overlays which rely on a globally predetermined replication factor hard-coded in the algorithms and the software. Likewise, for gossip based update propagation, the algorithm we propose has in-built mechanism to avoid duplicate messages, and can potentially be further modified to incorporate adaptivity to the environment. We have explored the potential of such adaptivity to reduce communication overheads even though we have not currently exploited such potential for the gossip scheme.

Finally, we'd like to reemphasize that even though this dissertation looks more into the design issues, the algorithms proposed here has led to a fully functional implementation of the P-Grid overlay network in Java (www.p-grid.org). A storage system exploiting specific properties of Digital Fountain erasure codes is also being implemented. The P-Grid overlay is in turn being used as an underlying primitive for various other applications like peer-to-peer information retrieval [30], peer data management systems and semantic overlays [3].

Part I

Background

2. Peering into peer-to-peer systems

“Interdependence is and ought to be as much the ideal of man as self-sufficiency. Man is a social being.” — Mohandas K. Gandhi

2.1 Back to the future

The essential philosophy of peer-to-peer (P2P) systems is to exploit resources available at participating members (peers) of the system.

In effect its a mechanism to enable the collaboration of and use of otherwise unused resources available at the individual end-users at the edge of the internet - rather than relying on dedicated service providers and infrastructures.

The resources at the edge can be physical resources like storage, computation or bandwidth, human resources or content or information/knowledge at these edge points of the network.

To emphasize the human factor which we won’t study any further in this thesis - we would like to give a few examples, which are inherently peer-to-peer in nature, even if not always explicitly recognized as such. Open-source software projects essentially use human resource in a peer-to-peer manner, as does the collaboratively written and maintained encyclopedia Wikipedia [171], even if these projects are maintained at central servers. Another instance is the PGP [66] decentralized public key infrastructure, where the social network is mapped into a web-of-trust to certify public keys.

The term peer-to-peer became popularized by music file-sharing networks, particularly by early birds Napster [170] and Gnutella [71], however the peer-to-peer paradigm has a much longer history in the world of computing, well summarized in “A revisionist history of peer-to-peer” [57].

Even before the file-sharing networks made peer-to-peer a commonly understood phrase, since the times when the internet was first conceived (ARPANET), the internet was designed to enable isolated and diverse computer networks across the U.S. to interact among themselves in a peer-to-peer fashion, so to say, without hierarchy or special roles for any specific participant. Similarly, many applications like FTP or Telnet use two computers as server and client, however any computer can play the role of both - essentially demonstrating the symmetric (peer-to-peer) relationship of the computers in the system. This role symmetry of individual peers acting as both a client and a server led to the notion of servents. Usenet’s server synchronization used a peer-to-peer gossiping scheme. Similar gossip/epidemic schemes for synchronizing autonomously replicated databases [54] have also been used in the database community. The DNS, organized hierarchically, uses delegation to scale, enabling numerous autonomous participants to come together and form the internet as we know it today. Finally, the routing infrastructure of the internet itself works in a peer-to-peer fashion, where different ISPs establish peering relationship to each other to forward the packets between diverse destinations and sources.

2.1.1 Rise of the servers

With the proliferation of personal computers hooked to the internet with slow modem connections, the dominant applications on the internet during the dot-com boom in the late 90s comprised of web-browsing, email & chat and e-commerce portals - relatively few service providers using dedicated infrastructure to cater to a huge number of users connected through poor connections with their relatively resource constrained personal computers. Such a usage pattern also meant that the internet infrastructure subsequently moulded itself accordingly - as can be seen from the asymmetry of upload/download connections most ISPs provide to this date, among other things. Moreover security aspects, and the sheer need to scale to the number of end users with a limited addressing space (which led assignment of temporary physical addresses) all meant that end-users directly communicating among each other was neither feasible, and mostly undesirable. Dynamic assignment of IP addresses, NATs and firewalls proliferated, to deal with the rapid expansion of the internet's userbase. Autonomous clients connecting to well-provisioned and well-maintained servers came to be the usage pattern.

This first phase of mass-commoditization of the internet thus saw a dominance of the client-server model, with relatively few servers catering to a large number of clients. Such a client/server usage model has many advantages, and will definitely play an important role in the future of the internet. Nonetheless, there are diverse application scenarios which require a peer-to-peer paradigm of user interactions.

2.1.2 P2P: A born again networking paradigm

Several things happened around the end of the last millenium so that the potential of peer-to-peer computing and networking paradigm was (re-)discovered. It is thus very likely that the future will see both client-server as well as peer-to-peer and hybrid architectures playing significant role, depending on application and resource requirements.

Moore's law caught up with the user-base of the internet. Users had larger disk-spaces, and improving internet connectivity often with a flat pricing. Users had local collection of music files thanks to the mature MPEG-1 Audio Layer 3 (MP3) compression technique. And may be, without advocating for piracy, but just as an explanation, one can argue that the monopoly of the music industry meant that audio CDs were exorbitantly priced. Almost all the ingredients for the next revolution were there - to democratize the internet

- where users could access resources in general, and music files in particular, stored by fellow users over the internet. The missing link was an effective way to find the available resources.

Finding resources and information spread across the internet is a non-trivial problem[1]. Napster served the niche market of discovering music (MP3) files spread across the internet in users' hard-disks by providing a server which would collect information and index the files stored by the clients. All Napster users could then search this central index to locate the file they were seeking. So far, every thing followed the client/server model.

However, bandwidth (and also storage) being precious resources, the server did not participate in the actual file transfer. Instead the file transfers were done directly by and between the end users. Thus we see in what has now come to be known as a *first generation P2P technology* how the server avoided the bandwidth and storage intensive tasks by exploiting the end user resources.

P2P was born again as a networking and computing paradigm.[2] Internet was back to the future, where it'd see the bulk of its traffic [61, 128] for applications using the communication mechanism for which the internet was originally conceived - computers at the edge of the network directly communicating and collaborating with each other.

Ironically, peer-to-peer came to the fore because of the buzz that followed the Napster system, which used the client-server model to solve the difficult problem of resource discovery over the internet.

Since those early days, there have been numerous developments. Among other things, decentralized and sophisticated mechanisms for resource discovery have been developed, which forms a cornerstone for almost all other peer-to-peer applications. In the remaining of this chapter we'll review the existing techniques based on overlay networks for decentralized resource discovery.

In Section 2.2 we'll introduce the concept of overlays and how an unstructured overlay can be used for resource discovery. Indexing is a well-known technique used for efficient and accurate search. In Section 2.3 we explain how structured overlays provide a decentralized index structure, and how such an index is searched based on query routing/forwarding. We take a closer look at some of the most significant structured overlay topologies in Section 2.4. In Section 2.5 we discuss what information is stored in the overlay index structures, and the general principles of placement of redundant information in structured overlays either for fault-tolerance, load-balancing or optimization of search cost. We conclude the chapter in Section 2.6 mentioning some other applications of overlays and peer-to-peer systems.

[1] This also explains the dominance of search-engines in the world wide web!

[2] Even before the Napster phenomenon, many projects like grid computing (e.g. Seti@Home [18]) were already using end user resources. Napster and successor Gnutella captured the hearts and minds of people, as well as academia and industry, as can be seen from the proliferation of the phrase (even when sometimes it is used more as a marketing gimmick).

2.2 Overlay networks

An overlay network is a type of computer network which is built on top of another network. Nodes in the overlay can be thought of as being connected by virtual or logical links, each of which corresponds to a path, perhaps through many physical links, in the underlying network.[3]

The concept of overlays is generic. Both application specific overlay networks can be built, as well as generic substrate supporting diverse applications can be built. Moreover, overlays can be layered - one kind of overlay built on top of another. An overlay network is thus an application layer internet which disentangles the underlying (physical) layer from the applications and supports programmability and customization to meet and optimize specific functionalities.

We describe next what are called unstructured and structured overlays which are primarily used for resource discovery in a wide area network. Refer to [150] for a detailed survey on search methods in peer-to-peer networks. The same overlay networks are often used as a substrate for different communication primitives as well.

2.2.1 Unstructured overlays

In unstructured overlays, any peer can potentially be responsible for one and all resources. The Gnutella [71] network is such an example. As discussed earlier, Gnutella replaced the first-generation P2P system Napster which used a centralized index. The so called second generation (2G) P2P systems like Gnutella avoided such a centralization by completely discarding the use of an index. Instead, in Gnutella, peers would simply broadcast/flood the network with queries [162]. Because of small-world clustering characteristics of the nodes [149, 164], Gnutella like networks support fairly effective search of resources, particularly for the abundantly replicated ones.

The emergence of the small-world clustering characteristics in Gnutella like systems is a result of self-organization based on mechanisms like preferential attachment [22], however the search process itself in this network is only a distributed algorithm. This example gives a nice example of how self-organization can help the scaling of such systems, as well as the three fundamental ingredients of overlay networks which were not clearly distinguished in the early days of overlay network study or design.

Three orthogonal ingredients which together make any overlay work are the following (i) *the association of resources to peers*, (ii) *the interconnection (topology) among the peers* and (iii) *the routing process*.

This separation of concerns has time and again been exploited by future overlay network designers. In the context of unstructured overlays this is illustrated by the various steps of development the unstructured overlay networks have witnessed over the course of the last five years.

In an unstructured overlay network, originally queries were routed by flooding the network [162]. To reduce the query traffic, new ways of searching have been proposed, including random walkers [111],

[3] From the Wikipedia.

percolation based search [156] and use of bloom filters [104]. In these approaches the routing process is changed.

Alternatively, the association of resources to peers can also be changed in an unstructured network in order to facilitate better search, and it has been shown that replicating the resource at random peers proportional to the square-root of the resources' popularity in conjunction with random walkers based search is optimal [111] in terms of reducing search cost and latency.

Finally, changing the topology itself can also help. Hierarchical (super/ultra-peers) topologies are used in many of the deployed file-sharing networks including the current version of Gnutella network. Local dynamic topology adaptations to spontaneously create semantic communities, so that most queries can be answered locally within such a subnetwork has been studied in [36]. Another topology adaptation mechanism builds a square-root topology [41] where the node degrees for each peer adapts proportional to the square-root of the number of queries (popularity) for the resources it stores. All these examples show how changing one or several of the ingredients that together make-up the overlay network - placement of resources, connectivity among peers and routing mechanism, one can improve performance, i.e., reduce search latency, increase recall or reduce messaging costs for search.

Research on unstructured overlays is still very active, but this is of only peripheral interest to us. Instead we'll focus on what have come to be known as *structured overlays*.

2.3 Structured overlays

In recent years the concept of structured overlays[4] has attracted a lot of attention because of its potential to become a generic substrate for internet scale applications - used for applications as diverse as locating resources in a wide area network in a decentralized manner, address independent and robust and flexible (group) communication - e.g., application layer multicast and internet indirection infrastructure and content distribution network to name a few.

The basic function of the structured overlay is to act as a decentralized index. To that end, for each resource, a globally unique identifier (called the key) is generated using some function suitable to the applications that are supposed to use the index. The codomain (loosely speaking, range) of this function is called the key-space. For example, the key-space may be the unit interval $[0, 1]$ or an unit circle $[0, 1)$, so that the keys can be any real number between 0 and 1. The key-value pair is stored at peers responsible for the particular key. Efficient search of this key helps the applications to access the resource itself.

[4] Super-peer based topologies are often referred to as 2.5G (and by some as third generation - 3G) P2P technologies, while structured overlays are often considered as the third generation peer-to-peer technology by others. Despite such nomenclature of generations, there are wide range of other peer-to-peer systems and applications than overlays, and hence such a nomenclature using *generations* of P2P technologies is an artefact of considering P2P and file-sharing to be synonymous, which is not at all true!

Since the primary purpose of the structured overlay is to discover resources, the value corresponding to the key may well be a (set of) pointer(s) to the actual resource. See Section 2.5.1 for more details on this separation of concerns of storage and discovery. To *retrieve* an object a *query* to *search* the object is *routed* to the responsible peer by *forwarding* it through intermediate peers, hence we will interchangeably use the words: "retrieve" "query" and "search"; and "route" and "forward".

Definition 1. *Structured overlay networks comprise of the three following principal ingredients:*
(i) Partitioning of the key-space (say an interval or circle representing the real number between the range $[0, 1]$*) among peers, so that each peer is responsible for a specific key space partition. By being responsible, we mean that a peer responsible for a particular key-space partition should have all the resources (or pointers) which are mapped into keys which are in the respective key-space partition.*[5]
(ii) A graph embedding/topology among these partitions (or peers) which ensures full connectivity of the partitions, desirably even under churn (peer membership dynamics), so that any partition can be reached from any partition to any other - reliably and preferably, efficiently.
(iii) A routing algorithm which enables the traversal of messages (query forwarding), in order to complete specific search requests (for keys).

A special class of structured overlays are the *distributed hash tables* (DHTs), where the keys are generated from the resources (name or content) using uniform hashing, e.g., SHA-1 (Secure Hash Algorithm [91]).

A structured overlay network thus needs to meet two goals to be functionally correct:
(i) *Correctness of routing*: Starting from any peer, it should be possible to reach the correct peer(s) which are responsible for a specific resource (key).
(ii) *Correctness and completeness of keys-to-peers binding*: Any and all peers responsible for a particular key-space partition should have all the corresponding keys/values.

Correctness of routing in structured overlays is achieved by maintaining the peers' routing tables correctly and using a proper routing algorithm. Correctness and completeness of binding is achieved by moving the corresponding keys (content) as and when the partition a particular peer is responsible for changes, and synchronizing the content among replica peers.

Various applications can use transparently the (dynamic) binding between peers and their corresponding key-space partitions as provided by the overlay for resource discovery and communication purposes in a wide area network.

One of the most important and distinguishing aspect of structured overlays is the peers' interconnection - the topology/geometry of the network.

[5] It is also possible that keys are not strictly associated with a specific peer and instead has a looser coupling. For example, in Freenet [39, 38], this association of keys to peers can be thought to be in a best effort fashion, such that instead of choosing the peer which is globally the closest to a key, the locally closest peer is delegated the responsibility of the key. Such a relatively loose coupling in Freenet was because its main objective was anonymous content distribution. Such systems are called semi-structured overlays.

How this topology is established in a dynamic setting, and whether it achieves some other properties (like - proximity and low stretch exploiting information from the underlying networking layer, load-balancing, security against various attacks, etcetera.) and how the invariants of the topology maintained over time in presence of membership dynamics and attacks are some of the most interesting questions that have been investigated in the P2P research community in these last years.

The graph obtained from these interconnections can be studied for its diameter - determining the worst case latency subject to use of an optimal routing mechanism, cut set - to determine its resilience, degree distribution - for determining some aspects of load-balancing and topology maintenance cost incurred by peers, to name a few implications of the topology. These properties include, among others, fundamental tradeoffs between routing table size and network diameter [173].

2.4 A taxonomy of structured overlay topologies

Next we briefly look into some of the important topologies. These are not necessarily the optimal topologies in the sense of achieving the smallest routing table size (constant, e.g., de Bruijn networks [93, 124]) or smallest diameter (constant, e.g., Kelips [77] and OneHop [76]), however have proven to be practical because of their overall characteristics. These systems have typically moderately small average routing table sizes which provide good resilience at reasonable maintenance cost, small diameter, good degree-distribution (congestion-free/load-balanced) and flexibility to deal with different kind of workloads, and last but not the least, they are also relatively simple. The complexity of the topology can play an important role in a peer-to-peer setting, where the topology invariants need to be established and maintained without global knowledge and coordination in presence of potentially high membership dynamics.

2.4.1 Ring

The ring based topology was pioneered in the context of overlays in the Chord [163] network. Chord uses SHA-1 based consistent hashing to generate an m-bit identifier for each peer p, which is mapped onto a circular identifier space (key-space).

Irrespective of how the peers' identifiers are generated in a ring based topology, what is essential is that the peer identifiers are distinct. Similarly, unique keys are generated corresponding to each resource. Each key on the key-space is mapped to the peer with the least identifier greater or equal to the key, and this peer is called the key's successor. Thus to say, this peer is responsible for the corresponding resource.

What is relevant for our study is how keys from the key-space are associated with some peer(s) and how the peers are interconnected (in a ring) and communicate among themselves.

A ring network is (1) weakly stable if, for all nodes p, we have $predecessor(successor(p)) = p$; (2) strongly stable if, in addition, there exists no peer s on the identifier space where $p < s < q$ where $successor(p) = q$; and (3) loopy if it is weakly but not strongly stable.

Condition (2) that there exists no peer s on the identifier space where $p < s < q$ if p and q know each other as mutual successor and predecessor determines the correctness of the ring structure. Figure 2.1(a) shows one such consistent ring structure (peer's position in the ring and its routing table). The order-1 successor known also just as "successor" of each peer is the peer closest (clock-wise) on the key-space.

If at any time such a s joins the system, the successor and predecessor information needs to be corrected at each of p, q and s. Maintaining the ring is basically to maintain the correctness of successors for all peers - this in turn provides the functional correctness of the overlay - i.e., successor peer for any identifier key can be reached from any other peer in the system (by traversing the ring). For redundancy, f_s consecutive successors of each peer are typically maintained, so that the ring invariant is violated only when any f_s consecutive peers in the identifier space all depart the system before a ring maintenance mechanism - Chord's self-stabilization algorithm - can amend for the changes.

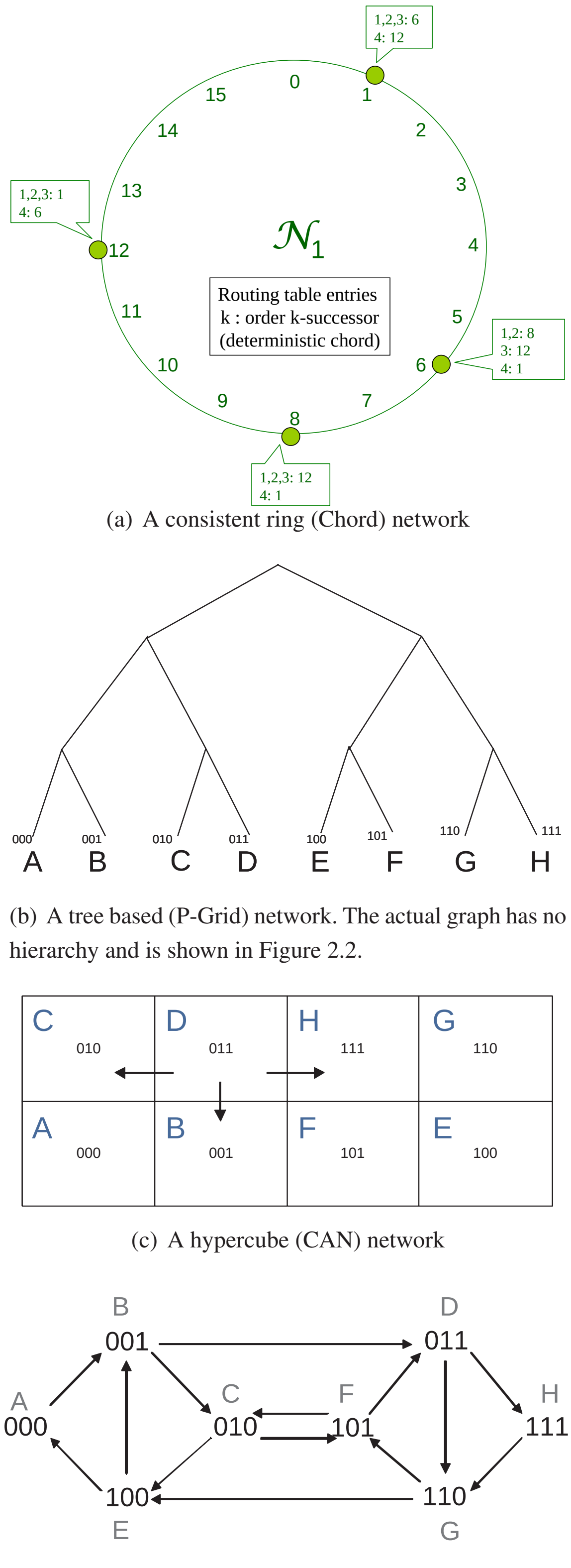
In addition to the successor/predecessor information, each peer maintains routing information to some other distant peers in order to reduce the communication cost and latency.

It is the way these long ranges are chosen which differ in many ring topology networks and has no critical impact on the functional correctness of the overlay. Distance in such ring based topologies is generally measured in terms of the absolute difference of the two concerned points on the key-space, but other metrics can as well be used. For the real topology, devoid of the artificial distance metrics, the long ranges are essentially to halve the number of peers (the "true" distance on a ring traversed sequentially) between the current peer and the destination peer [69].

Explicitly or implicitly, most variants of the ring topology exploit this fact and reduce the distance geometrically - either deterministically or probabilistically. The original Chord proposal advocated the deterministic use of the successor of the identifier $(p + 2^{k-1})$ modulo 2^m as an order-k successor of peer p or a finger table entry. Many other variants for choosing the long range links exist - e.g., randomized choice from the interval $[p+2^{k-1}, p+2^k)$ or exploiting small-world [98] topology [119, 26, 69, 105], or emulating Skip-Graphs [20, 83]. Other systems follow the same topology but uses different maintenance mechanisms [16].

The maintenance of the ring (strong stability) is critical for functional correctness of the routing process in ring based topologies. The ring invariant is typically violated when new peers join the network, or existing ones leave it. If such events occur simultaneously at disjoint parts of the ring, the ring invariant can easily be reestablished using local interactions among the affected peers. The self-stabilization mechanisms proposed in the original Chord proposal [163] exhaustively deals with the maintenance of the ring, and all other ring based topologies rely on similar mechanisms. It has been shown that the ring topology has better static resilience[6] than other topologies because of the greater flexibility to choose both routing table entries to instantiate the overlay, as well as to choose from multiple routes to forward a query at run time.

[6] Static resilience will be formally defined in Section 8.4.



(a) A consistent ring (Chord) network

(b) A tree based (P-Grid) network. The actual graph has no hierarchy and is shown in Figure 2.2.

(c) A hypercube (CAN) network

(d) A de Bruijn network

Fig. 2.1. Some structured overlay topologies

2.4.2 Tree

Arguably the earliest approach to locate objects in a distributed environment - the PRR [132, 133] scheme used a tree structure where searches were forwarded based on longest prefix matching. Tapestry [175], Pastry [154] (also uses the ring as a fall-back mechanism) and P-Grid [1] shown in Figure 2.1(b) among others [11, 116] uses similar prefix resolution in order to forward search operations, and has the tree topology. The leaf-nodes of the tree represent the key-space partitions (peers). The (maximum) distance between these partitions when the query is resolved based on prefix is then the height of the common subtree. Kademlia [120] resembles the tree structure and peers have the same routing choices as other tree-based networks. Despite having the same topology, Kademlia routing uses the XOR distance between the peer identifiers (essentially the binary string representing the node's path in the tree) instead of resolving common prefix.

Note that this also exemplifies the essential orthogonality of the topology itself from the routing strategy - the same graph connectivity can be explored based on different routing schemes, and thus defined as separate ingredients of a structured overlay network in Definition 1.

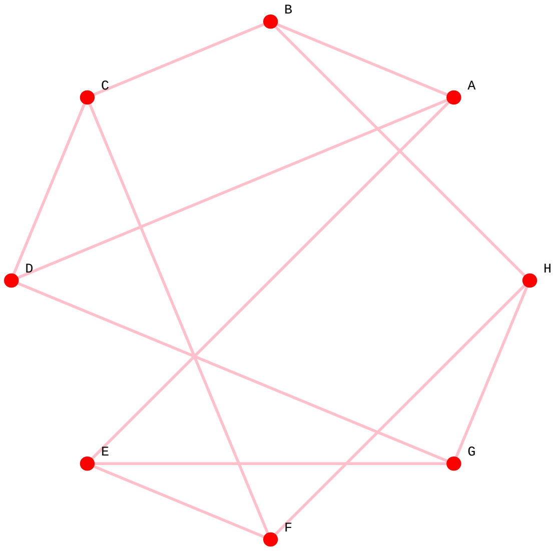


Fig. 2.2. The actual P-Grid connectivity graph does not have any hierarchy. The routes are randomly chosen from complimentary sub-trees of Figure 2.1(b). The basic P-Grid graph is directional, however since each link establishment and maintenance cost is the same, and from the symmetry of the routing choices, the actual P-Grid uses bidirectional routes.

For each level in a tree topology there are several choices to select routing table entries. However, it is essential for each peer to maintain at least one usable link for each possible level - which means that a tree based topology has a relatively lower static-resilience [75] than a ring based topology.

Since the structured overlay (P-Grid) network we exhaustively study in this thesis is tree-structured, we'll delve into further details of the tree structure later in Part II.

2.4.3 Hypercube

One of its kind, CAN [140] views the key-space as a d-dimensional Cartesian coordinate space d-torus. In Figure 2.1(c) a simple 2 dimensional CAN network is shown. Each partition is some distinct zone of this d-dimensional space, and maintains links to all the neighboring zones. In this example CAN network, a peer in zone $D(011)$ maintains routes to $C(010)$, $B(001)$ and $H(111)$. The binary string representation of the zones for this toy-example has been used to facilitate comparison with some other networks which use binary strings to identify peers/partitions. Keys are mapped to a point in the space, and routing is greedy - trying to reduce distance in any possible dimension. Hence, for our example of CAN network if D has to search the key 000, it can forward it to either of C or B to approach closer to the destination in one of the two possible dimensions. Thus peers have flexibility in choosing routes, however the choice of routing table entries is restricted to the immediate neighbors, which is undesirable for the resilience of the network under membership dynamics [75].

2.4.4 Others

Constant hop overlays. Latency in discovering resources is a critical metric. From the latency perspective, discovering the desired resource in a single hop will be ideal.

OneHop [76] maintains the full system state, i.e., information about all peers in order to provide single hop look-up. Kelips [77] uses $O(\sqrt{N})$ states to provide $O(1)$ 2-hop latency. Beehive [137] uses aggressive caching of the content itself on a Pastry overlay in order to provide $O(1)$ lookup.

Apart Kelips which scales to moderate network sizes, the other proposals, despite scalability claims don't really scale.

Caching [137] to provide constant lookup is definitely both bandwidth and storage intensive for various kinds of workloads, and is not practical.

Maintenance of full state [76] may be very expensive in terms of bandwidth consumption. The Accordion [105] system proposes a nice and practical mechanism where the size of the routing state at each peer is decided autonomously based on its local bandwidth budget. If all peers have high enough bandwidth budget to maintain full state, Accordion achieves constant hop lookups.

Constant degree networks. Node degrees in a routing network reflects the amount of system state information each peer needs to maintain. This becomes critical particularly if the system is dynamic, because the more systems states a peer has, more the maintenance cost. The first structured overlays, like Chord, P-Grid, Pastry (and CAN) used logarithmic routing table sizes to achieve logarithmic latency. However, a simple back of the envelope calculation tells us that a constant degree per node should be sufficient to have logarithmic diameter graphs, and hence, with proper routing algorithm achieve logarithmic latency.

CAN [140] has a constant routing table size. In a d-dimensional space CAN has $2d$ neighbors. However search cost in CAN scales according to $O(N^{1/d})$, and logarithmic search cost is achieved only when $d = (log_2N)/2$.

One of the earliest constant degree logarithmic latency overlay proposed was the Viceroy [113] overlay which emulates a Butterfly graph. Apart from being a fairly complex topology which makes it hard to realize in a highly dynamic system without global knowledge, the constants involved in the $log(N)$ routing is large.

Several overlays inspired by de Bruijn graphs [93, 124] also achieve logarithmic diameter with constant node degree. de Bruijn graph based networks have lower diameter than Butterfly networks.

Nonetheless, constant degree networks have poor or no flexibility of choosing alternative routing table entries nor redundant routes from one peer to another - and thus have poor resilience [75] against systems' dynamics.

Such constant degree networks, apart from being fragile against network dynamics, also lack the flexibility to accommodate heavily skewed load-distributions, as has been shown for de Bruijn networks [49]. Early DHT proposals ignored the possibility of skewed load-distributions, but increasingly the realization has dawned that it is necessary to deal with skewed load-distributions in order to support non-exact (complex) queries like range queries [26, 52, 63] and similarity queries [95].

Symphony [119] (and Accordion [105], which extends the ideas in Symphony) realizes a Kleinberg [98] style small-world using constant number of long-range links in order to achieve poly-logarithmic latency. Mercury [26] heuristically exploits similar small-world routes for skewed-load distributions [69].

As an aside, without getting into the details of de Bruijn graph, we show in Figure 2.1(d) how the same key-space partitions which were interconnected according to the tree topology (Figure 2.1(b)) and the hypercube topology (Figure 2.1(c)) can also be connected in yet another different topology. This exemplifies how the key-space partitioning and the interconnection between these partitions are essentially orthogonal issues, and thus defined as separate ingredients of a structured overlay network in Definition 1.

Hierarchical structured overlays. Often in distributed systems, hierarchy has been used for scalability. Implicit [72] or explicit [64, 122] hierarchy to exploit heterogeneity and delegate some peers to manage larger key-space partitions than a homogeneous model where each peer is treated to have same physical resources can significantly improve performance in terms of search latency. Other variants of hierarchy may involve a super-peer architecture where relatively stable peers participate in the structured overlay, while peers which are more unreliable do not participate in the indexing process (which is what the overlay is essentially about), and instead act more as parasites. Such free-riding may actually be beneficial for all participants as well as the system as a whole in terms of resource utilization since it will reduce the membership dynamics and the associated maintenance cost of the structured overlay. Prudent decision on how to realize such hierarchical or hybrid systems is still an interesting and outstanding problem in the area.

2.5 What is stored in structured overlays? Where is it stored?

2.5.1 Index vs. Storage: Separation of concerns

A structured overlay is essentially an indexing mechanism to discover resources in a distributed system in a decentralized manner. So the overlay in principle will store a (set of) pointer(s) to the actual resource. This resource may be any object (files) or service or information. However, traditionally many systems [46, 153] also store the actual object at the overlay replicas responsible for the corresponding key. We'd like to emphasize the separation of concerns of discovery and storage.

Under this notion of separation of concerns, placement of the resources at the peers responsible to index the corresponding keys becomes a special case of resource placement strategy.

There are several advantages of such a separation of concerns, since it gives users/applications control over the placement of the actual resources. Following is a non-exclusive list of such advantages:

(i) *Access control:* Owner of a resource may like to impose certain access controls over the resource, or even simply gather information about its usage pattern.

(ii) *Locality:* The resource may be placed based on the locality with respect to the end users in order to enhance performance. For instance, if a particular resource is frequently accessed from within an organizational domain, it is desirable to keep a local copy of the resource.

(iii) *Priority based redundancy:* The application or end users can determine which resource needs to be stored with what redundancy - so as to achieve resource dependent fault-tolerance and availability.

(iv) *Accountability and freedom of choice:* Users have the choice to store (and share) only resources that they want to share. In the context of file-sharing networks, this may be important both because users will like to store only files they are interested in, and the users are also accountable for the content they share. Moreover, the greatest amount of bandwidth and storage will be consumed by the actual resources, and hence such a freedom of choice to end users gives an automatic mechanism to balance load over the system.

(v) *Multi-key access:* Disentangling the index from the storage also means, one can build multiple indices for different attributes, say for performing more sophisticated multi-attribute searches [26, 65].

There is however a caveat. Using the overlay just as an index is cost effective in terms of bandwidth and storage only if the storage due to location-pointers is significantly less than the storage due to the resource itself [168].

At this juncture, it is also interesting to point out that depending on how the actual resources are stored in the system, there are two different notions of replicas. There are overlay replicas to provide redundancy to the index. The structural replicas in P-Grid, or a predefined number of adjacent peers in a ring fall into this category. There is also replication of the resource itself (managed by the applications). Hence it'll be necessary to update both kinds of replicas - replicas of the overlay as well as replicas of the resources themselves.

2.5.2 A taxonomy of replication in structured overlays

Following the principle of separation of concerns between indexing and storage, the replication of the actual resource will be done at the application layer. However, in order to enhance fault-tolerance, balance load and deal with hot-spots as well as reduce search latency, the indexed information in the overlay is also replicated. We can distinguish six general replication strategies in structured overlays:

Structural replication: In this replication scheme, multiple peers can be responsible for the same key space partition. This strategy can be seen as exploiting multiple identical non-replicated overlay networks superimposed on each other, where the routing choices are made randomly (or based on other considerations like proximity) from several choices of peers belonging to the same key space partitions. Such replication is called zone overloading in CAN [140] or simply replication of the tree leaves in P-Grid [8]. Symmetric replication [67] is a generalization of structural replication, where multiple peers are responsible for same set of key-space partitions.

Uniform structural replication for any topology is optimal with respect to search latency (and cost)[7]. However, such a placement strategy ceases to be optimal if different items are replicated with different frequency, and as a consequence, structural replication is not a suitable placement strategy for query-adaptive replication in general.

Constrained replication at peers with closest ID: In this scheme, keys are replicated at a globally fixed number of immediate successors of the peer responsible for a key for fault tolerance. This is the strategy that has typically been used in Chord based CFS [46] and the Chord variant DKS(n,f,k) [16]. All queries are typically routed to the primary replica, i.e., the one peer originally responsible for the key being queried, while other peers act purely as back up for fault tolerance. Because of this routing behavior, the search process does not exploit the wider availability of the resource for reducing the search cost or distribute query load. On the other hand, an advantage of this replica placement strategy is that updates are easy to perform, since all replicas can be deterministically located and there is a natural choice of a primary replica[8].

Replication along the query path: Replication can be done along the path used by a previous query. This strategy, apart from yielding increased redundancy, fault tolerance and adaptivity to queries, has the additional benefit that future queries can potentially be answered based on the cached information resulting in improved search performance. This is the strategy used in semi-structured networks like Freenet [39]. The drawbacks of this scheme are that, it does not completely exploit the structure of the network, so while

[7] By uniform we mean that each partition, and hence each item is equally replicated. The replica placement optimality criterion will be discussed in Section 5.3.

[8] Even if pointers instead of actual content is stored and replicated, these pointers may need to be updated, possibly with information on new copies of the actual content.

the query performance improves, it is not necessarily optimal. Additionally, since the scheme is inherently heuristic and non-deterministic, the replicas can not be located in the network efficiently and exhaustively, and hence replication along query path conflicts with effective updates of replicas.

Caching at querying peers: This strategy neither exploits the structure of the overlay at all to reduce search cost or distribute load, nor are replicas easily locatable for updates. A peer repeating a query however can find the resource locally.

Replication at least loaded peers: A recent load adaptive replication proposal (LAR [73]) places replicas purely based on peer load information gathered using sampling mechanisms, and can potentially create replicas at any nodes in the system. This requires modification of original DHT routing, leading to a different and more complex routing mechanism, based on disseminated information about replicas. This replication scheme fails to exploit the properties of the original structure of the network to reduce search cost and also loses the deterministic nature of the replica location in the network, which makes updates difficult. That apart, such a load-balancing strategy does not have a separation of concerns between routing structure and query-load balancing and loses the simplicity and efficiency of routing in structured overlays.

Optimal placement strategy: In general, different topologies will require different placement strategies, in order to optimize expected search cost. An interesting observation is that structural replication is an optimal placement strategy independent of the topology involved, provided every item is equally replicated. However, query-adaptive replication implies different items are replicated differently, needing different placement strategies. Later (Chapter 5) we'll explore optimal query-adaptive replica placement strategies for some important classes of overlay networks including generic tree structured overlay network [11] which subsumes Hypercubes and de Bruijn networks, and other tree abstracted networks like P-Grid, Pastry and XOR topology based Kademlia.

2.6 Conclusion

Apart overlays for resource discovery which are also used as a substrate for various other communication mechanisms, there also exist overlay networks built for specific purposes like the Virtual Private Network (VPN) overlay, peer-to-peer VoIP voice communication [160], anonymous communication [55] and content distribution networks [62] to name a few.

Besides resource discovery, sharing and dissemination, and communication primitives, there are various other applications of peer-to-peer systems, including storage systems [25, 80, 145, 103] and backup systems [42], distributing computational task [18, 37], peer-to-peer information retrieval [165] and peer data management systems [3, 81] based on semantic overlays [44] among others.

To conclude, we'd like to point out that the use of the peer-to-peer paradigm is not necessarily restricted to use in a completely open internet setting. Such a paradigm may well be used within a single organization or a group of relatively trusted organizations or within a close knit social community - which want to autonomously pool and use their collective available resources.

For instance, a geographically spread organization may use its organization's desktop PCs to realize a cost-effective back-up solution which will even survive physical destruction of one location, and in a more cost-effective manner than the use of hardware backup (e.g., tapes) since a peer-to-peer solution will eliminate both the material cost as well as reduce administrative overheads.

One can also find peer-to-peer systems resulting from collaboration of established organizations to pool their resources together, under a single administrative supervision, e.g., PlanetLab [37].

Depending on the deployment setting, there may be huge differences in the environment's characteristics - in terms of dynamics in the system, resource constraints and heterogeneity of the participants to name some of the most dominant factors. It is thus desirable to design systems which can be universally deployed without a priori assumptions on the environment characteristics, and still adapt to the specific environment to provide the desired functionalities reliably and efficiency, making judicious use of available resources without the need to manually configure and optimize the system parameters.

In fact, some of the self-organizational substrates we develop focus particularly on how to not only provide resilience against a wide degree of dynamicity, but also do so in an adaptive manner so that the mechanisms are efficient. By adaptive we mean that we design mechanisms which neither need a priori over provisioning nor render the systems vulnerable against increased dynamics. By efficient we mean that in a stable environment (e.g., when there are no membership changes), the maintenance cost will be minimal while the cost will gracefully increase with the increase in systems dynamics. Such maintenance principles are observed both for maintenance of structured overlays (Chapter 6) as well as maintenance of redundancy in storage systems (Chapter 8).

Part II

Self-organizing overlay substrate

3. The P-Grid overlay network

"One day Alice came to a fork in the road and saw a Cheshire cat in a tree. Which road do I take? she asked. Where do you want to go? was his response. I don't know, Alice answered. Then, said the cat, it doesn't matter." — Lewis Carroll, Alice's Adventures in Wonderland

3.1 Beyond DHTs

In the early days of structured overlay network research, where the overlays were designed primarily with network oriented applications in mind, a simple data structure, hash table, provided both the efficiency in locating a peer responsible for a specific key (exact queries), as well as provided good load-balancing among peers using uniform hashing, thus creating an uniform load-distribution over the key-space. However these distributed hash tables (DHT) as they have come to be known as are not well suited for data-centric applications which involve complex queries (e.g., range queries).

It is worthwhile to point out that even for exact key queries, DHTs typically need $O(log(N))$ communication (messages and latency) among peers in a population of N peers, in contrast to the hash table data structure which provides constant time look-up of a specific key. Thus, the primary advantage of constant time lookup in memory using a hash table is not valid when this hash table is distributed at multiple sites. Given the logarithmic search cost in terms of number of participants in typical DHTs, it is thus natural to ask if we can implement a more appropriate data structure in a distributed manner, which has logarithmic search cost anyway (and retaining its logarithmic search cost, making it comparable in performance to DHTs), but provides a better indexing, by maintaining a sorted data-structure.

We propose a data-structure, P-Grid, which abstracts a trie (prefix tree) which has similar efficiency as DHTs when it comes to exact key search. Using an order-preserving hashing to generate binary keys from natural language keywords, P-Grid supports complex queries like range queries also. Using order-preserving hashing however leads to skewed load-distribution over the key-space. Thus the key-space needs to be partitioned in different granularity depending on load in order to have load-balancing among the partitions, and hence the peers. Figure 3.1 shows a simple example of such a load-dependent partitioning and a P-Grid routing network for such a partitioning of the key-space.

Next, in Section 3.2 we provide a formal description of the P-Grid data-structure [1] along with a recapitulation of the proof [2] that locating a specific key requires on an average $O(log(|\Pi|))$ in terms of

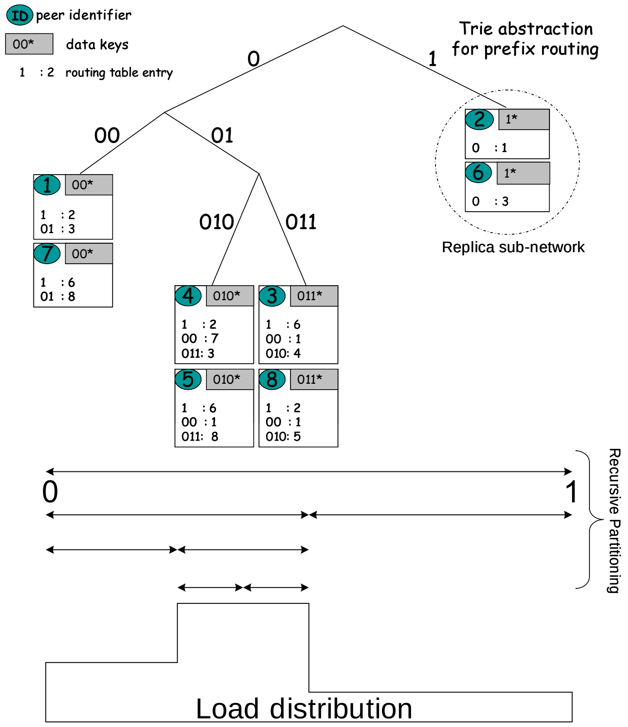


Fig. 3.1. P-Grid structure: Key-space is partitioned in a granularity adaptive to load-skew. In this example peers p_1 and p_7 are structural replicas for the partition for prefix 00. Peer p_1 has reference to peer p_2 for prefix 1, and to peer p_3 for prefix 01. Peer p_7 stores the same keys as peer p_1 (replicas), however they can and do have different routing table entries. In practice, for each level, each peer will also maintain multiple references primarily in order to have some fault-tolerance. Thus peer p_1 would also refer to some of p_3, p_5 or p_8 for the prefix 01.

the total number of key-space partitions $|\Pi|$, irrespective of the granularity of these partitions. That is to say, even if the trie-structure abstracted by P-Grid is unbalanced, searches stay efficient under the premise that communication cost is the dominant and deciding factor in large-scale distributed access structures. In Section 3.3 we describe two range query algorithms that can be used on the P-Grid overlay.

In Section 3.4 we list some contemporary complimentary ides which can be incorporated in P-Grid to enhance performance before concluding in Section 3.5.

In this chapter, we assume that the P-Grid network already exists. Subsequently, in Chapter 4 we'll provide details of how such a load-balanced P-Grid network is formed in a decentralized manner and in Chapter 6 we'll show how such an overlay network can be sustained over time even in presence of membership dynamics.

3.2 The P-Grid overlay network

P-Grid divides the key-space in mutually exclusive partitions so that the partitions may be represented as a prefix-free set $\mathit{\Pi} \subseteq \{0,1\}^*$. Stored data items are identified by keys in $\mathcal{K} \subseteq \{0,1\}^*$. We assume that all keys have length that is at least the maximal length of the elements in $\mathit{\Pi}$, i.e.,

$$\min_{k \in \mathcal{K}} |k| \geq \max_{\pi \in \mathit{\Pi}} |\pi| = \pi_{max}$$

Each key belongs uniquely to one partition because of the fact that the partitions are mutually exclusive, that is, different elements in $\mathit{\Pi}$ are not in a prefix relationship, and thus define a radix-exchange trie.

$$\pi, \pi' \in \mathit{\Pi} \Rightarrow \pi \nsubseteq \pi' \wedge \pi' \nsubseteq \pi$$

where $\pi \subseteq \pi'$ denotes the prefix relationship. These partitions also exhaust the key-space, so to say, the key-space is completely covered by these partitions so that each key belongs to one and only one (because of exclusivity) partition.

In P-Grid each peer $p \in P$ is associated with a leaf of the binary tree, and each leaf has at least one peer associated to itself. Each leaf corresponds to a binary string $\pi \in \mathit{\Pi}$, also called the *key-space partition*. Thus each peer p is associated with a path $\pi(p)$. For search, the peer stores for each prefix $\pi(p,l)$ of $\pi(p)$ of length l a set of references $\rho(p,l)$ to peers q with property $\overline{\pi(p,l)} = \pi(q,l)$, where $\overline{\pi}$ is the binary string π with the last bit inverted. This means that at each level of the tree the peer has references to some other peers that do not pertain to the peer's subtree at that level which enables the implementation of prefix routing for efficient search. The whole routing table at peer p is then represented as $\rho(p)$. Moreover, the actual instance of the P-Grid is determined by the randomized choices made at each peer for each level out of a much larger combination of choices. The cost for storing the references and the associated maintenance cost scale as they are bounded by the depth of the underlying binary tree. This also bounds the search time and communication cost.

Each peer stores a set of data items $\delta(p)$. For $d \in \delta(p)$ the binary key $\kappa(d)$ is calculated using an order-preserving hash function. $\kappa(d)$ has $\pi(p)$ as prefix but it is not excluded that temporarily also other data items are stored at a peer, that is, the set $\delta(p, \pi(p))$ of data items whose key matches $\pi(p)$ can be a proper subset of $\delta(p)$. Moreover, for fault-tolerance, query load-balancing and hot-spot handling, multiple peers are associated with the same key-space partition (structural replication). $\Re(\kappa)$ represents the set of peers replicating the object corresponding to key κ. Peers additionally also maintain references to peers with the same path, i.e., their replicas $\Re(\pi(p))$, and use epidemic algorithms to maintain replica consistency.

P-Grid's hash function maps application keys to binary strings. In the reference implementation we assume application keys to be strings for simplicity, but in fact any data type can be used. The hash function is order-reserving, i.e., it satisfies the following property for two input strings s_1 and s_2:

$$s_1 \subseteq s_2 \;\Rightarrow\; \kappa(s_1) \subseteq \kappa(s_2)$$

where $\subseteq$ means *is-prefix-of.*

P-Grid not only supports the two basic operations of DHTs: *Retrieve(key)* for searching a certain key and retrieving the associated data item and *Insert(key, value)* for storing new data items, but also range queries for any given range $\mathcal{R}$. DHTs often use Get()/Put() instead of Retrieve()/Insert(), but that is purely a syntactical difference [147].

Retrieve(key) is of complexity $O(\log |\varPi|)$, measured in messages required for resolving a search request, in a balanced tree, i.e., all paths associated with peers are of equal length. Skewed data distributions may imbalance the tree, so that it may seem that search cost may become non-logarithmic in the number of messages. However, as we'll prove next, due to the randomized choice of routing references from the complimentary sub-tree, the expected search cost remains logarithmic $(0.5 \log |\varPi|)$, independently of how the P-Grid is structured. The intuition why this works is that in search operations keys are not resolved bit-wise but in larger blocks thus the search costs remain logarithmic in terms of messages. This is important as P-Grid uses order-preserving hashing to compute keys which may lead to non-uniform key distributions.

The search uses *greedy routing* and is shown in distributed Algorithm 1. p in the algorithm denotes the peer that currently processes the request. The *Insert* operation uses the same routing mechanism to locate a peer responsible for the key to be inserted.

Algorithm 1 Search in P-Grid: Retrieve(κ, p)

```
if $\pi(p) \subseteq \kappa$ i.e., $p \in \Re(\kappa)$ then
    return($d \in \delta(p) | \kappa(d) = \kappa$);
else
    determine $l$ such that $\pi(\kappa, l) = \overline{\pi(p, l)}$;
    $p'$ = randomly selected element from $\rho(p, l)$;
    Retrieve($\kappa$, $p'$);
end if
```

Since routing cost is basically the cost of traversal within the key-space partitions, without loss of generality, to determine the routing cost we'll assume that there is one and only one peer associated with each partition, and only one random routing entry at each level of each peer. The additional redundancy based on structural replication provides scopes for optimizations which we overlook in the following proof. Under this premise a peer and its key-space partition are equivalent, and we'll use them synonymously, i.e., $p \equiv \pi(p) \equiv \pi$ in Section 3.2.1 for the following proof. We'll however distinguish peers from partitions in rest of this thesis unless otherwise stated.

3.2.1 Average Search Cost Analysis

Definition 2. *A random instance of a P-Grid $P \in \mathcal{P}_{\Pi}$ is a mapping determined by the random and mutually independent routing choices at each peer (partition) for each level, $\rho_{\Pi}^{P}(\pi, l) : \Pi \to \Pi$.*

Definition 3. *The length of a specific $\kappa \in \Pi$ is given as $|\kappa|$. We define $\Pi_j^{\kappa} \subseteq \Pi$ s.t. $x \in \Pi_j^{\kappa} \Leftrightarrow \pi(x, j) = \pi(\kappa, j)$.*
We define $\Pi_{j-}^{\kappa} \subseteq \Pi$ s.t. $x \in \Pi_{j-}^{\kappa} \Leftrightarrow \overline{\pi(x, j)} = \pi(\kappa, j)$.
We represent the cardinalities as follows: $|\Pi_j^{\kappa}| = n_j^{\kappa}$ for $j \geq 0$ and $|\Pi_{j-}^{\kappa}| = n_{j-}^{\kappa}$ for $j > 0$.
Note that $n_0^{\kappa} = |\Pi|$, $n_{d_\kappa}^{\kappa} = 1$, $n_{j-}^{\kappa} = n_{j-1}^{\kappa} - n_j^{\kappa}$ for $j > 0$, and $\sum_{j=1}^{l} n_{j-}^{\kappa} = |\Pi| - n_l^{\kappa}$.

The definition of P-Grid does not exclude the case where the depth of the trie is up to linear in $|\Pi|$. Therefore searches can require a linear number of messages in the worst case which would make the access structure non-scalable. In the following we show that the expected average search cost is however logarithmic irrespective of the specific way Π the key space is partitioned.

Definition 4. *The* search cost *of a search in a random instance of P-Grid $P \in \mathcal{P}_{\Pi}$ for a data key $\kappa \in \Pi$ starting at $\pi \in \Pi$ is the number of invocations of the function $Retrieve(\kappa, \pi)$. We denote this cost by $\sigma_P^{\pi}(\kappa)$.*

Theorem 1. *The expected search cost $E[\sigma_P^{\pi}(\kappa)]$ for the search of a specific key $\kappa \in \Pi$ using a P-Grid $P \in \mathcal{P}_{\Pi}$, that is randomly selected among all possible P-Grids $\mathcal{P}_{\Pi}$ for a specific key-space partitioning Π, starting at a randomly selected peer (partition) $\pi \in \Pi$ is less than* $\log(|\Pi|)$.

Proof (adapted from [2]): Since the choice of routes $\rho_{\Pi}^{P}(\pi, l) \in \Pi_{l-}^{\kappa}$ are independent, the set of all possible P-Grids can be given as the product:

$$\mathcal{P}_{\Pi} = \bigotimes (\Pi_{1-}^{\kappa}, ..., \Pi_{|\pi|-}^{\kappa})$$

The references $\rho_{\Pi}^{P}(\pi, l)$ are selected with uniform probability from Π_{l-}^{κ}. Let X_l^{π} be a uniformly distributed random variable over Π_{l-}^{κ}. We use the random variable Y to represent the joint occurrence representing the choice of a random P-Grid out of $\mathcal{P}_{\Pi}$, which has the following joint probability distribution:

$$Y = \bigotimes (X_1^{\pi}, ..., X_{|\pi|}^{\pi})$$

Next we determine the probability distribution of the search cost on a P-Grid which itself is chosen according to the distribution Y. This cost depends on the length of the common prefix of the peer's key at which the search starts and the searched key κ. We denote the probability distribution for a given search key κ from a peer depending on the value of the common prefix length $l, l > 0$ as $\sigma_Y^{X_l^{\pi}}(\kappa)$.
If we define X_0^{π} as a uniformly distributed random variable over Π, then $\sigma_Y^{X_0^{\pi}}(\kappa)$ denotes the probability distribution of the cost of searches starting at a randomly selected peer. We distinguish the different classes

of peers that can be reached depending on the number of matching bits of the common prefix of the search key and the peer identifier. For $\pi \in \Pi^{\kappa}_{j-}$ the expected search cost of the remaining search is $E[\sigma_Y^{X_j^\pi}(\kappa)]$ since it starts at a randomly selected element $\rho(\pi, j) \in \Pi^{\kappa}_{j}$ determined by $\rho^P_\Pi(\pi, l)$. Thus we obtain:

$$\begin{aligned} E[\sigma_Y^{X_0^\pi}(\kappa)] &= \sum_{j=1}^{|\kappa|} Pr[\pi \in \Pi^{\kappa}_{j-}] E[\sigma_Y^{X_j^\pi}(\kappa)] \\ &= \sum_{j=1}^{|\kappa|} \frac{|\Pi^{\kappa}_{j-}|}{|\Pi|} E[\sigma_Y^{X_j^\pi}(\kappa)] = \frac{n^{\kappa}_{j-}}{|\Pi|} E[\sigma_Y^{X_j^\pi}(\kappa)] \end{aligned} \tag{3.1}$$

We obtain the above equation 3.1 under the assumption that query begins at a random peer chosen uniformly, and search is routed to a peer (out of the potential peers) chosen uniformly randomly at each step.

We proceed analogously for determining $E[\sigma_Y^{X_l^\pi}(\kappa)]$ within the relevant sub-partition of the key-space. A search starting at a randomly selected element from Π^{κ}_{l-} for $0 < l < |\kappa|$ is computed as:

$$\begin{aligned} E[\sigma_Y^{X_l^\pi}(\kappa)] &= 1 + \sum_{j=l+1}^{|\kappa|} Pr[\pi \in \Pi^{\kappa}_{j-}] E[\sigma_Y^{X_j^\pi}(\kappa)] \\ &= 1 + \sum_{j=l+1}^{|\kappa|} \frac{|\Pi^{\kappa}_{j-}|}{|\Pi^{\kappa}_{l}|} E[\sigma_Y^{X_j^\pi}(\kappa)] \end{aligned} \tag{3.2}$$

We add 1 to the expected search cost to account for the message used to reach the referenced peer.

Since $\Pi^{\kappa}_{l-} = \Pi^{\kappa}_{l-1} \setminus \Pi^{\kappa}_{l}$ we have for $j > 0$ (as stated earlier):

$$n^{\kappa}_{j-} = n^{\kappa}_{j-1} - n^{\kappa}_{j} \tag{3.3}$$

From equations 3.1, 3.2 and 3.3 we obtain(in the following we use the shorter notation E^{κ}_{l} to represent $E[\sigma_Y^{X_l^\pi}(\kappa)]$):

$$E^{\kappa}_{l} = 1 + \sum_{j=l+1}^{|\kappa|} \frac{n^{\kappa}_{j-1} - n^{\kappa}_{j}}{n^{\kappa}_{l}} E^{\kappa}_{j}, 0 \le l < |\kappa| \tag{3.4}$$

Calculating $n^{\kappa}_{l} E^{\kappa}_{l} - n^{\kappa}_{l+1} E^{\kappa}_{l+1} = (n^{\kappa}_{l} - n^{\kappa}_{l+1}) + (n^{\kappa}_{l} - n^{\kappa}_{l+1}) E^{\kappa}_{l+1}$ we obtain the following recurrence relationship for E^{κ}_{l}:

$$E^{\kappa}_{l} = \frac{n^{\kappa}_{l} - n^{\kappa}_{l+1}}{n^{\kappa}_{l}} + E^{\kappa}_{l+1}$$

We have $E^{\kappa}_{|\kappa|} = 1$. Therefore we have for the expected search cost

$$
\begin{aligned}
E_0^\kappa &= \sum_{j=0}^{|\kappa|-1} \frac{n_j^\kappa - n_{j+1}^\kappa}{n_j^\kappa} = \sum_{j=0}^{|\kappa|-1} \int_{n_{j+1}^\kappa}^{n_j^\kappa} \frac{1}{n_j^\kappa}\, dx \\
&\leq \sum_{j=0}^{|\kappa|-1} \int_{n_{j+1}^\kappa}^{n_j^\kappa} \frac{1}{x}\, dx = \int_{n_{|\kappa|}^\kappa}^{n_0^\kappa} \frac{1}{x}\, dx = \int_1^{|\Pi|} \frac{1}{x}\, dx = \log(|\Pi|)
\end{aligned}
$$

— *q.e.d.*

Theorem 2. *The probability that a search in a P-Grid $P \in \mathcal{P}_\Pi$ for a key $\kappa \in \Pi$ starting at a randomly selected peer $\pi \in \Pi$ does not succeed after k steps is smaller than $\frac{\log(|\Pi|)^{k-1}}{(k-1)!}$.*

The above theorem essentially means that the probability of a query not resolved after any given number of hops reduces very fast. This has critical implications on search performance in the P-Grid network since it essentially means that with high probability the worst case scenarios (search cost comparable to the P-Grid's depth) don't happen. The proof for the theorem can be found in [2].

3.3 Range queries: Algorithms and complexity

As described in the previous sections, P-Grid uses an order-preserving hash function. Thus the resulting P-Grid tree, apart from topologically abstracting a binary tree, also realizes a trie-index for the keys. This may lead to skewed data distributions despite which P-Grid can still guarantee logarithmic search complexity, as shown above. Order-preserving hash functions enable prefix queries and thus range queries of arbitrary granularity can be processed efficiently as well in P-Grid.We will discuss two algorithms: which we call the *min-max traversal* algorithm sequentially traverses all the partitions (one peer each from each partition) which cover the range's minimal and maximal key values, and the *shower* algorithm which parallelizes the execution of range queries.

We use the notation $\Pi_\mathcal{R}$ to represent the set of partitions which cover the contiguous range $\mathcal{R}$ in the key-space. We use $\pi_{\mathcal{R}_{min}}$ and $\pi_{\mathcal{R}_{max}}$ to represent the partitions at the periphery of this range, accounting for the keys in the lower and upper bounds of the range. We represent these keys themselves as $\kappa_{\mathcal{R}_{min}}$ and $\kappa_{\mathcal{R}_{max}}$ respectively.

3.3.1 Min-max traversal algorithm

Range queries can be processed sequentially by starting from a peer holding data items belonging to one bound of the range and forwarding the query to a peer responsible for the next partition of the key space, until a peer responsible for the other bound of the range is encountered. We call this strategy is of traversing from *min-max traversal*. The underlying data structure itself does not always have the information about peers belonging to the next neighboring key space partitions. However, such routes can be established either during the construction of the P-Grid overlay structure (algorithmically trivial), or at run-time using the existing routing information at the peers. Algorithm 2 shows the min-max traversal algorithm in pseudo-code, along with an illustration of how the algorithm works for a toy-example in Figure 3.2(a).

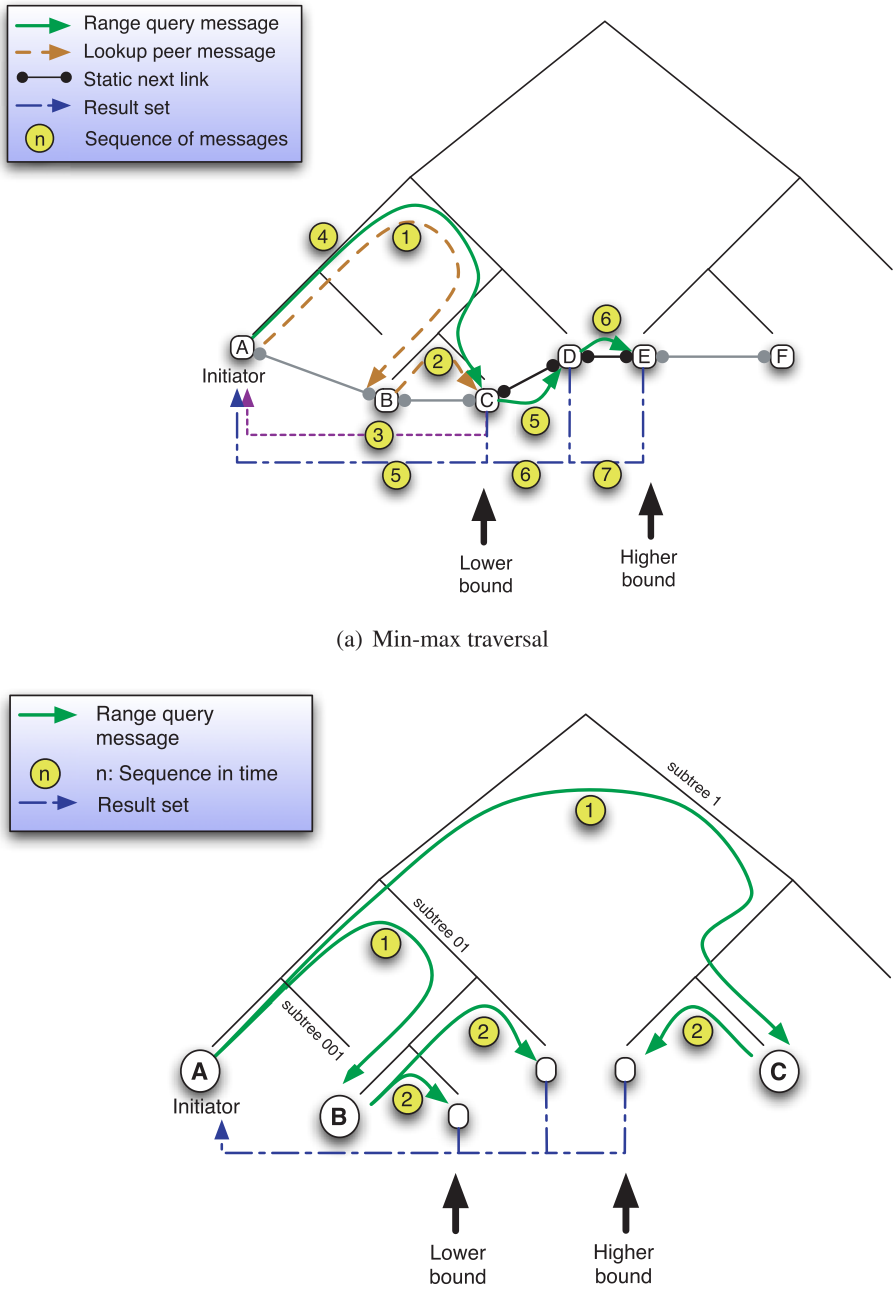
First peer A initiates the range query by querying P-Grid for the lower bound of the range which is peer C in this example. Steps (1) and (2) denote standard P-Grid routing and in step (3) the result is returned to peer A, i.e., peer C. Then in step (4) peer A sends the range query request to peer C and peer C sends its data pertaining to the interval to peer A (in the implementation steps (3), (4), and (5) are actually done in one step). Concurrently the range query is forwarded to peer D using the "next" pointer. Peer D checks whether it is in the queried range, and if yes, peer D sends its data pertaining to the interval to peer A, and concurrently forwards the range query to peer E which repeats the same operations as peer D except that it does not forward the query to another peer as it has checked that it is a peer responsible for the other bound of the queried range.

Algorithm 2 Sequential range queries: minmax($\mathcal{R}$, p)

```
if $\pi(p) \subseteq \mathcal{R}$ then
    return($d \in \delta(p) | \kappa(d) \in \mathcal{R}$);
    determine a peer $p'$ such that $p'$ is responsible for the next key space partition;
    minmax($\mathcal{R}, p'$); {Forwarded to $p'$}
end if
```

For simplifying the analysis[1] we assume that the algorithm starts at the lower bound of the range at a peer p such that $\pi(p) = \pi_{\mathcal{R}_{min}}$. It is assumed that the neighbor links are cached at each peer during the construction of the trie (this is also algorithmically trivial). In the complexity analysis of this algorithm we can assume storage load-balancing (which is achieved stochastically by the P-Grid base system) and that on average there exist $\mathcal{M}$ data items per key space partition. Then, if there is a range query for the range $\mathcal{R}$, such that there are $\mathcal{D}$ data items in the given range, search cost and latency using min-max traversal (assuming "next" links have been established during construction) is $O(\log |\Pi|) + |\Pi_{\mathcal{R}}| - 1$, where $|\Pi_{\mathcal{R}}|$ is the number of partitions over which the whole range is stored in P-Grid. The search cost and latency using min-max traversal is dependent on the size of the answer set $\mathcal{D}$ for the range query, but independent of the size of the range $\mathcal{R}$ of the query. This is because $|\Pi_{\mathcal{R}}|$ has an expected value of $\mathcal{D}/\mathcal{M}$, and in particular, using Markov's inequality, $Pr[|\Pi_{\mathcal{R}}| \geq c\mathcal{D}/\mathcal{M}] \leq \frac{1}{c}$ for any positive c thus giving already a weak bound on the deviation. We do not consider the trivial case $\mathcal{D} \leq \mathcal{M}$ as this would only affect 1 or 2 peers and concentrate on the more general case of $\mathcal{D} > \mathcal{M}$.

[1] The routing of the query to the lower bound is not shown here, but is algorithmically trivial in P-Grid. Moreover, such sequential traversal on two directions instead of one, from any partition in $\Pi_{\mathcal{R}}$ is just a simple adaptation of the algorithm shown here.



(a) Min-max traversal

(b) Shower

Fig. 3.2. Range query algorithms illustrated for a query for the range $\mathcal{R} = [0101, 100]$

As already mentioned, establishing and maintaining “next” pointers in P-Grid is algorithmically trivial and most other DHTs proactively maintain it as well. Without them, an additional overhead upper-bounded by $|\Pi_{\mathcal{R}}|O(\log |\Pi|)$ will be incurred.

3.3.2 Shower algorithm

The other variant for processing range queries is to do them concurrently. Here, the range query is first forwarded to an arbitrary peer responsible for any of the key space partitions within the range, and then the query is forwarded to the other partitions in the interval using this peer’s routing table. The process is recursive, and since the query is split in multiple queries which appear to trickle down to all the key-space partitions in the range, we call it the *shower algorithm*. The intuition of the algorithm is shown in Figure 3.2(b), while Algorithm 3 gives the pseudo code.

The essential idea is to create multiple range queries for smaller, non-intersecting ranges, i.e., with different prefixes as in steps 9-12 in Algorithm 3, and forwarding the query to some peer from the routing table, which is closer to each of the ranges. Thus, in Figure 3.2(b) for a range query $[0101, 100]$ peer A whose path is 000 forwards two range queries, one to the subtree with prefix 1 (peer C in the example), and another to the subtree with prefix 01 (peer B in the example). This is then continued recursively.

In the course of query forwarding, it is possible that the query is forwarded to a peer responsible for keys outside the range. However, it is guaranteed that this peer will forward the range query back to a (new) key-space partition within the range. This is so because the P-Grid routing ensures that no key space partition will get duplicates of the range queries.

Algorithm 3 Parallel range queries: shower($\mathcal{R}, l_{current}, p$)

```
 1: if $\pi(p) \subseteq \mathcal{R}$ then
 2:     return($d \in \delta(p) | \kappa(d) \in \mathcal{R}$);
 3: end if
 4: determine $l_l$ such that $\pi(\kappa_{\mathcal{R}_{min}}, l_l) = \overline{\pi(p, l_l)}$;
 5: determine $l_h$ such that $\pi(\kappa_{\mathcal{R}_{max}}, l_h) = \overline{\pi(p, l_h)}$;
 6: $l_{min} = max(l_{current}, min(l_l, l_h))$;
 7: $l_{max} = max(l_l, l_h)$;
 8: if $l_{current} < l_{max}$ then
 9:     for $l = l_{min}$ to $l_{max}$ do
10:         $p'$ = randomly selected element from $\rho(p, l)$;
11:         shower($\mathcal{R}, l + 1, p'$); {Forwarded to $p'$}
12:     end for
13: end if
```

The search cost (in terms of messages) of this variant is lower bounded by $O(x) + |\Pi_{\mathcal{R}}| - 1$ where x is the expected cost of reaching the subtree which contains the peers from the range being queried. The first part of the shower algorithm just tries to reach any such peer, using the normal P-Grid routing. Note that even after that, some of the branches of the shower may be directed to peers outside the range but every message created in the range sub-space reaches a different leaf node (since the sub-spaces are exclusive).

However, since these partitions outside the range still need to share the common prefix for some subtree within the range, there can't be more than 2 times the actual number of partitions which fall within the range. This gives one possible upper bound of $2|\Pi_{\mathcal{R}}|$.

If the tree is rather balanced, then a depth based upper bound is stricter that $2|\Pi_{\mathcal{R}}|$. The total key-space partitions which are in the same subtree (even if not in the range) can't be larger than $2^{Depth-x})$ since that's the scope of the shower, where $Depth$ is the maximum path length of any partition in the range. This number can however be rather large in a skewed tree.

Thus the upper bound of the number of messages for processing a range query based on the shower algorithm is $O(x) + min(2|\Pi_{\mathcal{R}}|, 2^{Depth-x})$.

Thus the complexity of the shower algorithm is again dependent only on the size of the answer set $\mathcal{D}$ for the range query, but independent of the size of the range $\mathcal{R}$ of the query.

The upper bound for latency is $O(x) + O(Depth - x)$. In particular, unlike in the sequential variant, the latency of the parallelized shower algorithm is independent of the number of data items in the range $\mathcal{R}$, but depends on the distribution of the data items (which determines the $Depth$). Note that the issuer of the query will start getting responses for part of the range with a minimum latency of $O(x)$, since it will already encounter some peer responsible for part of the range.

The expected value of x is $0.5 \log{(n\mathcal{M}/D)}$. The intuition for the value of x is that, if we increase the average memory of each logical partition to $\mathcal{D}$ instead of $\mathcal{M}$, there will be $\frac{n}{\mathcal{D}/\mathcal{M}}$ key space partitions in total, otherwise retaining the routing network's properties, and since first the query needs to reach any arbitrary peer within the range, this translates into reaching this virtual partition of average size $\mathcal{D}$, and hence x is the expected search cost in this new network, which has the same topological properties, but fewer $(n\mathcal{M}/\mathcal{D})$ partitions.

3.4 Complementary contemporary contributions

There are numerous complimentary work which the P-Grid system can use for performance improvements. We briefly explain how exactly these ideas can be integrated in the P-Grid system. However this thesis does not deal with these issues.

3.4.1 Locality

Structured overlay networks provide transparency from the underlying network, and are typically optimized to provide low number of overlay hops. However, in order to improve end-to-end latency and reduce the overall resource usage of the underlying network it is desirable to take into account physical proximity of peers in the underlying network [60, 75]. There are various ways such peer proximity may be used. The routing table entries themselves may be chosen based on proximity. This is known as proximity neighbor selection (PNS). It has been shown [75] that using PNS significantly improves end-to-end latency. P-Grid uses randomized routing entries from the whole set of peers from a complimentary sub-tree at each level, and there are multiple peers (structural replication) at each leaf-node of the tree. Thus incorporating PNS in P-Grid is straightforward.

Once the routing table is chosen, the choice of next-hop when routing to a particular destination may depend on the physical proximity of the immediate neighbors. Such a scheme is called proximity route selection (PRS). Such an assumption may be misplaced because even if the next immediate hop is chosen to be closer, without global knowledge, it may still lead to longer latencies in consequent hops, so that the improvements using PRS or PRS along with PNS are marginal [75]. Moreover such a choice of routes will not take into account more critical aspects like congestion in the system. Nonetheless, its worthwhile to point out that because of structural replication and multiple routing entries per level at each peer, PRS can also be incorporated in P-Grid, contrary to the claims in [75] that tree-topologies can't use PRS.

3.4.2 Look-ahead routing

Apart modifications in P-Grid's routing mechanism for proximity route selection or congestion control, it is also possible to use a variant of look-ahead [118] routing. The basic idea behind look-ahead routing is to maintain soft-state information about the routing table entries of a peer's routing table entries. Thus, instead of greedily forwarding a query to the routing table entry based on its proximity to the target, look-ahead forwards the query to the peer which would in the next hop lead the query closest to the target. It has been shown that in some randomized networks, using look-ahead is asymptotically optimal, requiring $\Theta(logn/loglogn)$ hops in contrast to greedy routing mechanism which requires $\Omega(logn)$ hops with high probability. Such a look-ahead based routing is an optimization technique, and occasional inaccuracies in the soft-state only results in longer than optimal number of hops, but the correctness of the routing process is not affected, and overall provides performance benefits over purely greedy routing. Even though the effects of look-ahead on routing in a key-space partitioned in a skewed manner (unbalanced tree) has not been studied, such a look-ahead based routing can be used heuristically in P-Grid to choose the specific routing entree out of the redundant ones at any level in order to reduce overlay hops and thus improve latency. This mechanism is essentially an alternative to the PRS scheme, where the proximity is being chosen on the logical key-space.

3.4.3 Abstracting k-ary trees

A more systematic mechanism that can be employed is to choose the redundant routing information in a clever way. The P-Grid structure uses r redundant entrees in each level to alleviate the poorer static resilience of tree topologies. These r entries need not be completely randomized, and instead can be such that a k-ary ($k \leq r$) tree is mapped onto the binary one. Thus, if a peer with path $0*$ needs to choose four or more redundant routing entries for prefix 1, it can as well choose entries with prefixes 100, 101, 110 and 111. In-fact some other contemporary overlays with tree-topology like Pastry uses such a k-ary tree structure in order to reduce the search latency measured in terms of overlay hops.

3.4.4 Iterative vs. Recursive processing of an isolated query

Queries in structured overlays may be resolved in two manners. It may be processed iteratively. In this case the originator of the query contacts sequentially a series of nodes (referred by the previous node) in order to reach the peer which actually stores the key being searched. A variant, which is more commonly used, and we also use in P-Grid is the recursive processing of a query. In this case, the originator of the query forwards the query to an entry in its own routing table, and then this peer forwards the query directly to an entry in its routing table, and the process recurs until the query arrives at a peer which is responsible for the queried key. This peer can either directly reply back to the query originator, or the reply may also traverse the reverse path as the query in the network. A direct reply to the query originator has several advantages like lower latency and lower load (the answer set can be huge), and is the approach used in the P-Grid implementation.

For the purpose of disambiguation, we'll like to point out here that we'll introduce the notion of recursive queries for a self-referential directory in Chapter 6, which is completely different from recursive processing of a single isolated query.

3.5 Conclusion

In this chapter we introduced an overlay network - P-Grid - which supports efficient search of individual keys as well as range of keys. In the following chapters of this part of the dissertation, we'll delve into further details of how to construct and maintain the P-Grid overlay network so that diverse kinds of load in the system are balanced.

Later, we'll show how to realize a self-referential directory service using P-Grid, and use such a self-referential directory to device a family of efficient route maintenance algorithms to maintain the overlay itself under churn (membership dynamics).

Recent results show some additional resilience properties of the P-Grid network structure, which are not discussed in this thesis. In particular, we show that originally distinct (either because of network partitions or because they were constructed separately) P-Grid networks merge into a single network gracefully in [48].

The current java based P-Grid implementation uses exclusively (but not exhaustively) algorithms proposed in this dissertation, and we'll also report on some initial experiences and experiments done with the implementation on PlanetLab.

4. Multi-faceted load-balanced overlay

“Life is like riding a bicycle. To keep your balance you must keep moving.” — Albert Einstein

4.1 Gamuts of load-balancing in structured overlays

Peer-to-peer systems in general and overlay networks in particular are instances of large-scale distributed systems. Load-balancing in distributed systems in order to judiciously and fairly use the available physical resources is both desirable and even critical. In structured overlays, there’s a multitude of aspects with respect to which the peers need to achieve load-balancing. Moreover the available resources as well as the demand of resources in a structured overlay is often very skewed, and thus the load-balancing mechanisms need to account for such skew apart from dealing with the usual vagaries of the peer-to-peer environment - very large (internet)-scale, lack of global knowledge and dynamicity.

4.1.1 Sources of load-skew

Load skew in a structured overlay network can arise mainly because of the following reasons:

Non-uniform key distribution: Keys may be distributed non-uniformly over the key-space. This is the case particularly when keys are generated using a function other than uniform hashing function (in contrast to DHTs). The simplest case when non-uniform key distributions are observed is when lexicographic ordering is preserved in order to support range-queries.

Non-uniform key usage: Access to different keys can be different. In some applications this may even vary by orders of magnitude. For instance, in data-oriented applications, often relative frequency of queries for keys have Zipf distribution.

Routing hotspots: Overlay networks resolve queries (perform resource discovery) using query forwarding. Also, communication primitives using structured overlay networks - e.g., application layer multicast - exploit the structural properties of the overlay and uses message forwarding. These messages are forwarded based on the local information at each peer (their routing tables). It may so happen - by chance or design, depending on how the local choices are made by individual peers, that most messages are routed through a small subset of peers, thus overloading these peers. Such overloading of some peers makes the system inefficient - both because the overloaded peers will become point of failures and bottleneck due to congestion, as well as because other peers' available resources stay untapped.

Participant heterogeneity: Individual peers have different physical resources that it has or is willing to devote to the system - like storage space, bandwidth and computational power.

Balls into bins effect: Suppose that m balls are thrown into n bins *sequentially* by choosing the bins independently and uniformly at random for each ball. This will mean that each bin on an average will have the same number of balls. This model has been intensively studied since it provides the basic abstraction for studying online load-balancing. We refer to such a balancing as a first-order load-balance. The variance in the number of balls per bin is still high [135]. We refer to such high statistical noise observed when load is uniformly randomly allocated as the "balls into bins" effect. It is necessary to employ some second-order mechanisms to reduce the statistical noise. By second-order balancing we mean mechanisms to reduce the variation as is observed based on solely first-order balancing. A simple and popular way of achieving such a second order load-balancing uses multiple choices. Thus multiple bins (say k) are chosen at random, and the least filled bin among these k bins is assigned the *next ball*. Power of two choices [121] is a special case of using such multiple choices.

4.1.2 Alleviating load-skew

Resource-surplus physical nodes can participate in the system as proportionally multiple *virtual peers*. The virtual peers are then considered to comprise of a homogeneous peer population. The rest of this work is under such a premise, and focusses on alleviating the effects of non-uniform key distribution and usage on peers physical resources like storage and bandwidth (implicitly also computation) by using the available resources judiciously. Given the flexibility that peers have in P-Grid in choosing the key-space they are supposed to be responsible for, virtual peers can be used such that the virtual peers running on the same physical machine cluster together in a contiguous portion of the key-space. The issue of peer heterogeneity is excluded in the rest of this dissertation, and we focus on the other aspects of load-balancing - assuming a homogeneous set of collaborative peers.

Natural language names for resources often have non-uniform distributions. Distributed Hash Tables used uniform hashing in order to generate uniform distribution of keys to be stored in the overlay. Doing so realized a first-order balancing of load. Various mechanisms have since been proposed to achieve second-order load-balancing in DHTs, including using power of two choices [32, 72, 94, 117, 138, 163]. Uniform hashing however destroys (lexicographic) locality information. Complex queries, like range queries need such information. Preserving lexicographic ordering information leads to non-uniform distribution of keys over the key-space, and several recent approaches try to achieve first-order load-balancing despite such a skew [6, 20, 26, 63].

In order to deal with *non-uniform distribution of keys* over the key-space, P-Grid partitions the key space in a granularity adapted to the local density of keys, so that the number of keys in each resulting partition do not vary much from that in other partitions [4, 6, 8].

In P-Grid each of the key-space partitions is replicated among multiple peers (structural replication). The number of peers replicating each key-space partition is determined dynamically [4, 6, 8] to judiciously use the total storage capacity of the peer population. This is in contrast to most other approaches, where replication is determined by a globally predetermined and hard-coded replication factor and lacks adaptivity either to exploit extra available resources or to deal with scarcity of resources.

If keys have different usage frequency, we deal with such *non-uniform key usage* by caching the keys. To that end, in Chapter 5 we determine the optimal number of caches for any key - depending on its relative usage with respect to other keys - and the optimal placement of the cache exploiting the structural properties of the overlay network, assuming that the system's total storage capacity is limited.

We also heuristically use standard multiple choice based variance reduction techniques to balance the in-degree at each peer in randomized routing networks (like P-Grid or other overlays based on small world routing [98]), which is necessary to avoid *routing hotspots* and balance the routing traffic in the overlay.

In Chapter 3 we had introduced a distributed data-structure (P-Grid), which can provide load-balancing and efficient queries in presence of non-uniform key distribution. In this chapter we will study how we can construct and maintain such a load-balanced overlay network. In the next chapter we will study balancing query-load in such a storage-load balanced overlay.

The rest of this chapter is organized as follows.

Most existing overlay construction mechanisms assume - either explicitly or implicitly - that the network population changes incrementally. There are compelling reasons to construct a new structured overlay network among an existing and large population of peers. We motivate the need for fast overlay construction in Section 4.2. We introduce our approach of fast load-balanced overlay construction in a parallelized manner in Section 4.3. In Section 4.4 we provide some heuristics necessary to use the fast overlay construction algorithm in practice. Apart parallelized overlay construction, we still need to accommodate newly arriving peer in an already evolved P-Grid network. We provide details to deal with sequential peer joins/leaves in Section 4.5. The sequential approach incorporates multiple choice based variance (of replication factor) reduction techniques. However, the parallelized overlay construction only achieves a first-order balancing of replication factor. In Section 4.6 we provide details of a background process for a second-order balancing used in P-Grid in order to reduce the replication factor variance. We show in Section 4.7 how the P-Grid overlay can also be used as a DHT. We conduct simulations to validate and evaluate the performance of our algorithms, and provide the results in Section 4.8. Differences with existing load-balancing techniques is discussed in the related works Section 4.9.

Later in Chapter 7 we also report on our experiences and experiments with a real implementation on the PlanetLab testbed.

At this juncture we'll like to briefly point out that, even though we have primarily been modeling the system based on storage load, the load over the key-space is essentially abstract and could have been because of anything else as well, like computational load - where small tasks are assigned to specific keys. So our load-balanced indexing scheme is more generally applicable.

4.2 Need for speed in overlay construction

In standard database systems it is common practice to regularly (re-)index attributes to meet changing requirements and optimize search performance. Recently, structured peer-to-peer overlay networks are increasingly being used as an access structure for highly distributed data-oriented applications, such as relational query processing, metadata search or information retrieval [10, 125]. Structured overlays' use was motivated by the presence of certain features that are supported by their design such as scalability, decentralized maintenance, and robustness under network churn. Compared to unstructured overlay networks which are also being proposed for these applications [81, 100], structured overlay networks additionally exhibit much lower bandwidth consumption for search as well as guarantee completeness[1] for search results.

The standard maintenance model for peer-to-peer overlay networks assumes a dynamic group of peers forming a network where peers can join and leave, essentially in a sequential manner. In addition proactive or reactive maintenance schemes are used to repair inconsistencies resulting from node and network failures or to re-balance load in order to react to data updates. These approaches to maintenance, that have been extensively studied in the literature, correspond essentially to updating database index structures in reaction to updates.

In contrast to this, almost no results exist on how to efficiently construct a large overlay network from scratch, i.e., how to bootstrap a new, large-scale, structured overlay network in a practical way within reasonable time. This is understandable insofar as most of the work on overlay networks was done under the assumption of providing an efficient resource location scheme using an application-specific, yet fairly stable, resource identifier space (e.g., file names for file sharing).

With the increasing adoption of structured overlay network technology for data-oriented applications this assumption no longer holds. Resources are identified by dynamically changing predicates and different overlay networks can be used simultaneously, each of them supporting a specific addressing need. We can illustrate these requirements by a typical application case of peer-to-peer information retrieval.

[1] In terms of information retrieval terminology, recall = 1.

The standard application of structured overlay networks in peer-to-peer information retrieval is the implementation of a distributed inverted file structure for efficient keyword based search. In this scenario, several situations occur, in which the overlay network has to be constructed from scratch:

- A set of documents that is distributed among (a potentially very large number) of peers is identified as holding information pertaining to a common topic. To support efficient retrieval for this specific document collection, a dedicated overlay network implementing inverted file access may have to be set up.
- A new indexing method, for example, a new text extraction function for identifying semantically relevant keywords or phrases, is being used to search a set of semantically related documents distributed among a large set of peers. Since the index keys change as a result of changing the indexing method a new overlay network needs to be constructed to support efficient access.
- Due to updates to a distributed document collection an existing distributed inverted file has become obsolete. This may either result from not maintaining the inverted file during document updates or due to changing characteristics of the global vocabulary and thus changing the indexing strategy (e.g., term selection based on inverse document frequency). Thus a complete reconstruction of the overlay network is required.
- Due to catastrophic network failures the standard maintenance mechanisms no longer can reconstruct a consistent overlay network. Thus the overlay networks needs to be constructed from scratch. Of course, this scenario applies generally in any application, but becomes more probable when multiple overlay networks are deployed in parallel.

In principle a (re-)construction of an overlay network in any of these scenarios can be achieved by the standard maintenance model of sequential node joins and leaves. Most existing proposals for structured overlay networks [117, 154, 163] do not offer a completely parallel construction process involving all peers simultaneously. They assume a model of joins of peers in an essentially sequential process. However, this approach encounters two serious problems:

- The peer community will have to decide on a serialization of the process, e.g., electing a peer to initiate the process. Thus the peer community has to solve a leader election problem, which might turn out to be unsolvable for very large peer populations without making strong assumptions on coordination or limiting peer autonomy.
- Since the process is performed essentially in a serialized manner, it incurs a substantial latency. In particular it does not take any advantage of potential parallelization, which would be a natural approach.

In principle some systems like Pastry [154] would support concurrent construction as they take an optimistic approach in which concurrent node joins are possible as long as there are no conflicts. Similarly, in ring based topologies (like Chord), the most critical part for the overlay's functional correctness is to maintain the ring itself. Thus if there are simultaneous peer joins at disjoint portions of the ring (involving neighborhood changes of disjoint set of peers), such membership changes can be taken care of by a self-stabilization mechanism as described in the original proposition of Chord [163], or future refinements of the same [16].

However, this assumes that there already exists a large overlay, so that conflicts are rather unlikely. In an early stage of bootstrapping and with large number of peers joining concurrently, conflicts will however be very likely. Thus this type of strategy is not applicable to the problem we are addressing.

In this chapter we will address the problem how a structured overlay network can be constructed efficiently from scratch, a problem that the research community has only recently identified and started to address [6, 19, 92]. Our approach is a generic mechanism to autonomously partition a key-space in a completely parallel manner. The approach can potentially be used for constructing other structured overlays with fixed key space partitioning [17].

In data-oriented applications there exists an additional constraint to parallelized overlay construction - balancing load despite the skew in key distribution. We want to build an index structure which preserves the (lexicographic) locality of resources. Canonical methods of uniform hashing of keys to remove skew in the key distribution as used in DHTs are no more applicable. This has led to substantial research on including load balancing features into overlay networks [6, 63, 117]. The overlay construction approach needs to balance such skews in key-distribution over the key-space. During the overlay construction process we will address two types of load balancing problems simultaneously - the balancing of storage load among peers under skewed key distributions (i.e., number of keys per partition is balanced) and the balancing of the number of replica peers across key space partitions. The first problem is important to balance workload among peers and is solved by adapting the overlay network structure to the key distribution. The second one is important to guarantee approximately uniform availability of keys in unreliable networks where peers have potentially low availability. This is similar to a classical “balls into bins” scenario, where the key-space partitions are the bins and the peers (replicas) the balls. The extra twist, why existing solutions for balls into bins problems can’t however be directly used is that the number of bins (key-space partitions) itself is dynamic.

Our overlay construction approach is based on a key-space bisection process through a completely decentralized, parallel, and randomized algorithm for assigning peers to key space partitions in proportion to the key distributions of the partitions. By recursively applying key-space bisections, peers can incrementally construct the overlay network while maintaining load balance. We will introduce our approach in the context of the P-Grid overlay network structure, though the essential elements of the approach are applicable to overlay networks using fixed key space partitioning schemes, such as CAN [140] or Pastry [154]. We demonstrate the theoretical correctness of the basic key-space bisection process by analysis and simulation. The feasibility of building a complete system matching the theoretically predicted behavior has been validated with experimental results obtained from a full-fledged implementation deployed on the PlanetLab [37] infrastructure, and reported in Chapter 7.

4.3 Fast construction of load-balanced overlay

The process of constructing an overlay network from scratch should require low latency, i.e., be highly parallel and require minimal bandwidth consumption. At the same time ideally the following load balancing criteria should be achieved:

1. The partitioning of the search space should be such that each partition holds a maximal data load of d_{max}, e.g., measured as the number of keys present in the partition. We will call d_{max} also the maximal storage load in the following. [2] This allows the peers to describe the absolute (storage) load they are willing to contribute to the overlay. This is in contrast to existing approaches [32] where storage load is measured relative to other peers load. Using an absolute notion of load at peers means that the system as a whole adapts the average replication factor dynamically, based on the total storage capacity of the system and the total number of unique keys to be stored.
2. Each resulting partition should be associated with a constant number of peers n_{min}, such that the availability of the different data keys is approximately the same. We will call n_{min} also the minimal replication factor in the following.

With perfect load balancing these properties can be achieved if $d_{tot}n_{min} = d_{max}n$, where d_{tot} is the total number of data keys and n is the number of peers. Algorithm 4 shows a globally coordinated partitioning algorithm $Partition$ that attempts to achieve these load balancing goals by best effort while bisecting the key space. A typical overlay network that will ideally evolve for a skewed load-distribution had been shown in Figure 3.1.

The algorithm works as follows. Assume n peers are associated with one key space partition containing d data keys and two sub-partitions p_0 and p_1 containing d_0 respectively d_1 data keys, such that $d = d_0 + d_1$. To achieve load balance criterion 1, a fraction of $n\ \frac{d_i}{d}$ of peers should be associated with partition p_i for $i = 0, 1$. In case $n\ \frac{d_i}{d} < n_{min}$ at least n_{min} peers should be associated with p_i to achieve load balance criterion 2. $Partition$ recursively applies this bisection step to the key space.

Even with global knowledge and coordination, for various reasons, this algorithm can achieve the load balancing goals only approximately. Provided the number of data keys is large enough, i.e., $d_{tot} > d_{max}n/n_{min}$, the number of peers associated with a partition will be between n_{min} and $2n_{min} - 1$, instead of constant n_{min}. For very skewed data distributions it may happen that very small partitions contain a large fraction of the data keys, and bisection "disperses" many peers to underloaded partitions even before reaching such partitions. These are fundamental problems of any bisection approach. However, for practical

[2] In fact its roughly the average load each peer is willing to have. The actual maximum load at peers can be $2d_{max}$.

Algorithm 4 Partition(p, n, d)

```
 1: if $d \geq 2d_{max}$ and $n \geq 2n_{min}$ then
 2:   if $n \frac{min(d_0,d_1)}{d} \geq n_{min}$ then
 3:     $n_0 = n \frac{d_0}{d} n$; $n_1 = n \frac{d_1}{d}$
 4:     Partition($p_0$, $n_0$, $d_0$); Partition($p_1$, $n_1$, $d_1$)
 5:   else
 6:     if $d_0 < d_1$ then
 7:       $n_0 = n_{min}$; $n_1 = n - n_0$
 8:       Partition($p_0$, $n_0$, $d_0$); Partition($p_1$, $n_1$, $d_1$)
 9:     else
10:       {analogous}
11:     end if
12:   end if
13: end if
```

data distributions and large peer populations these problems are more theoretical in nature and $Partition$ achieves good load balancing properties provided n_{min} and d_{max} are chosen properly.

We will use in the following $Partition$ as an algorithm that defines what we consider as an optimal partitioning of the search space among peers and a resulting optimal overlay network - that is, our baseline case. Since in a peer-to-peer system no global coordination exists, the problem we intend to solve is to achieve the partitioning generated by $Partition$ by a decentralized process approximately. We will measure the quality of a solution by determining the deviation from the baseline case.

In a decentralized process peers do not have precise information on the number of peers and keys present in a partition and cannot know which decision the other peers in a partition take with respect to associating themselves with a sub-partition. The only available information is on the set of locally stored data keys and information gathered from local interactions with other peers.

The decentralized process we design is based on random peer encounters and a set of basic local interactions. The random encounters can be initiated by performing random walks on a pre-existing unstructured overlay network. The possible interactions peers can perform in their encounters can be classified in three categories - repartition, replicate or refer. These are shown in Figure 4.1.

If peers belong to the same partition they can either $repartition$ the present partition (a divide-and-conquer strategy) or $replicate$ the data keys they currently hold. If they do not belong to the same partition, they can $refer$ each other to other peers using their routing table entries and thus route to a peer that belongs to the same partition.

If peers from the same partition meet, they may decide to $repartition$ in case the current partition contains a sufficient number of data keys to justify a further split, i.e., the partition is overloaded (corresponding to line 1 in $Partition$). They can coordinate locally their decision. In addition, peers keep a reference to the peer encountered after a split, and thus incrementally construct their routing tables.

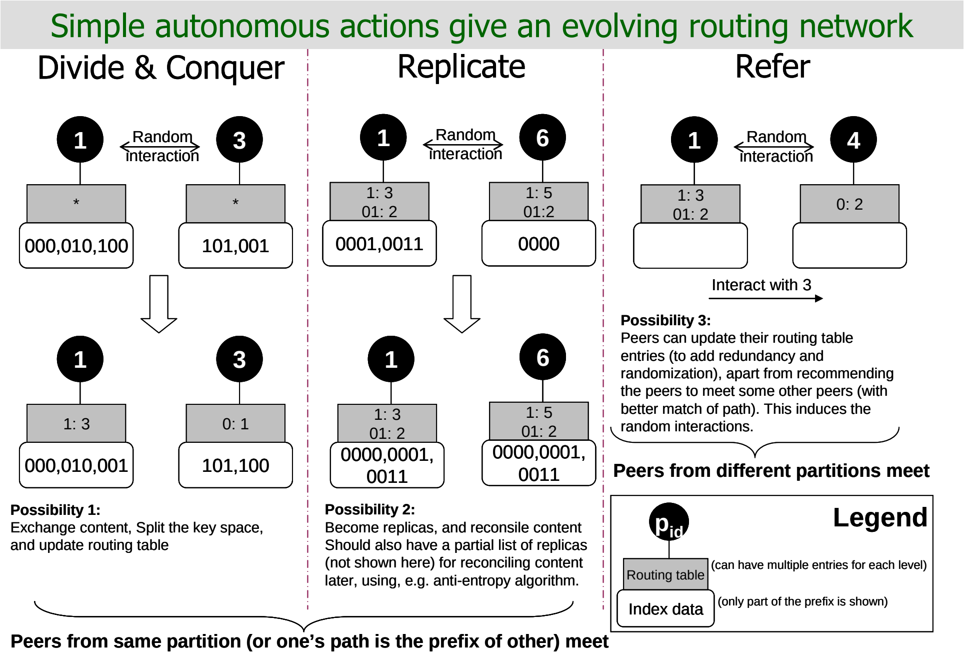

Simple autonomous actions give an evolving routing network
Divide & Conquer
Replicate
Refer
1
Random Interaction
3
*
*
000,010,100
101,001
1
3
1: 3
0: 1
000,010,001
101,100
Possibility 1:
Exchange content, Split the key space, and update routing table
1
Random Interaction
6
1: 3 01: 2
1: 5 01:2
0001,0011
0000
1
6
1: 3 01: 2
1: 5 01: 2
0000,0001, 0011
0000,0001, 0011
Possibility 2:
Become replicas, and reconsile content Should also have a partial list of replicas (not shown here) for reconciling content later, using, e.g. anti-entropy algorithm.
1
Random Interaction
4
1: 3 01: 2
0: 2
Interact with 3
Possibility 3:
Peers can update their routing table entries (to add redundancy and randomization), apart from recommending the peers to meet some other peers (with better match of path). This induces the random interactions.
Peers from different partitions meet
Legend
p_id
Routing table
(can have multiple entries for each level)
Index data
(only part of the prefix is shown)
Peers from same partition (or one's path is the prefix of other) meet

Fig. 4.1. Network evolution based on pairwise peer interactions

We can thus reduce the problem of load-balanced overlay network construction to the problem of decentralized partitioning of one key space partition. The problem is that a large number of peers have to perform the decision to split independently for allowing a fast construction of the overlay network, while making these independent decisions in a way that the ratio of the number of peers matches the ratio of the data load in the two partitions. In other words, the global behavior of the distributed decision making process should match the outcome of the partitioning step in the global partitioning algorithm $Partition$ (corresponding to lines 3 and 7 in $Partition$). The solution to this problem is one of the central contributions of the chapter and will be discussed in detail in next section.

4.3.1 Decentralized Partitioning

Consider a set P of $n+1$ peers which hold data keys from key space K. The space K is partitioned into two parts, 0 and 1, such that the load measured in number of data keys related to the partitions, l_0 and l_1 are p and $1-p$. In the following we assume w.l.o.g. that $0 \leq p \leq \frac{1}{2}$. Then the partitioning that we would ideally like to achieve should have the following properties:

1. *Proportional replication:* Each peer has to decide for one of the two partitions such that (in expectation) a fraction p of the peers decides for 0 and a fraction $1-p$ for 1. Thus the workload becomes uniformly distributed among the peers, meeting the load-balancing criteria in the resulting overlay.

2. *Referential integrity:* During the process each peer has to encounter at least one peer that decided for the other partition. Thus the peers have the necessary information to construct a routing structure, i.e., the overlay infrastructure, for delegating requests for keys they are no longer associated with.

A peer can initiate interactions with any peer selected uniformly randomly from P. We measure the cost of an algorithm solving the problem in terms of the number of interactions initiated by peers and this cost should be minimized. The quality of an algorithm solving the problem is measured by the deviation of the resulting distribution of peers from an optimal distribution that can be achieved based on global knowledge and coordination. First we assume that the value of p is known to all peers. We will analyze the influence of having only approximate knowledge of p by sampling the locally stored data keys later.

To clarify the critical issues we first discuss two simple heuristic approaches: In the case of $p = \frac{1}{2}$, a simple strategy to adopt would be that peers which have not yet decided for a partition, initiate a random interaction. If the contacted peer is also undecided, the peers decide for different partitions (*balanced split*), otherwise the peer initiating the interaction decides opposite to the contacted peer which has decided already (*unbalanced split*). In this way it learns about a peer from the other partition. Since the algorithm is symmetric, in expectation the same number of peers will decide for each partition, and it provides the best possible performance within the model, since in each interaction every possible decision is taken. We call this strategy *eager partitioning*. While the eager partitioning strategy works well for $p = \frac{1}{2}$, it cannot be employed for other values of p.

For an arbitrary but known p, a possible strategy, which we call *autonomous partitioning* **(AUT)**, would be that each peer makes a decision for one of the two partitions in advance, even without meeting any other peer and then tries to meet some peer from the other partition in order to satisfy the referential integrity constraint. In this setting, obviously some of the peer interactions are "wasted," whenever peers which have decided for the same partition meet. For the specific case of $p = \frac{1}{2}$, by modeling the interactions as Markovian processes, we observed that $2\log(2) = 1.386$ interactions are initiated on an average per peer asymptotically (i.e., for large n), as compared to $\log(2) = 0.693$ interactions per peer with eager partitioning. Thus autonomous partitioning is not an optimal strategy.

4.3.2 Adaptive eager partitioning

In the following we introduce a method for such an optimized solution to the partitioning problem, that has the characteristics of eager partitioning but works for all p.

Adaptive eager partitioning ***(AEP)*** *algorithm:.*

1. Each undecided peer initiates interactions with a uniformly randomly selected peer until a decision is reached. Selecting peers uniformly at random is a non-trivial problem in itself which we solve by a variant of random walks.

2. If the contacted peer is undecided the peers perform a balanced split with probability $0 \leq \alpha(p) \leq 1$ and maintain references to each other.
3. If the contacted peer has already decided for 0 then the contacting peer decides for 1 and maintains a reference to the contacted peer.
4. If the contacted peer has already decided for 1 then the contacting peer decides for 0 with probability $0 \leq \beta(p) \leq 1$ and with probability $1 - \beta(p)$ for 1. In the first case it maintains a reference to the contacted peer. In the second case it obtains a reference to a peer from the other partition from the contacted peer.

It is straightforward to see that condition (2) of the partitioning problem is satisfied. The question is now to determine how to satisfy condition (1) by properly choosing the probabilities $\alpha(p)$ and $\beta(p)$.

We model the peer interactions as a Markov process using mean value analysis. We assume that in each step i a peer which has not yet found its counterpart contacts another randomly selected peer. By p_i^0 and p_i^1 we denote the number of peers that have decided in step i for 0 and 1, respectively. Initially, $p_0^0 = p_0^1 = 0$. At the end of the process in some step t we have $p_i^0 + p_i^1 = n + 1$. We first assume that $\alpha(p) = 1$. Informally speaking, with this $\alpha(p)$ the partitioning proceeds as fast as possible, optimizing the required number of interactions. Then the model can be given as

$$p_i^0 = p_{i-1}^0 + \frac{1}{n}(n - p_{i-1}^0 - (1-\beta)p_{i-1}^1) \tag{4.1}$$

$$p_i^1 = p_{i-1}^1 + \frac{1}{n}(n - \beta p_{i-1}^1) \tag{4.2}$$

To determine the proper value of β for a given value of p, we have to solve this recursive system. The first important observation is that the recursion terminates as soon as no more undecided peers exist, i.e., as soon as $p_i^0 + p_i^1 = n + 1$. Thus we have first to find a value t_β such that $p_{t_\beta}^0 + p_{t_\beta}^1 = n + 1$. In general this will not be an integer value, but in the context of mean value analysis we allow fractional steps. By standard solution methods we obtain

$$\begin{aligned} p_i^0 &= \frac{n}{\beta}(2\beta - 1 + (1 - \frac{\beta}{n})^i - 2\beta(\frac{n-1}{n})^i) \\ p_i^1 &= \frac{n}{\beta}(1 - (1 - \frac{\beta}{n})^i) \end{aligned}$$

and evaluating the termination condition, we obtain

$$t_\beta(n) = \frac{\log(2)}{\log(\frac{n}{n-1})} \tag{4.3}$$

Note, that t_β does not depend on p, and thus the partitioning process requires the same number of interactions among peers independent of the load distribution. By definition $p = \frac{p^0_{t_\beta}}{n+1}$, thus we obtain a relationship between the network size $n+1$ and the load distribution p with $\beta(p,n)$, the decision probability to be used.

Having $\beta(p,n)$ dependent on n is problematic for two reasons: First the resulting equation is hard to solve, and second, more importantly, n is not necessarily known to the peers. Since we are interested in situations where n is (relatively) large we thus perform an asymptotic analysis. By letting $n \to \infty$ we obtain the following relationship among p and $\beta(p)$

$$p = 1 - \frac{1}{\beta}(1 - 2^{-\beta}) \tag{4.4}$$

Positive solutions for $\beta(p)$ cannot be obtained for all values of p. From Equation 4.4 we derive that positive solutions exist for $p \geq 1 - \log(2)$. This means that the algorithm cannot partition correctly for too highly skewed partitions. Therefore for $0 \leq p < 1 - \log(2)$ we have to pursue a different strategy, by reducing the probability of balanced splits, i.e., $\alpha(p) < 1$.

Through an analogous analysis, by setting $\beta(p) = 0$, we can derive relationships for $\alpha(p)$:

$$t_\alpha(\alpha, n) = \frac{\log(2\alpha)}{\log(n) - \log(1 - 2\,\alpha + n)} + 1 \tag{4.5}$$

and for the relation between $\alpha(p)$ and p when $n \to \infty$

$$p = -\frac{\alpha\ (1 - 2\,\alpha + \log(2\alpha))}{(1 - 2\,\alpha)^2} \tag{4.6}$$

Before we continue with the discussion of different partitioning algorithms, a statement on the modeling approach is necessary: We use a sequential approach to model and analyze what is a concurrent process. This is a simplification as well as an appropriate approximation for our purpose. Assume that the latency in one interaction is such that c other interactions among peers occur concurrently. Then the concurrent behavior of N peers corresponds (approximately) to the sequential behavior of $\frac{N}{c} = n + 1$ groups. The analysis we perform shows that the models we use are sufficiently accurate for relatively small n. Thus for large numbers of peers the model is a sufficiently good approximation, whereas for small N concurrency is less likely to occur and less critical.

The analysis provided in this section assumes that the system resides deterministically in its mean state. We call such an approximation as mean value analysis (MVA). A more accurate analysis of the distributed partitioning process looking into the time evolution of the probability mass/density function (EoDF) of the system is provided in Appendix A.1.

An interesting observation we make based on the EoDF analysis is that using AEP we have a lower variance than a binomial distribution as would have been achieved using AUT. Thus to say, AEP not only reduces the communication cost with respect to AUT, it also reduces the statistical noise associated with the partitioning process.

So far we also made an idealizing assumption that all participating peers know exactly the value of p during the decentralized partitioning process. In practice assumption of such global knowledge is however unrealistic. Instead peers need to use local estimate based on a limited number of interactions/samples. We study the effect of such approximate information in Appendix A.2

4.4 Algorithmic issues and heuristics

In order to use AEP for implementing the $Partition$ algorithm in a decentralized fashion we have to address several issues related to the global organization of the indexing process.

4.4.1 Initiating the indexing process

In absence of global coordination the mechanism to reach a decision to initiate the indexing process is not obvious. While it is not the focus of this chapter, and the initiation process is orthogonal to the index evolution process, we nonetheless describe a simple, decentralized strategy.

Depending on locally observed queries, individual peers may make autonomous decisions on whether a new index may be necessary or re-indexing may be required. Any of the peers that locally decide that indexing is useful can initiate a vote, by flooding the peer network. This flooding can use the pre-existing, generic, unstructured overlay network which we assume to exist.

When peers receive a voting request they can reply back their local decision. Additionally, helpful information, such as locally available storage space that the peer is willing to contribute to store information for the new index and the number of local data items to be indexed can be piggy-backed. Votes are sent back along the paths they arrived, and multiple votes are aggregated while flowing back to reduce bandwidth consumption. Based on the number of positive and negative responses, the peer which initiated the voting can then decide whether to initiate index construction or not, and can flood the decision back to all peers. Additionally, based on the aggregate storage space available, and the amount of storage required for all the data items (references) in the system, the decision will contain the parameters for ensuring optimized utilization of the available resources (i.e., d_{max} and n_{min}) and for synchronization of the indexing process. For simplicity we assume a collaborative environment where the majority of peers does not behave maliciously or in a Byzantine manner, and adheres to the democratic decision of the group, and thus participates in the indexing irrespective of their individual votes. In an alternative scenario, a part of the peer population may opt out of the indexing process, in which case we need to ignore their presence in the whole process, and only consider the peers which agree to participate in the indexing process. Some of such peers who originally decide not to participate in the indexing may later want to join the already (partially) evolved overlay, and will be treated as new peers joining the system. These peers will use the sequential joining algorithm described in Section 4.5 to join the overlay.

4.4.2 Synchronizing and terminating the indexing process

The partitioning algorithm introduced in Section 4.3 enables reaching a decision in parallel on bisecting the key space proportionally among a group of autonomous peers. In the indexing process the algorithm is executed multiple times and a synchronization mechanism is needed. In addition peers need to autonomously recognize when to terminate the indexing process. We realize this as follows.

The peer communicating the decision to start the indexing process provides the parameters d_{max} and n_{min} as used in $Partition$. The values are chosen such that $d_{max} = d_{avg}n_{min}/2$, where d_{avg} is the average number of data keys peers hold (as mentioned in Section 4.4.1 this information can be derived from information piggy-backed to the votes). Additionally, it provides a time t_{init}. Before starting to partition, peers replicate their data keys at time t_{init} to n_{min} randomly chosen other peers. Thus at the start of the indexing process all data keys are already replicated the desired number of times in the network.

Besides estimating the number of data keys in the current partition, peers also have to estimate the number of current peers, in order to perform the proper decisions in algorithm $Partition$. Attempting this directly, by learning about all existing replicas at each level of the partitioning process, would unnecessarily slow down the progress of indexing. Instead, we estimate the number of replicas in a partition by analyzing the overlap in the sets of data keys of two peers interacting in a balanced split. If D_i denotes the set of data keys peers $p_i, i = 1, 2$ hold, and $D = D_1 \cup D_2$, then $\frac{|D_1||D_2|n_{min}}{|D|d_{max}}$ is a maximum likelihood estimate of the expected number of peers in the current partition. For example, if $D_1 = D_2$ and $|D_1| = d_{max}$ then it should be expected to have n_{min} peers in the partition since initially data keys have been replicated n_{min} times. To ensure the correctness of this estimation was the purpose of initially replicating the data.

During partitioning, peers that have extended their paths attempt to immediately contact other peers to perform the partitioning at the next level. If they do not succeed in identifying a different peer in the same partition with which a useful interaction can take place, i.e., "divide and conquer" or "replicate", after a fixed number of attempts (e.g., 2), using the *refer* interaction (see Figure 4.1), they stop to initiate interactions and only will continue after being contacted by another peer. In this way peers that are "ahead of the crowd", e.g., due to faster network connections, are forced to wait for the slower ones. The same mechanism also eventually leads to termination of the process, when peers encounter only fully synchronized copies of themselves.

Initiating the indexing process, as well as synchronizing and terminating it are required to bootstrap the index. After the load-balanced distributed index is constructed, it also needs to be maintained. Maintenance operations needs to account for several sources of dynamics in the system - particularly membership and workload dynamics.

Membership dynamics caused by new peers joining an existing network is dealt with as will be described in next Section 4.5. We study how to deal with existing members leaving and re-joining the network with potentially different physical address in Chapter 6.

Changing workload, caused by addition of new keys may lead to further partitioning of the affected overloaded key-space partition(s). Deletion of existing keys being indexed is dealt with by coalescing underloaded partitions as is described in the following. Finally, several of these maintenance mechanisms as well as the overlay construction mechanism in itself leads to variation of replication factor across different partitions, and a background process to re-balance replication across partitions will be described in Section 4.6

4.4.3 Coalescing partitions (path retraction)

If load in any particular portion of the key-space decreases, such that the number of keys in some partitions is lower than the minimal load peers are willing to accept, the corresponding peers will then retract their path, essentially leading to coalescing partitions. If replication of a partition falls below the n_{min} threshold (because of departing peers) and the background process of replication balancing does not repopulate the partition quickly enough, peers from two complementary partitions need to retract paths in order to provide the minimal desired fault tolerance.

Path retraction is essential in order to provide the P-Grid network adaptivity to changing load, and complements the partitioning based peer path extension.

Such decrease of load may result from deletion of keys, or the global load distribution changes, so that the average load per peer needs to deal with (determined by parameter d_{max}) is increased.

Note that in a setting where the load is different from storage, say computational load, the computational capacity at peers instead of storage space will determine the parameter d_{max} and end of a computational task will be equivalent to deletion of the corresponding key.

4.4.4 Complexity

The goal of our approach to index construction is to perform it with low bandwidth consumption and low latency. With regard to bandwidth consumption a necessary requirement is to perform comparably to a sequential approach using standard construction mechanisms, i.e., $O(n\ log^2 n)$. To study this, we look at the complexity in the case of a balanced key distribution ($p = \frac{1}{2}$ for all granularity of key-space partitions). Then for partitioning at one level, peers engage in $log(2)$ bilateral interactions on average. In addition, to locate a random peer in the same partition at level k, peers have to route on expectation $log(k)/2$ steps when performing the *refer* interaction. This shows that the total number of interactions is also of order $O(n\ log^2 n)$.

However, the overlay construction latency is $O(log^2 n)$ because of parallelization of peer interactions as opposed to $O(n)$ in the standard maintenance model.

4.5 Peers joining a (partially) existing P-Grid network

In a decentralized system design, synchronization of peer activities can neither be guaranteed, nor should it be required. One practical implication of this is that some peers may complete a (or several) phase(s) of proportional partitioning, while some other peers lag behind, still running the proportional partitioning for a higher level of the tree (a larger key-space). Moreover, new peers will join the overlay which might have already evolved either completely or partially. Such new peers may comprise either existing peers which decided to participate in the indexing (overlay) later than other peers, or totally new peers.

Structurally it is the same problem from the point of view of any individual peer, irrespective of its current degree of "specialization" in terms of the key-space partition it is responsible for. If its own path is strictly a prefix of any other peer's path, then it knows that it has lagged behind in the process of partitioning (not if difference of path lengths is 1). In such a scenario, this peer will need to catch up with the partitioning process. Thus to say, it has to specialize its path so that it also is responsible for a leaf node of the current snap-shot of the evolved/evolving P-Grid network.

In this setting it is desirable that each key-space partition is replicated by the same number of peers. Assuming that replication across all partitions is originally balanced, a first order balancing will be to choose any of the partitions with equal probability. The challenge in doing so comes from the fact that typically the size of the partitions are not same (unbalanced tree), which requires a non-trivial mechanism to choose any particular key-space uniformly randomly. If this can be achieved, we'll have the normal balls into bins scenario - the key-space partitions being the bins, and the newly joining peers being the balls.

At a first glance, it may however look like a more complicated problem if the key-space partitions are not equally replicated. Then the question we need to answer is - "How do the joining peers choose a key-space partition so that the imbalance of replication across different key-spaces is reduced?" Intuitively, choosing uniformly randomly any of the partitions no more seems to be a realistic means, since more peers should replicate the underpopulated partitions.

On the other hand, knowing globally the replication at each partition, and determining the weighted probability with which to choose peers so that each corresponding partition is chosen uniformly looks both computationally non-trivial, even if global knowledge and coordination could have been used, and also impractical in a decentralized setting.

To apparently make things worse, randomly contacting existing members of the network will mean favoring the overpopulated partitions.

To solve this problem we propose a heuristic, which separates the two concerns of choosing partitions uniformly and then replicating these partitions uniformly.

Once we choose the partitions uniformly, simulation based study shows that the same mechanism of using multiple choices to achieve a second degree of more refined load-balancing in the original (un-weighted) balls into bins mechanism can also be used in such a situation. Its worth pointing out that such multiple choices are anyway necessary in order to reduce the otherwise high variance observed by uniformly assigning balls into bins. Thus, multiple key-space partitions are chosen uniformly by the joining peer, and then the partition with the least replication is replicated.

Two practical problems need to be dealt with in a peer-to-peer environment in order to realize such a separation of concern based sequential peer joins in an existing network.

First of all, how do we choose a key-space partition uniformly randomly out of all the partitions?

If any random peer is chosen, the probability of choosing a more replicated partition will be higher than that of choosing a less replicated partition. Thus randomly choosing a peer itself will provide a biased mechanism to choose a key-space partition. However, that is the basic primitive we need to start with. The separation of concern between the uniform choice of a key-space partition irrespective of how replicated it is and hence how a random peer will be chosen, is achieved by the Algorithm 5 described below.

Uniformly choosing the key-space partition using $FindPartitionUniformly$ algorithm 5 thus achieves a first-order load-balance. In fact it becomes exactly a balls-into-bins problem, with the key-space partitions being the bins, and the sequentially joining peers being the balls. As with sequential balls-into-bins approaches the statistical noise can be reduced using multiple choices. The joining peer should choose to replicate the least replicated partition.

Secondly, choosing a key-space partition means communicating with a peer replicating that partition. This peer may not know all other replicas of the partition - because of the dynamics of the system, and hence, it will provide an imperfect information about the replication factor in the key space partition. Thus, unlike in the classical use of multiple choices based on perfect knowledge, we need to make the decisions based on imperfect information. Simulations (in Section 4.8.2) show that our heuristic is fairly robust to such imperfect knowledge of the load and achieves good load balancing.

In the following, we explain how key-space partitions can be uniformly randomly chosen.

4.5.1 Local view of the global structure

Lets consider the following idealized scenario, where a P-Grid network is completely formed - all peers are responsible for leaf-nodes of the tree. If there is no further restructuring of the P-Grid network, each peer can determine the number of key space partitions that exist for any given prefix of its own path, based solely on communication with peers in its own routing table.

As a slight abuse of the notation introduced in Section 3.2.1, we say Π_l^p is the key-space partition with prefix $\pi(p,l)$ and $|\Pi_l^p|$ is the number of leaf nodes in the P-Grid corresponding to that part of the key-space. Note that, each of these partitions will have multiple peers, but we are counting just the partitions themselves. Since once the overlay construction is finished, a peer is always responsible for a leaf node, hence peer p will have $|\Pi^p_{|\pi(p)|}| = 1$.

Then the information $|\Pi^p_l|$ can be obtained by each peer by collaborating with only peers it already knows from its routing table for all $l = 1, ..., |\pi(p)|$. This information is calculated and locally cached at peer p using information from a suitable peer in its routing table $r \in \rho(p, l+1)$ as follows:

$$|\Pi^p_l| \quad = \quad |\Pi^p_{l+1}| + |\Pi^r_{l+1}| \tag{4.7}$$

The overheads of such communication is minimal, and in fact can be piggy-backed with several other kinds of messages like route maintenance messages or query forwarding messages. When a network is stable - that is, the key-space is no more re-partitioned - this local view of the global network can be obtained at each peer for any level incurring a latency corresponding to the maximum depth of the P-Grid tree for that subspace, and message complexity of its own path length.

4.5.2 A new peer joining an existing P-Grid network

When a new peer q wants to join the existing P-Grid network, it first contacts any existing peer p of the network. Then, based on local information stored at the entry peer p, q is referred probabilistically to another peer (or p itself), so that finally q can make an uniformly random choice of the key-space partition to replicate. The references that p provides come from its own routing table, and the probabilistic decision that p makes in referring q is also based on information obtained in collaboration with its routing table entries as described above.

The process for newly joining peer q to choose a random key-space partition is given in Algorithm 5, where q makes a request: FindPartitionUniformly(p, 1).

Algorithm 5 New peer q joins P-Grid - interaction at peer p: FindPartitionUniformly(p, l)

```
if $\pi(p) == l$ then
    Return $p$'s partition; {$q$ has chosen a partition (that of $p$'s) uniformly randomly among all the key-space partitions.}
else
    Draw a random number $Rand$ uniformly from $[0, 1]$;
    if $Rand \leq |\Pi^p_l|/|\Pi^p_l| + |\Pi^r_l|$ where $r \in \rho(p, l)$ then
        $q$ executes FindPartitionUniformly(p,$l + 1$); {Note that this is a local step at $p$.}
    else
        $q$ executes FindPartitionUniformly(r,$l + 1$) where $r \in \rho(p, l)$; {This involves communication. Note also that the joining
        peer $q$ may populate (some of) its routing tables till level $l - 1$ from $p$'s routing table, and can have $p$ as a level $l$ reference.
        This process makes the need to replicate complete routing table from any one peer unnecessary, and de-correlates the
        routing entries among replica peers.}
    end if
end if
```

This process enables a peer to locate each P-Grid key-space partition with the same probability. The intuition behind this is as follows. Lets say there are $|\Pi_0|$ and $|\Pi_1|$ partitions with prefixes 0 and 1 respectively, where the total number of partitions $|\Pi| = |\Pi_0| + |\Pi_1|$. Then if q approaches a peer p with prefix 0, it will refer q to a peer with prefix 1 with probability $|\Pi_1|/(|\Pi_0| + |\Pi_1|)$ and q will decide for its first bit as 1, otherwise q will decide on the first bit of its path as 0 with probability $|\Pi_0|/(|\Pi_0| + |\Pi_1|)$ and the same process will be continued until a peer is reached when q has decided the number of bits corresponding to the path length of that peer. Since the decision by q at each step to extend its path with 0 or 1 is independent of the decision in the previous step, these probabilities are multiplied to determine the probability of q's choosing any specific path. This yields the probability of q's choosing one and all existing partitions of the P-Grid a value of $1/\Pi$, i.e., each of these partitions are uniformly chosen.

Choosing multiple partitions based on this scheme and replicating the least replicated partition (counting the actual number of peers in a partition) reduces the variation of replication factor across the various partitions. This is an idea borrowed from the use of multiple choices in a balls-into-bins problem.

Thus peer joins do not lead to any restructuring of the network, and more significantly, the existing peers do not need to change their routing tables at all for functional correctness! This is yet another unique feature of P-Grid network, and hence P-Grid can deal with very high rates of incoming peers, without affecting the functionality of the existing peers, nor needing any stabilization algorithm for peer joins. However, routing entries can be changed slowly in a background process to provide better connectivity, exploiting better the available redundancy, and for (routing and query answering) load-balancing purposes.

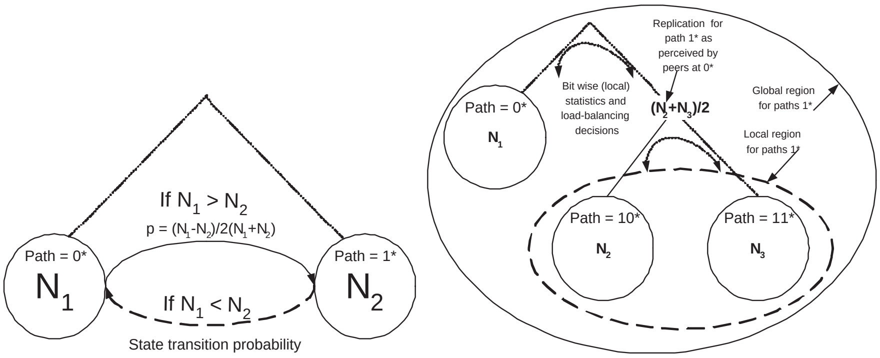
The new set of replicas can at some point again initiate partitioning their local portion of the key-space using the decentralized partitioning algorithm, without affecting peers from the rest of the network. When to repartition a key-space is similar in nature to when to start indexing in general. Heuristics include starting repartitioning if the number of replicas in a partition exceeds a threshold. Another reason to restart partitioning is increase of (storage) load at peers.

4.6 Re-balancing structural replication

The parallelized overlay construction mechanism ensured that all key-space partitions are on an average equally replicated, achieving a first-order balancing of replication factor. The variance from a single partitioning process is shown in Appendix A.1. Apart the effect of construction process, peers may go offline, which changes the replication factor for different partitions.

Here we investigate how to reduce the variation of replication factor for each key-space partition based on migration of peers from one partition to another. We use a reactive randomized distributed algorithm which tries to achieve globally uniform replication adaptive to globally available resources based on locally available (gathered) information. Before introducing the algorithm we introduce the intuition underlying its design.

Consider a P-Grid of leaves as shown in Figure 4.2(a). Let $N_1 > N_2$ be the actual number of replica peers with paths 0 and 1. To achieve perfect replication balancing $\frac{N_1 - N_2}{2}$ of the peers with path 0 would need to change their path to 1. Since each of the peers has to make an autonomous decision whether to change its path, we propose a randomized decision: Peers decide to change their paths with probability $p_{0\rightarrow 1} = max(\frac{N_1 - N_2}{2N_1}, 0)$ (no $0 \rightarrow 1$ transition occurs if $N_2 > N_1$).



If N₁ > N₂
p = (N₁-N₂)/2(N₁+N₂)
Path = 0*
N₁
Path = 1*
N₂
If N₁ < N₂
State transition probability
Replication for path 1* as perceived by peers at 0*
Bit wise (local) statistics and load-balancing decisions
(N₂+N₃)/2
Global region for paths 1*
Local region for paths 1*
Path = 0*
N₁
Path = 10*
N₂
Path = 11*
N₃

(a) P-Grid with two leaves (b) P-Grid with three leaves

Fig. 4.2. Re-balancing replication factor: Migration of peers from one key-space partition to another.

Now, if we set $p_0 = \frac{N_1}{N_1+N_2}$ as the probability that peers have path 0, and similarly $p_1 = \frac{N_2}{N_1+N_2}$, then the migration probability becomes $p_{0\rightarrow 1} = max(\frac{1}{2}(1 - \frac{p_1}{p_0}), 0)$. It is easy to see that with this transition probability on an average an equal replication factor is achieved for each of the two paths after each peer has taken the migration decision. In a practical setting peers do not know N_1 and N_2, but they can easily determinate an approximation of the ratio $\frac{N_1}{N_2}$ by keeping statistics of the peer they encounter in (random) interactions.

Now consider the case of a P-Grid with three leaves, as shown in Figure 4.2(b), with N_1, N_2 and N_3 replicas for the paths starting with 0, 10 and 11 respectively. This extension of the example captures the essential choices that have to be made by individual peers in a realistic P-Grid. In an unbalanced tree, knowing the count of peers for the two sides at any level is not sufficient because, even if replication is uniform, the count will provide biased information, with a higher value for the side of the tree with more leaves. On the other hand, knowledge of the whole tree (shape and replication) at all peers is not practical but fortunately not necessary either. For example, in the P-Grid with three leaves, peers with path 0 will meet peers with paths 10 and 11. Essentially, they need to know that there are on an average $\frac{N_2+N_3}{2}$ peers at each leaf of the other sub-tree, but do not need to understand the shape of the sub-tree or the distribution of replication factors.

Thus, while collecting the statistical information, any peer p counts the number of peers encountered with common prefix length l for all $0 \leq l \leq |\pi(p)|$. It normalizes the count by dividing it with $2^{|\pi(q)| - |\pi(p) \cap \pi(q)|}$.

Thus peers obtain from local information an approximation of the global distribution of peers *pertaining to their own path*. The latter aspect is important to maintain scalability.

In our example, peers with path 0 will count on an average $\frac{N_2+N_3}{N_1}$ as many occurrences of peers with path 10 or 11 than they will count with path 0, but will normalize their count by a factor of $\frac{1}{2}$. Thus at the top level they will observe replica balance exactly if on an average $N_1 = \frac{1}{2}(N_2 + N_3)$. If imbalance exists they will migrate with probability $p_{0\to 1} = max(\frac{1}{2}(1 - \frac{p_1}{p_0}), 0)$, where, now $p_0 = \frac{N_1}{N_1+\frac{1}{2}(N_2+N_3)}$ and $p_1 = \frac{\frac{1}{2}(N_2+N_3)}{N_1+\frac{1}{2}(N_2+N_3)}$.

Once balance is achieved at the top level, peers at the second level with paths 10 and 11 will achieve balance as described in the first example. Thus local balancing propagates down the tree hierarchy till global balance is achieved. The peers with longer paths may have multiple migration choices, such that balancing is performed at multiple levels simultaneously. For example, if $N_1 = N_2 < N_3$ peers with path 11 can choose migrations $11 \to 0$ and $11 \to 10$ with equal probability.

Note that N_i changes over time, and thus the statistics have to be refreshed and built from scratch regularly. Thus the algorithm has two phases, (1) gathering statistics and (2) making probabilistic decisions to migrate. Now we introduce the algorithms extending the intuition to the general situation.

4.6.1 Collecting statistical information at a peer

In a decentralized setting, a peer p has to rely on sampling to obtain an estimate of the global load imbalance: Upon meeting any random peer q, peer p will gather statistical information for all possible levels $l \leq |\pi(p)|$ of its path, and update the number of peers belonging to the same subspace $\Sigma_p(l) = |\{q \text{ s.t. } |\pi(p) \cap \pi(q)| \geq l\}|$ and the complimentary subspace $\overline{\Sigma}_p(l) = |\{q \text{ s.t. } \overline{\pi(p,l)} = \pi(q,l)\}|$ at any level l. When peers p and q interact statistics gathering is performed as follows:

```
$l := |\pi(p) \cap \pi(q)|$;
$\overline{\Sigma}_{p\|q}(l) := \overline{\Sigma}_{p\|q}(l) + 2^{1+l-|\pi(q\|p)|}$;
$\forall 0 \leq i < l \; \Sigma_{p\|q}(i) := \Sigma_{p\|q}(i) + 2^{1+i-|\pi(q\|p)|}$;
```

where the meta-notation $p\|q$ denotes that the operations are performed symmetrically both for p and q. We use a weighted aggregate, with longer path differences making smaller contributions, compensating for the fact that longer paths mean more partitions and hence more peers even if the replication of each of these partitions are same.

4.6.2 Choosing migration path for a peer

A path change of a peer only makes sense if it reduces the number of replicas in an underpopulated subspace (data). Therefore, as soon as a minimum number of samples have been obtained, the peer tries to identify possibilities for migration. It determines the largest l_{max} such that $\frac{\Sigma_p(l_{max})}{\overline{\Sigma}_p(l_{max})} > \zeta$ where $\zeta \geq 1$ is a dampening factor which avoids migration if load-imbalance is within a ζ factor. We set $l_{max} := \infty$ if no level satisfies the condition.

If all peers try to migrate to the least replicated subspace, we would induce an oscillatory behavior such that the subspaces with low replication would turn into highly replicated subspaces and vice versa. Consequently, instead of greedily balancing load, peers essentially have to make a probabilistic choice proportional to the relative imbalance between subspaces. Thus $l_{migration}$ is chosen between l_{max} and $|\pi(p)|$ with a probability distribution proportional to the replication load-imbalance $\frac{\Sigma_p(i)}{\bar{\Sigma}_p(i)}$, $|\pi(p)| \geq i \geq l_{max}$. Thus the migrations are prioritized to the least populated subspace from the peer's current view, yet ensuring that the effect of the migrations is fair, and not all take place to the same subspace. There are subtle differences in our approach to replication balancing in comparison to the classical balls into bins load balancing approach, because in our case there are no physical *bins*, which would share load among themselves, and it is rather the *balls* themselves, which need to make an autonomous decision to migrate. Moreover, the load sharing is not among bins chosen uniformly, but is prioritized based on locally gathered approximate global imbalance knowledge.

Migration is an expensive operation—it leads to increased network maintenance cost due to routing table repairs, apart from the data transfer for replicating a new key space—it should only occur if long-term changes in data and replication distribution are observed and not result from short term variations or inaccurate statistics. To further reduce oscillatory behavior, the probability of migration is reduced by a factor $\xi \leq 1$. The parameters ζ and ξ are design parameters, which we chose based on performance in terms of reduction of variance without unnecessary oscillations as observed based on simulations in Section 4.8.3.

4.6.3 Migrating a peer

The last aspect of replication load balancing is the action of changing the path. For that, peer p needs to find a peer from the complimentary subspace and thus inspects its routing table $\rho(p, l_{migration})$ s.t. $\pi(p) \cap \pi(q) = l_{migration}$. After identifying a peer q, p clones the contents of q, including data and routing table, i.e., $\delta(p) := \delta(q)$ and $\rho(p, *) = \rho(q, *)$, and the statistical information is reset in order to account for the changes in distribution. Peers which referred to p in their routing tables will subsequently discover the change of p's path, and update their routing table accordingly.

4.7 P-Grid as a DHT

The P-Grid construction mechanism described in this chapter can accommodate a wide range of distribution of keys over the key-space. Since not all applications need preservation of lexicographic ordering, and a simpler data-structure like DHT may well serve such applications, we'd like to reemphasize that a DHT is a special instance of the P-Grid overlay network. If an uniform hashing function is used to generate the keys for P-Grid to index, it'll realize a DHT.

For parallelized construction using recursive repartitioning, this implies that the partition parameter $p = 0.5$ should be chosen at all levels of partitioning to realize a DHT.

That apart, a DHT P-Grid network can be easily constructed sequentially as described next.

4.7.1 Balanced tree construction with controlled replication

A traditional DHT like Chord or Pastry has a predetermined globally fixed number (say f) of replicas for each key. Moreover, load-balancing for DHTs often aim to partition the key-space as uniformly as possible - which is equivalent to building a tree which is as balanced as possible.

Construction of such a traditional DHT is straightforward using the $FindPartitionUniformly$ algorithm 5. A newly joining peer uses the algorithm to choose two (multiple) key-space partitions randomly, and decides to replicate the partition with shortest path. Path extensions are done when the replication of the specific path reaches $2n_{min}$, and each of the resulting partitions are replicated by n_{min} peers. In the whole process, if we ignore node departures, apart the very beginning when the network population is less than n_{min}, all the partitions are always replicated by at least n_{min} peers, providing a minimal fault-tolerance.

Finally, if peers depart and the replication of a particular key-space partition falls below n_{min}, the corresponding peers can as well coalesce the partition with the complementary partition.

Following these simple heuristics a DHT with fairly good balance of the key-space partition sizes and replication factor between n_{min} and $3n_{min}$ can be realized, and the variance further reduced based on a background process.

We described above a simple P-Grid construction mechanism to emphasize that the P-Grid network itself is no more complex than other DHTs and is as simple to build. The complex load-balancing techniques we discussed earlier in the chapter however are to realize other properties, like accommodating arbitrary load-skews and construction of the index (overlay) in a rapid manner.

4.8 Evaluation results

4.8.1 Parallelized load-balanced overlay construction

In order to study the global behavior of the indexing algorithms when using the decentralized proportional partitioning (AEP) algorithm recursively in the sub-partitions based on local estimates of the load-skew, we performed simulation studies implemented in Mathematica. We were mainly interested whether the desired load balancing properties would be achieved under the various approximations and whether the algorithm performs as predicted.

In the simulations we used peer populations of sizes 256, 512, and 1024. As data distributions we used a uniform distribution, a Pareto distribution with PDF $\frac{a\,k^a}{x^{1+a}}$ with parameters $k = 1$ and $a = 0.5, 1.0, 1.5$, and a Normal distribution with mean value $\frac{1}{2}$ and standard deviation 0.0513, and test data from text retrieval experiments (project Alvis [30]). In Figure 4.3 these distributions are denoted as *U*, *P0.5*, *P1*, *P1.5*, *N* and *A*. The Pareto and Normal distributions represent cases with extremely skewed distributions. Initially, we randomly assigned 10 keys from the distributions to peers, so that they held samples. We tested with $n_{min} = 5$ and $n_{min} = 10$ such that at least 5 (respectively 10) replicas of the keys are generated. Typically the experiments had $d_{max} = 10n_{min}$. All experiments were repeated 10 times and the results were averaged. The algorithms were implemented as described above. The experiments were executed on a workstation cluster using up to 36 machines and were running for more than a week. Note that there were 36 separate experiments, each conducted 10 times. Furthermore, in a real network the peers would use exclusive resources, and thus the actual overlay construction process is much faster.

For evaluating the experiments we primarily were determining the degree to which the load balancing of peers across key space partitions worked. To do so, we compared the generated key sets to the distribution, that would be generated by global coordination ($Partition$ algorithm).

The $Partition$ algorithm generates a distribution $(k_i, n_i), i = 1, \ldots, K$, where k_i are the K partitions of the key space generated and n_i are the number of peers associated with each partition. We compared this distribution to the distribution (k_i^d, n_i^d) generated by the decentralized algorithm.

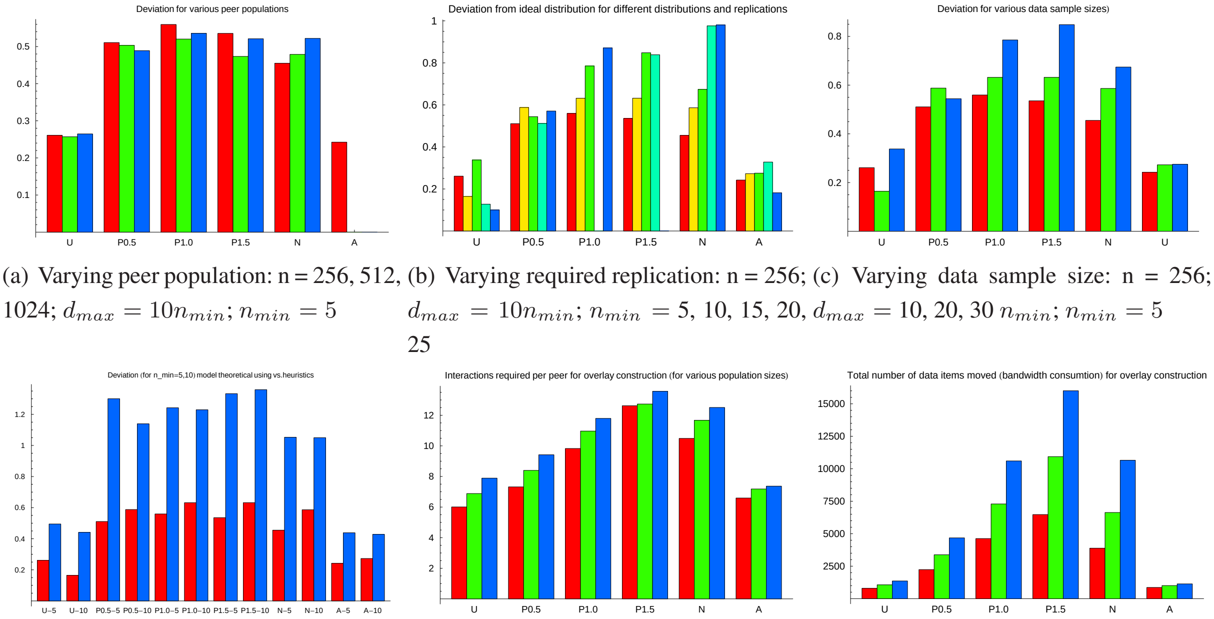
$$\frac{\sqrt{\sum_i^K (n_i - n_i^d)^2}}{\frac{1}{K}\sum_i^K n_i^d}$$

As explained in Section 4.3, we consider the distribution generated by $Partition$ as the optimal distribution. Measuring the distance to this distribution provides a measure for the quality of load balancing.

The first experiment (Fig 4.3(a)) for $n_{min} = 5$ and $d_{max} = 10$ shows the quality of load balancing depending on the peer population size for the different distributions. One can observe that the quality remains practically stable independent of the size.

We also investigated the influence of the replication factor n_{min} by comparing $n_{min} = 5, 10, 15, 20, 25$ (Fig 4.3(b)). In principle the load balancing properties should not be affected as we measure deviations relative to the average replication. This is confirmed for less skewed distributions, whereas for the strongly skewed distributions a certain degradation can be observed. We have still to investigate in detail the reasons for this effect, but most likely it is related to the relatively low number of partitions with high replication factors.

We were also interested in the influence of the sample size d_{max} on the quality of load balancing. It might be expected that more samples lead to higher accuracy. In fact, the result (Fig 4.3(c)) shows that no



(a) Varying peer population: n = 256, 512, 1024; $d_{max} = 10n_{min}$; $n_{min} = 5$

(b) Varying required replication: n = 256; $d_{max} = 10n_{min}$; $n_{min} = 5$, 10, 15, 20, 25

(c) Varying data sample size: n = 256; $d_{max} = 10$, 20, 30 n_{min}; $n_{min} = 5$

(d) Theory vs. Heuristics

(e) Interactions per peer: n = 256, 512, 1024; $d_{max} = 10n_{min}$; $n_{min} = 5$

(f) Bandwidth consumed (data keys moved): n = 256, 512, 1024; $d_{max} = 10n_{min}$; $n_{min} = 5$

Fig. 4.3. Simulation results for various experiment scenarios.

such influence exists. This is insofar important as it shows that the partitioning can be done using very small samples which enables several possibilities for optimization to reduce bandwidth consumption.

In order to understand the quality of the load distributions achieved we also analyzed the role of our theoretical framework (Fig 4.3(d)). We replaced the functions $\alpha_{corr}(p)$ and $\beta_{corr}(p)$ by heuristic functions which likely would be chosen in the absence of a theoretical understanding of their properties. The functions $\alpha_{corr}(p)$ and $\beta_{corr}(p)$ are derived by extending the analysis of Section 4.3.2 to account for the fact that peers do not have exact knowledge but only local estimates of the load-skew (parameter p) during the partitioning process. Such an error analysis assuming that the peers have a randomly uniform sample of load-skew is provided in Appendix A.2. The hypothesis we wanted to verify was whether the concrete nature of these functions plays a significant role in view of the many approximations made in the overall distributed algorithm. We chose

$$\alpha_{heur}(p) = \frac{1}{\frac{1}{p} - 1}, \beta_{heur}(p) = 0$$

These functions exhibit qualitatively the same behavior as the ones used by AEP. The experiment was executed for $n = 256$ and $n_{min} = 5$. The conclusion is clear from the result: Even a minor change to the theoretically correct functions degrades the quality of load balancing substantially. Thus the theoretical basis proves valuable despite many idealizing assumptions.

We also analyzed the communication costs of the algorithm. We can see that both the number of interactions per peer (Fig 4.3(e)), and the overall bandwidth consumption per peer measured in terms of the total number of data keys exchanged among all peers during the interactions (Fig 4.3(f)) grow gracefully in terms of the network size, as expected from theory. However, skew in the data distribution can significantly increase the bandwidth consumption.

4.8.2 New peers joining an existing network

We study the process of new peers joining an existing P-Grid network. We conducted simulation experiments with a randomly generated P-Grid network which had 120 partitions of the key-space according to the density shown in Figure 4.4. Each vertical line in the figure corresponds to a partition of the $[0, 1]$ interval of the key-space in the instantiated P-Grid corresponding to the decimal representation of the partition's path (which is a binary string). The unequal spacing between these lines in the plot is because the instantiated P-Grid is not balanced,and different partitions have different path lengths.

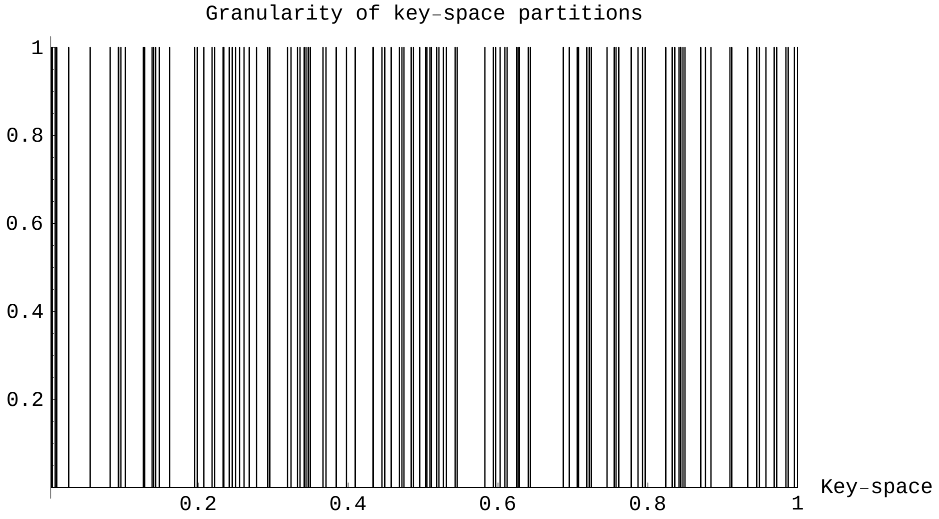


Fig. 4.4. Key-space partitioning granularity of a randomly generated unbalanced P-Grid network with 120 partitions. Each vertical line in the figure corresponds to a partition of the key-space in the instantiated P-Grid.

The average replication of each of these partitions was 20. However the experiments were conducted with different initial variance of replication factor across the key-space partitions. In one setting, the actual replication factor was chosen to be precisely 20 across all partitions, while other settings included the replication factor chosen uniformly randomly (integer) between $[20 - i, 20 + i]$ for various values of i.

As previously mentioned, in a real setting, individual peers may not know the correct replication factor corresponding to its own key-space partition. So we conducted experiments for various degree of error in local estimates of replication factor at peers. In these experiments, each peer made an estimation error (a maximum) of $j\%$. Thus to say, the local estimate of replication factor varied between $100 - j\%$ to 100% of the actual replication factor in the partition, chosen uniformly randomly. The model for replication factor estimation error itself is not necessarily representative, but the main intention of these experiments was to see how errors in replication factor estimates influence the sequential peers joining, and the observed replication factor variance.

We report our simulation results in Figures 4.5 to Figure 4.9.

In each of the plot, the *x-axis* shows the number of new peers (per peer in the network at the beginning of the experiment) joining the existing network and structurally replicating an existing partition. We thus do not study any repartitioning and restructuring of the P-Grid key-space partitions in these experiments. The *y-axis* shows the variance of the replication factor across the different key-space partitions. Each experiment was repeated ten times. We plot the mean along with the error bars showing the standard deviation.

Figures 4.5 to Figure 4.8 each show experiment results for a given range of initial replication factors. Each plot corresponds to a different level of maximum replication factor estimation error. Finally, each of these plot shows the results obtained based on $k = 1, 2, 3$ random tentative choices that a peer makes, and then chooses to replicate the least replicated key-space partition. The decision on least replicated key-space partition is based on the erroneous estimates provided to it by the contacted peers of the corresponding partitions. These three cases are represented as *Random*, *2 choices* and *3 choices* in the plots.

In experiments starting with perfect balance in replication factor, we observe that the absolute variance increases, as expected from the stochastic nature of the peer joining mechanism. When there is a high variation in the initial replication factor, we observe that use of the heuristic of multiple choices reduce the variation, as we anticipated in Section 4.5.

As expected, under perfect knowledge (zero estimation error), using more choices of tentative partitions, and choosing the least replicated partition reduces the variance of replication factor. Also the second-order statistical noise, that is, the variance of the variance across different experiments is also less with the use of more choices, as observed from the smaller error bars.

Also expected, if not obvious, is that when no choices are involved, estimation (error) has no role to play. This is also confirmed by the different sets of experiments.

What is reassuring is that even in presence of estimation error, while the variance is more than the case without estimation errors, the performance deterioration is graceful - both the (mean) variance, and the variance of the variance across different experiments being small. More importantly, the benefits of multiple choices stay significant even in presence of estimation errors, making the peer join algorithm practical for the typical P2P environment.

Figure 4.9 provides some more experiment results to directly compare the performance under various degrees of estimation errors when 3 choices are used. We observe that the variance of replication factor across key-space partitions gracefully increases with estimation error.

These simulation results demonstrates the robustness of our heuristic sequential join mechanism (Section 4.5) to balance replication when more peers join an existing overlay sequentially.

4.8.3 Replication load balancing

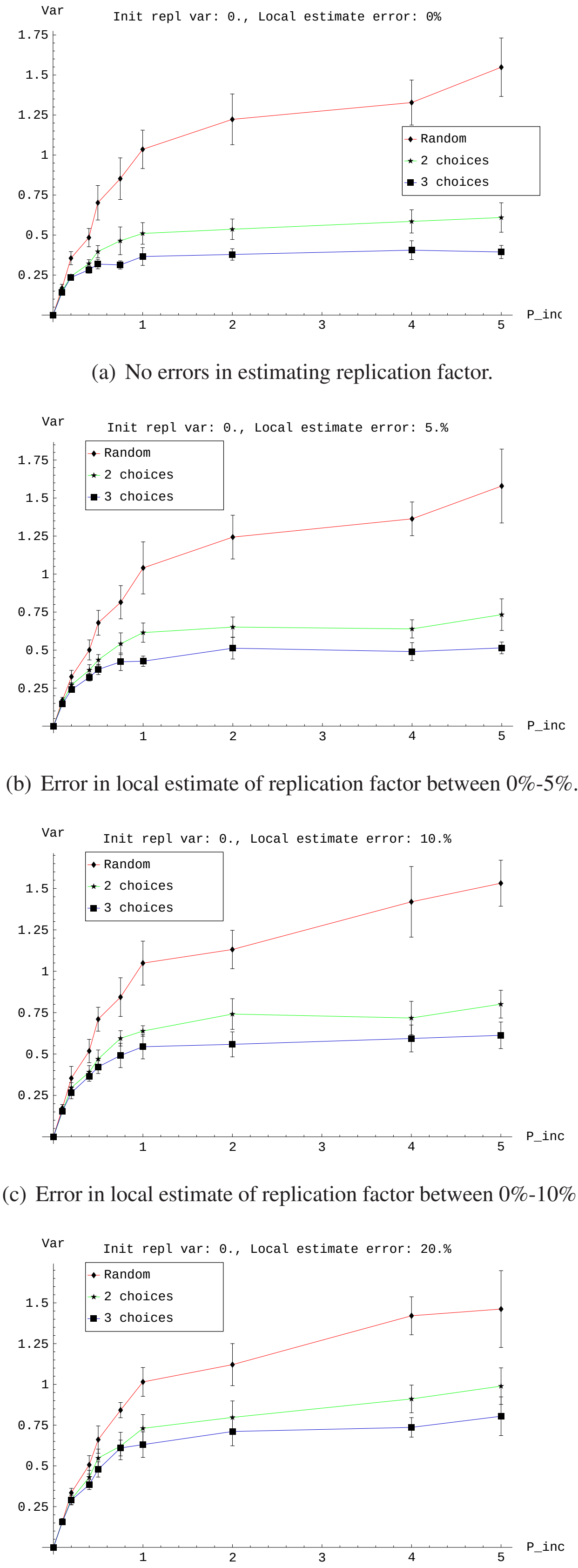
Given a P-Grid that partitions the data space such that the storage load is (approximately) uniform for all partitions, peer migrations are used to balance replication factors for the different partitions without changing the key-space partitioning. For the experiments we chose the design parameters $\zeta = 1.1$ (required imbalance for migration), $\xi = 0.25$ (attenuation of migration probability) and a statistical sample size of 10. These parameters had been determined in initial experiments as providing stable (non-oscillatory) behavior.

The performance of the migration mechanism depends on the number of key space partitions and the initial number of peers associated with each partition. Since the expected depth of the tree structure grows logarithmically in the number of partitions, and the maintenance is expected to grow linearly with the depth of the tree (since each peer uses its local view for each level of its current path), we expect the maintenance algorithm to have logarithmic dependency between the number of partitions and the rate of convergence.

Figure 4.10 shows the reduction of the variance of the distribution of replication factors compared with the initial variance as a function of the number of key space partitions. The simulations started from an initially constructed, unbalanced P-Grid network with replication factors chosen uniformly between 10 and 30 for each of the key space partitions.

In Figure 4.10(a) we compared the effect of an increasing the number of key-space partitions ($p = \{10, 20, 40, 80\}$) on the performance of the replication maintenance algorithm. One observed that the reduction of variance increases logarithmically with the number of partitions. For example, for $p = 80$ the initial variance is reduced by approximately 80%. We conducted 5 simulations for each of the settings. The error bars give the standard deviation of the experimental series.

Figure 4.10(b) shows the rate of the reduction of variance of replication factors as a function of different numbers of peers associated with each key partition. We used a P-Grid with $p = 20$ partitions and assigned to each partition uniformly randomly between k and $3k$ peers, such that the average replication factor was $2k$. The other settings were as in the previous experiment. Actually variance reduction appears to slightly improve for higher replication factors. This results from the possibility of a more fine-grained adaptation with higher replication factors.



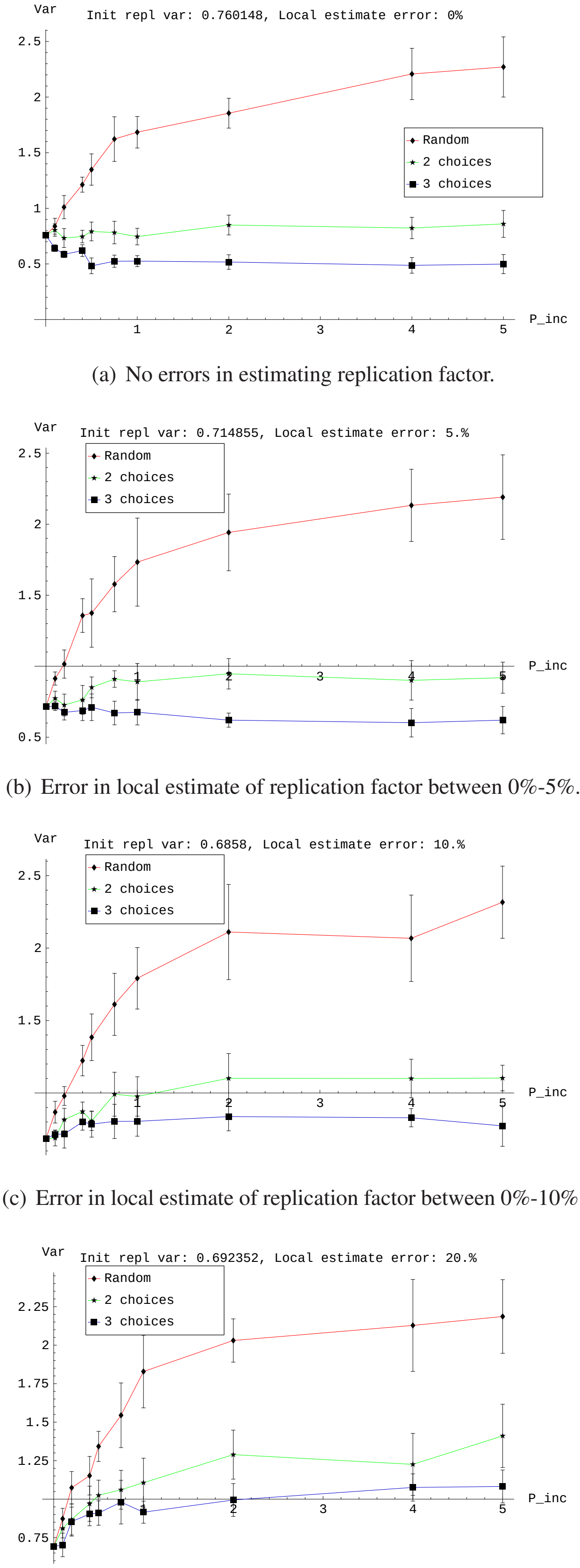
(a) No errors in estimating replication factor.

(b) Error in local estimate of replication factor between 0%-5%.

(c) Error in local estimate of replication factor between 0%-10%.

(d) Error in local estimate of replication factor between 0%-20%.

Fig. 4.5. Sequential join of peers in an existing P-Grid network without variance in replication factor.



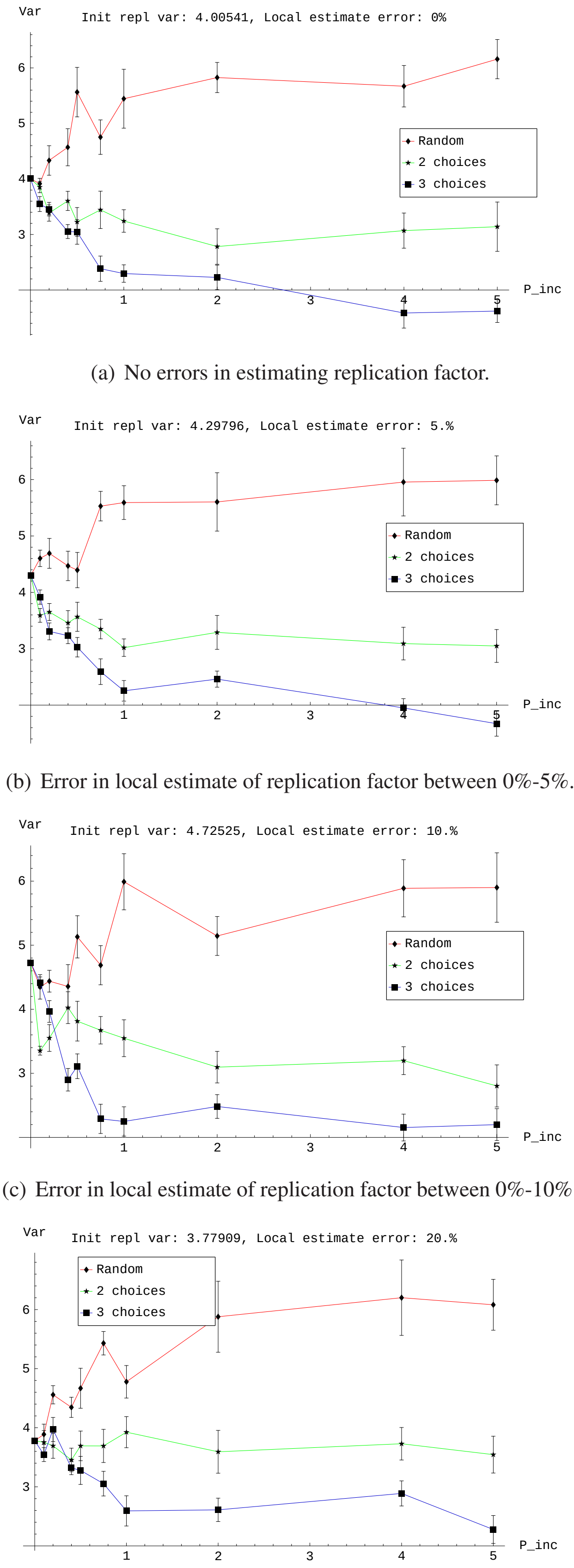
(a) No errors in estimating replication factor.

(b) Error in local estimate of replication factor between 0%-5%.

(c) Error in local estimate of replication factor between 0%-10%.

(d) Error in local estimate of replication factor between 0%-20%.

Fig. 4.6. Sequential join of peers in an existing P-Grid network with variance (<1) in replication factor.



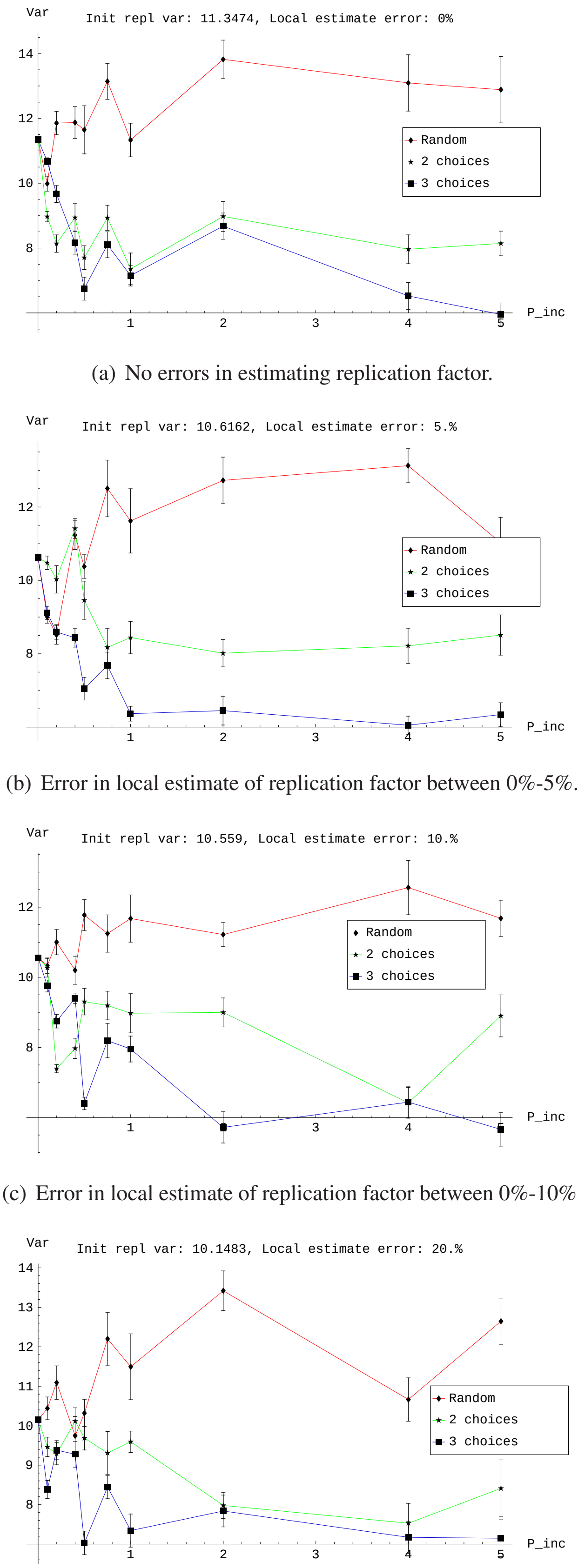
(a) No errors in estimating replication factor.

(b) Error in local estimate of replication factor between 0%-5%.

(c) Error in local estimate of replication factor between 0%-10%.

(d) Error in local estimate of replication factor between 0%-20%.

Fig. 4.7. Sequential join of peers in an existing P-Grid network with variance (∼4-5) in replication factor.



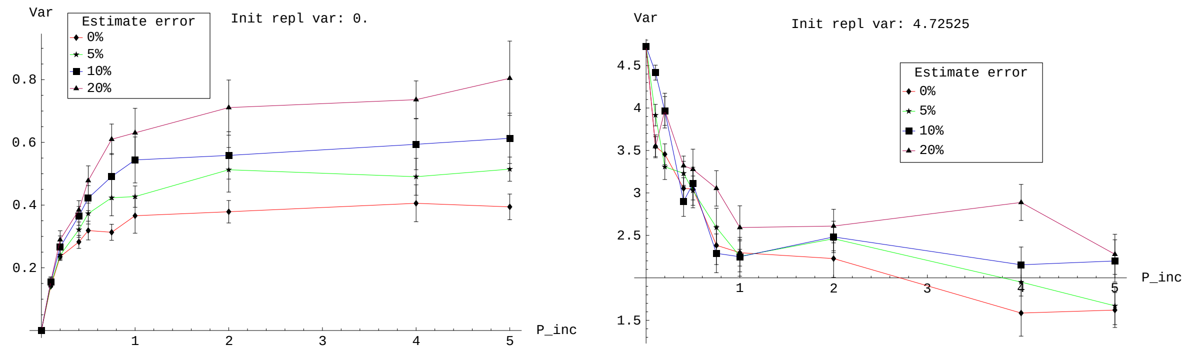
(a) No errors in estimating replication factor.

(b) Error in local estimate of replication factor between 0%-5%.

(c) Error in local estimate of replication factor between 0%-10%.

(d) Error in local estimate of replication factor between 0%-20%.

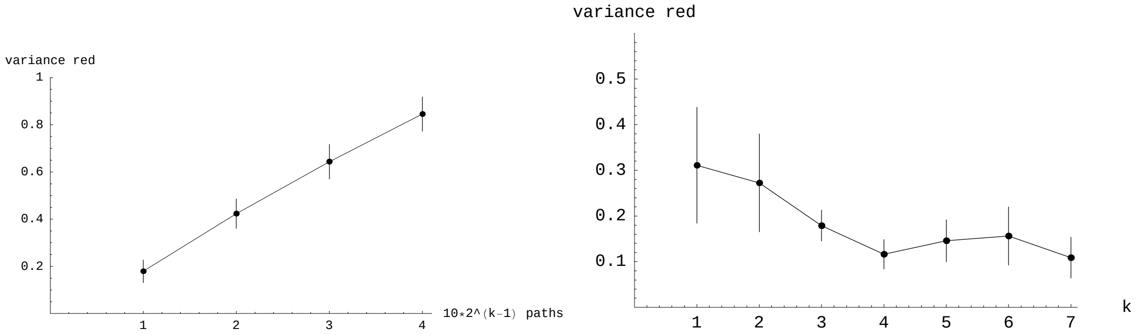
Fig. 4.8. Sequential join of peers in an existing P-Grid network with variance (∼10-12) in replication factor.



(a) No initial variance in replication

(b) Moderate initial variance in replication

Fig. 4.9. Effect of replication factor estimation error at peers on sequential join of peers in an existing P-Grid network



(a) Effect of the number of key-space partitions

(b) Effect of the peer population for same key-space partitioning

Fig. 4.10. Maintenance of replication load-balance

4.8.4 Simultaneous balancing of storage and replication load in a dynamic setting

In this experiment we studied the behavior of the system under dynamic changes of the data distribution. Both storage load balancing by restructuring the key partitioning (i.e., extending and retracting paths) and replication balancing by migration were performed simultaneously. We wanted to answer the following two questions: (1) Is the maintenance mechanism adaptive to changing data distributions? (2) Does the combination of restructuring and migration scale for large peer populations?

For the experimental setup we generated synthetic, unbalanced P-Grids with $p = 10, 20, 40, 80$ paths and chose replication factors for each path uniformly between 10 and 30. Thus, for example, for $p = 80$ the expected peer population was 1600. The value δ_{max} was set to 50 and the dataset consisted of approximately 3000 unique Zipf-distributed data keys, distributed over the different peers such that each peer held exactly those keys that pertained to its current path. Since the initial key partition is completely unrelated to the data distribution the data load of the peers varies considerably, and some peers temporarily hold many more data items than their accepted maximal storage $2\delta_{max}$ load would be. Then the restructuring algorithms, i.e., path extension and retraction used for P-Grid construction and path migrations used for replication load balancing, were executed simultaneously.

Table 4.1 shows the results of our experiments. We executed an average of 382 rounds in which each peer initiated interleaved restructuring and maintenance operations, which was sufficient for the system to reach an almost steady state. R_{σ^2} is the variance of the replication factors for the different paths and D_{σ^2} is the variance of the number of data items stored per peer representing replication and storage load balancing respectively.

Number of peers	Number of paths		R_{σ^2}		D_{σ^2}	
	initial	final	initial	final	initial	final
219	10	43	55.47	3.92	180,338	175
461	20	47	46.30	10.77	64,104	156
831	40	50	40.69	45.42	109,656	488
1568	80	62	35.80	48.14	3,837	364

Table 4.1. Results of simultaneous balancing

The experiments show that the restructuring of the network as well as replication balancing was effective and scalable: (1) In all cases the data variance dropped significantly, i.e., the key space partitioning properly reflects the (changed) data distribution. Because of the randomized choices of the initial P-Grid structure and the data set, the initial data variance is high and varies highly. It actually depends on the degree to which the randomly chosen P-Grid and the data distribution already matched. From the case $p = 40$ (number of initial paths), we conclude that this has also a substantial impact on the convergence speed since more restructuring has to take place. Actually, after doubling the number of interactions, the replication variance dropped to 20.93, which is an expected value. (2) With increasing number of replicas per key partition the replication variance increases. This is natural as fewer partitions mean higher replication on an average and thus higher variance. (3) With increasing peer population the final data variance increases. This is expected as we used a constant number of interactions per peer and the effort of restructuring grows logarithmically with the number of key partitions.

The algorithms do not require much computation per peer hence have a low overhead. Simulating them, however takes considerable effort: A single experiment with $3*10^5$ interactions for the results in this section took up to 1 full day. Thus we had to limit the number and size of the experiments. Nevertheless they indicate the feasibility, effectiveness and scalability of the algorithms.

4.9 Related work

The fundamental problems to address for any large-scale distributed indexing systems are distributed index construction and load-balancing. Traditionally structured overlay networks, mainly based on distributed hash tables (DHTs), have followed sequential construction and maintenance strategies (online balancing) [63, 117, 154, 163]. In contrast to this, our approach applies a highly parallel strategy which speeds up the construction process, takes advantage of the distributed computing resources by allowing the participants to work independently and asynchronously on the construction, and enables the merging of independently created indices.

To address load-balancing, the standard strategy of overlay approaches is to use uniform hashing of keys to remove skew from the distribution. However, this defeats the applicability of overlay networks to semantic processing of keys (range queries, etc.). Thus in standard overlay approaches, typically an additional index on top of the overlay network needs to be created [136]. The advantage of this approach is its universal usability on top of any DHT. However, it is considerably less efficient than our approach since semantically close data items are not necessarily stored close to each other in the overlay network (high fragmentation), and hence, multiple overlay network queries are required to locate all the semantically close content. Thus, apart from the additional effort of constructing an additional index, such schemes additionally suffer from inefficiencies throughout the operational phase of the system.

In contrast to that, we build a trie that clusters semantically close data, thus realizing *in-network indexing* which enables more efficient query processing. This comes at the expense of a more sophisticated construction process for such data-oriented overlay networks. Additionally, more complex online load-balancing strategies have to be applied, as presented.

Online load-balancing is widely researched area in the distributed systems domain which often been modeled as “balls into bins” [135]. Traditionally, randomized mechanisms for load assignment, including load-stealing and load-shedding and power of two choices [121], have been used, some of which can partly be reused in the context of P2P systems [32, 94], but with limited applicability. For example, [94] provides storage load-balancing as well as key order preservation to support range queries, but at the cost that efficient searches of isolated keys can no longer be guaranteed.

Most of these load-balancing approaches look into the relative load at peers, counted in term of the number of keys stored at each peer. We consider the absolute (storage) load peers are willing to contribute to the overlay. End users are concerned more about their absolute load, and different key-value pairs (which is stored in the overlay) require different amount of resource. Our mechanism of allowing peers to decide the absolute load, and balancing the replication based on available capacity in the system takes into account such practical concerns.

The dynamic nature of P2P systems is also different from the online load-balancing of temporary tasks [21] because of the lack of global knowledge and coordination. Moreover, for replication balancing, there are no real bins, and actually the number of bins varies over time because of storage load balancing, but the balls (peers) themselves have to autonomously migrate to replicate overloaded key spaces. Also, for storage load balancing, the balls are essentially already determined by the data distribution, and it is essentially the bins that have to fit the balls by dynamically partitioning the key space, rather than the other way round.

A distinguishing property of our approach to all other related load-balancing strategies is actually that we address two, sometimes conflicting load-balancing problems—storage load, i.e., balancing the amount of storage used at the nodes, and replication load, i.e., ensuring approximately uniform data availability by having roughly the same number of replicas per data partition. The first step in that direction was a heuristic key space bisection proposal [6]. In comparison to the heuristics, we now exhaustively analyze and refine the bisection mechanism, in order to better understand and guarantee superior load-balancing characteristics in the overlay network emerging from the recursive use of the bisection algorithm. Additionally, we not only simulate the construction process, but verify several of the analytically predicted properties using a fully-fledged implementation (P-Grid) deployed on PlanetLab to validate some of our analysis and simulation results with a moderately large-scale experimental data. The PlanetLab experiment results are reported later in Chapter 7. The overlay network is already used as a substrate for two data-oriented applications—a peer-to-peer search engine [30] (http://www.alvis.info/) and a semantic overlay network [3].

Furthermore, most existing load-balancing as well as overlay network construction mechanisms have so far been sequential. However, the need for faster overlay construction has recently generated interest in the research community, as is evident from some recent publications [19, 92].

Both [19] and [92] use random interactions among peers, induced potentially by the original unstructured topology, and try to build a desired topology, by essentially trying to sort the peers according to their identifiers that are generated at the beginning of the process. These mechanisms can again be used for overlay networks construction which support search of keys generated by uniform hashing, since then peer identifiers can be simply generated using uniform hashing, as there is no skew in the load-distribution. However, for data-oriented applications such a mechanism has a critical limitation, since peers are predestined for the amount of load (based on the whole set of peer identifiers generated at the beginning of the process), and there is no flexibility or adaptivity for load-balancing, particularly if the load is skewed. Our scheme on the other hand adaptively creates the key space partitions and assigns peers to these partitions based on load characteristics, and is thus a more generic parallel overlay construction mechanism. For the special case of uniform load distribution (as it is traditionally assumed in DHTs using uniform hashing), we can easily construct a load-balanced overlay by requiring $p = 0.5$ in each step of the partitioning.

The Y_0 extension of Chord [72] deals with heterogeneity by allowing resource rich peers to participate in the overlay as multiple virtual peers responsible for a contiguous stretch of the identifier ring. A similar approach may be used in P-Grid where resource rich peers can assume responsibility for (replicate) multiple contiguous key-space partitions. This essentially means that such a peer will be responsible for a subtree instead of just the leaf-node of the tree abstraction.

In our approach we assumed that the load-balance is to be achieved with the collaboration of other peers. Such an assumption is in the same spirit as most other related work on load-balancing in peer-to-peer systems. In reality, this need not be the case always. The simplest and passive form of departure from such a collaborative norm is freeriding [13], and hence it may be necessary to enforce cooperation. One possible way to realize cooperation is to provide incentives [31]. However, the system may also come under attack from proactive attacks. To point out one example - resource rich peers may participate with multiple peers to gain access to a large pat of the key-space. Such an attack is commonly known as the Sybil attack [56], which can then be used for further distributed denial of service attacks (DDoS). Using virtual peers to deal with and heterogeneity in the system is for instance in direct contradiction with Sybil attack. Thus, security and enforcement of cooperation are important aspects of load-balancing in overlay networks, which we, as well as most existing related work have not addressed so far, and will be critical for deployment of structured overlays out in the open. Nonetheless, the load-balancing techniques we study are very relevant, even if the overlays are to be deployed within more trusted environments, say within an organization or a single trust domain like PlanetLab, or even in a less trusted environment in conjunction with other mechanisms to enforce cooperation holding participating peers accountable for their (in)actions and secure the system against active attacks.

5. A first-order balancing of query-load

"Sometimes questions are more important than answers" — Nancy Willard

5.1 Introduction

Balancing number of keys per peer for both uniform as well as skewed distribution of keys over the key-space has been extensively studied for structured overlays over the last few years as discussed in detail in the previous chapter. Query related load - both in terms of the traffic (query forwarding) as well as query answering also need to be balanced. The main focus of this chapter is to look at the aspects of balancing query-load in structured overlays, and compliments the work on balancing of storage-load at peers and structural replicas in the overlay network.

Access to different keys may be with different frequency, and the employed load-balancing techniques will need to account for such skews also. Dealing with skewed access load on keys in structured overlays has been dealt mostly with caching heuristics, where both the placement and frequency of caching is typically ad-hoc [39, 46, 140]. This chapter is a step towards a more objective look at the replication/caching strategies for balancing skewed query-load in structured overlays.

Our current work is a first important step towards query-load balancing [53], and as we will infer from our current results, balancing query-load in structured overlays is still an outstanding practical issue because optimal first-order query load-balancing as we achieve currently is essential but in itself is not sufficient.

Since first-order balancing is still essential, and compliments the previously studied storage load and structural replica balancing problems, and the results lead us also to interesting general observations on structured overlay networks which have not been reported in the literature yet, we present them here.

Here we show that balancing query load in structured overlay networks, i.e., if replication of a key is proportional to query frequency for the key, is also optimal with respect to reduction of expected search latency given a limited aggregate storage capacity in the system. This result may appear obvious at the first glance, but search cost minimization and load-balancing are two different constraints in a system, and meeting one may be in conflict with another. Indeed, both for unstructured overlay networks [111] and for mobile broadcast environments [12] it has been shown that following a square-root rule for replication is optimal for access latency reduction, and has thus often been used generally as a rule of thumb. The square-root rule is to replicate or cache objects proportional to the square-root of the access/query frequency for these objects.

Additionally, in the context of unstructured overlays, if objects are replicated proportional to the square root of their popularity to reduce (minimize) search latency, the query-load is not balanced across the peers. Moreover, achieving even approximately square-root replication without global knowledge and coordination is complicated [111].

Thus, based on our result, we conclude that for structured overlay networks accidentally replication can be simultaneously optimized with respect to minimizing search latency as well as non-uniform query loads, apart that approximate proportional replication is straightforward to achieve even in a distributed manner.

Caching techniques are inherently adaptive, but such adaptivity in itself does not imply self-organization. For practical query-load balancing we need to determine an appropriate rate at which caching should be done (the constant of proportionality) given absolute query rate and cost of caching in an overlay. We also see in this chapter that even if appropriate number of replicas are chosen to match query-load, because of randomness in the (greedy) route forwarding process, there is huge variance in query-forwarding as well as query-answering loads. Determining alternative routing strategies taking into account the load at peers, to reduce variance of load across peers (second-order balancing) are also necessary. Determining algorithmic parameters (for determining the number of replicas) or modifying the current query forwarding schemes in order to dynamically balance query-forwarding load would require a more rigorous understanding of the (modified) routing and replication strategies, and may have self-organizing characteristics.

Before delving into the study of caching-based query-answering (access) load-balancing, in Section 5.2 we look into balancing degree distribution in randomized structured overlays in order to eliminate hot-spots created by the overlay (instantiated graph) itself. Balancing in-degree eliminates any systematic query traffic hotspots at peers with larger (than other peers) in-degree. Balancing the degree-distribution of different peers is also desirable so that peers have comparable overhead to maintain their routing tables.

In Section 5.3 we determine the criterion to decide whether a specific cache placement strategy is optimal with respect to search cost, that is, given a number of caches for an object, what is the best expected search latency we can achieve for that object. Then we determine the *optimal replication strategy* for structured overlays in Section 5.4. There we prove that for a wide range of popular structured overlays (with logarithmic search cost), caching proportional to the query load which by definition provides a first-order query answering load-balancing, also minimizes the expected search latency for a system with fixed storage capacity, assuming that these caches can be placed optimally. In Section 5.5 we determine that apart some approximations which are necessary in any realistic system, it is indeed possible for a broad class of (at-least tree) structured overlays to place caches optimally to achieve optimal reduction of search cost. We present the results in Section 5.6.

These early results expose the *limitations of caching* techniques for (a) not only improving search latency in structured overlay networks (because of the enormous storage consumption for marginal gains in latency), (b) but we also expose that even the optimal first-order balancing of query load based on caching is not good enough because of statistical noise. (c) Furthermore, similar statistical noise leads to hotspots (congestion) in an overlay network even when the routing network graph in itself has good in/out-degree distributions.

These results need to be seen in context with the current understanding of structured overlays, where caching has been proposed for constant time lookups [137], and often caching is proposed to alleviate hot-spots without further experimental validation of the quality of the actual load-balancing achieved.

Our results raise important concerns also about the effectiveness of any of the existing heuristics found in the structured overlay literature with respect to alleviating query-load. Furthermore, we see that greedy routing as used in most overlays can lead to congestion.

These observations highlight the need to use some second-order load-balancing techniques. We outline some potential approaches to achieve such second order balancing and conclude in Section 5.7.

This work makes the following assumptions: (i) The number of keys a peer is responsible for is already balanced, which is, as discussed in previous chapter, more or less achieved under various settings - range-partitioned or DHTs - by [8, 32, 63]. (ii) All peers have same[1] and limited storage, part of which is dedicated to store the keys it is responsible for based on its role in the index, and the rest is used in order to dynamically cache some keys in order to alleviate load-balancing and improve search latency. (iii) The routing network itself does not lead to any systematic hot-spots because of large variations in degree distribution at each peer. This is a critical issue particularly for overlays with randomized routing networks [8, 154] (deterministic ones [163] already have good balance), which has not been studied in the literature in great detail, but can be achieved fairly well using standard techniques like power-of-two choices as we show next.

5.2 Route in-degree in randomized overlay topologies

We will like to eliminate any systematic bottle-necks or hot-spots in the routing network. In order to do so, we need overlay networks where peers have their out/in-degrees well balanced.

There are several overlay networks where once the peers assume identifiers in the identifier space, the routes are chosen deterministically. Apart from close to logarithmic out-degree (in terms of peer population) for all peers, a well-designed deterministic topology also tends to have good balance of in-degree (logarithmic), such that there is no routing hot-spots. This is, for instance, the case for the original Chord proposal.

While the early DHT proposals like Chord considered deterministic choice of routes, randomized choices of route have several advantages - it naturally means more choices so that there is more flexibility to incor-

[1] Homogeneity can be achieved by using multiple virtual peers for resource rich peers.

porate locality as well as other considerations like reliability of the target peers. Route maintenance in such overlays, referred as randomized routing networks [115] is also easier. Such networks include small-world networks [98, 26], Symphony [119] and SkipGraphs [20], as well as other topologies like Pastry [154], P-Grid, Kademlia [120] and randomized versions [115] of other deterministic networks to name a few.

However a simple randomized choice as advocated so far in most of these systems will lead to poor balancing of in-degree distributions, because of the *balls into bins* effect, where the peers can be considered as the bins, and the in-coming links the balls. In the context of in-degree balancing, the problem has however not been addressed in the literature. Symphony [119] tries to bound in-degree using a global bound for incoming links. Such a global bound hard-coded in a system design is both restrictive as well as unnecessary. The standard randomized mechanisms like use of multiple choices, e.g., *Power-of-two choices* [121] can however be used effectively. Power-of-two-choices has in fact already been used in the context of storage-load balancing in DHTs [32].

Such a balancing of in-degree can be achieved in an existing overlay network by running a background process of rewiring the network (changing routing table entries at peers). In a real network, such multiple choices can be obtained as follows. A peer queries for (two) random keys belonging to the region of the key space for which it requires routing entries, and then gather from the contacted peers their in-degrees (which can be locally observed by them), and thus choose the less loaded peer as a candidate entry for the routing table.

The analysis in Section 3.2.1 to determine the expected search cost in P-Grid only assumed uniform random choice of routes for each level at each peer. Even after in-degree balancing, the assumption of uniform random route choices hold. So balancing the in-degree in randomized overlays according to the above mentioned mechanisms do not affect the expected search cost. Hence it has no bearings on the analysis in the subsequent sections where we determine the optimal number of replicas to minimize search cost based on query frequency.

5.3 Replication and search cost

The latency to search resources in a wide area network is an important performance metric. Structured overlays provide a reasonably good guarantee for the search latency. However if there is extra capacity among the peers to selectively replicate some keys more often, these keys may be found with shorter latency. Replicating highly queried keys more frequently can then reduce the expected search cost on the overlay. Similar intuition is used for diverse distributed systems including web caching, content delivery networks and unstructured overlays. Here we look into how replication affects search cost in structured overlays.

Definition 5. *Let $\sigma(N, r)$ be the expected cost to search a key in a structured overlay network (using any particular topology/routing mechanism) with N peers, and r replicas of the key.*

Consider that we form clusters of r peers such that there are N/r logical units, with one of these clusters consisting of the r peers storing the replicas of the concerned data item. These clusters may be considered to be connected using the same topology, as the original network of N peers. Then exploiting the self-similarity of the structured overlay networks, the expected cost of locating the particular cluster is $\sigma(N/r, 1)$, independent of the topology. Stated otherwise, we can assume r parallel networks, each with population of N/r, to achieve a search cost of $\sigma(N/r, 1)$ for a key replicated r times in a network of N peers.

Observation 1. *In a structured overlay network of N peers, if a data item is replicated r times, then there exists (for a wide range of structured overlays) a policy to place these replicas so that at least any one of the r replicas can be located with an expected search cost $\sigma(N, r)$ of $\sigma(N/r, 1)$.*

Of the various existing replication strategies (summarized in Section 2.5.2) for diverse overlay topologies, none is known to have better search performance, and most even do not match up with the above mentioned potential search cost reduction. In the following we use this known best achievable search cost reduction criterion to define optimality for replica placement strategies. Whether even further search cost reduction is possible in an overlay by cleverly placing the replicas and possibly even changing the routing strategy (subject to the same topology) is an interesting open question.

Definition 6. *The optimal replication placement strategy from the search cost perspective in an overlay network is the one which guarantees that $\sigma(N, r) = \sigma(N/r, 1)$ for a data item replicated r times.*

Note that when different data items need to be replicated with different frequency, for example, for query-load balancing; it may not even be possible to exactly form such clusters (or juxtaposed networks) in practice. In such cases we can only aim to determine the best replica placement in the network in order to be as close to this optimal as possible.

Uniform structural replication for any topology is optimal with respect to search latency (and cost)[2]. However, such a placement strategy ceases to be optimal if different items are replicated with different frequency, and as a consequence, structural replication is not a suitable placement strategy for query-adaptive replication in general.

5.4 Optimal query-adaptive replication strategy for structured overlays

For most realistic applications queries are non-uniformly distributed. The standard solution approach both in an internet setting as well as in any overlay network in order to provide good load-balancing as well as guarantee low latency is to cache (replicate) the queried objects in a query-adaptive manner. Caching in standard networks is a mature technology, e.g., Akamai [15]. Here we want to systematically study the implications of caching in structured overlays. For the rest of this chapter we consider the family of structured

[2] By uniform we mean that each partition, and hence each item is equally replicated.

overlays which have logarithmic expected search cost, i.e., $\sigma(N, 1) = c\, log(N)$ for a peer population of size N, where c is an overlay topology and routing algorithm dependent constant. Additionally, we assume for the time being, that it is indeed possible to optimally place replicas in the network, such that for a data item d_i with r_i replicas, $\sigma(N, r_i) = c\, log(N/r_i)$.

Theorem 3. *The replication strategy which is optimally adaptive to query frequency in structured overlays with logarithmic search cost is the strategy which proportionally (with respect to query frequency) replicates data items, assuming replicas are placed optimally. (Placement strategy optimality as defined in Definition 6.)*

Proof: Consider that peers have the possibility to replicate a data item such that replica placement is optimal with respect to the search cost. Thus to say, search cost for any one instance of a data item replicated r times in a network of N peers is $c\, log(N/r)$.

Assume also that there are M distinct data items in total, and the access (query) probability for data item $d_i, i = 1, \ldots, M$ equals q_i and d_i has r_i replicas.

The total storage used in the system is then (with R being the average replication factor in the system)

$$\sum_{i=1}^{M} r_i = RM$$

The expected access time (in terms of messages) to access any data item is

$$T = c \sum_{i=1}^{M} q_i log(\frac{N}{r_i})$$

In order to determine the optimal allocation of replicas for a given global average replication factor R (essentially determined by the total storage capacity of the system, and the total number of distinct data items in the system) we have to solve the system of partial differential equations

$$\frac{\partial T}{\partial r_i} = 0, i = 1, \ldots, M$$

Substituting $r_M = RM - \sum_{i=1}^{M-1} r_i$ and differentiating we obtain

$$\frac{\partial T}{\partial r_i} = -q_i \frac{1}{r_i} + q_M \frac{1}{r_M} = 0$$

from which we conclude that $\frac{q_i}{r_i}$ must be constant $\forall i = 1, \ldots, M$.

Note that there are thus two dimensions of optimality for replication:

(a) Query-adaptivity determines how many replicas to maintain for individual data items to minimize expected search cost. Unlike in unstructured overlays [111], it so happens that for a large family of (logarithmic search-cost) structured overlays, this simultaneously provides a first-order query-load balancing.

(b) The placement strategy determines, for r replicas of a data item to be placed in the network, where exactly these replicas are to be placed in order to reduce search latency.

Next we explore the specific placement strategy for some important groups of structured overlays, particularly determining the optimal placement strategy for P-Grid.

5.5 Optimal replica placement

In recent years, many routing network topologies have been proposed. There have been recent attempts to derive abstracted, generalized models [11, 139] for these diverse topologies. However, none of these models are exhaustive, and in absence of any universal abstraction, we limit the discussion to tree-structured overlays [11] which particularly include P-Grid [8] and XOR topology based Kademlia [120] and other PRR [133] variants like Pastry [154].

In these virtual tree based access structures, a peer is responsible for all data items corresponding to a leaf node of the tree. This peer then acts as the primary replica for the data items. When a particular data item needs to be replicated, the optimal replication strategy places the replicas at the other leaf nodes which share common intermediate nodes in the tree. In effect, this is like replicating in the tree at a higher level (or reducing the depth of the search tree), since the data item will then be found at any of all the peers which split the intermediate (imaginary) node. This process can be repeated, successively propagating the replication to larger number of peers, which all share the same internal nodes in the tree abstraction. Such a placement strategy effectively leads to logically coalescing of the key space partitions and has been shown in Figure 5.1. In terms of the generic overlay network scheme [11], essentially, the search space can be seen as being split among peers recursively (when joining). So a peer should replicate the content at other peers with whom it had conducted the splitting operation, such that effectively the data is available in the logically coalesced search space.

The basic idea of such a replication scheme is to coalesce r partitions which are topologically closest (based on incoming links), effectively leading to a logical network of N/r partitions. Lookups from different clients for the same data item tend to converge to the same set of peers, such that a replica is located earlier, hence speeding up the lookup process. The N/r partitions are connected with the same topology as the original N partitions, hence the search speeding up matches the optimality criterion in the ideal case. In realistic scenarios, overlay networks are not evenly partitioned, and this replication approach, while being the best strategy, may not quite match the theoretical optimum.

Even though Chord does not have a direct tree-mapping nor is modeled by the generic abstraction [11], the routing network resembles a tree-like access structure from the perspective of individual peers. If a peer p_1 is the primary responsible peer for a data item, p_1 should replicate this data item at the closest preceding peers from which p_1 has incoming links. This information is locally available to p_1, (from the route maintenance operations). This replication scheme is in principle a small variation of the caching strategy (along

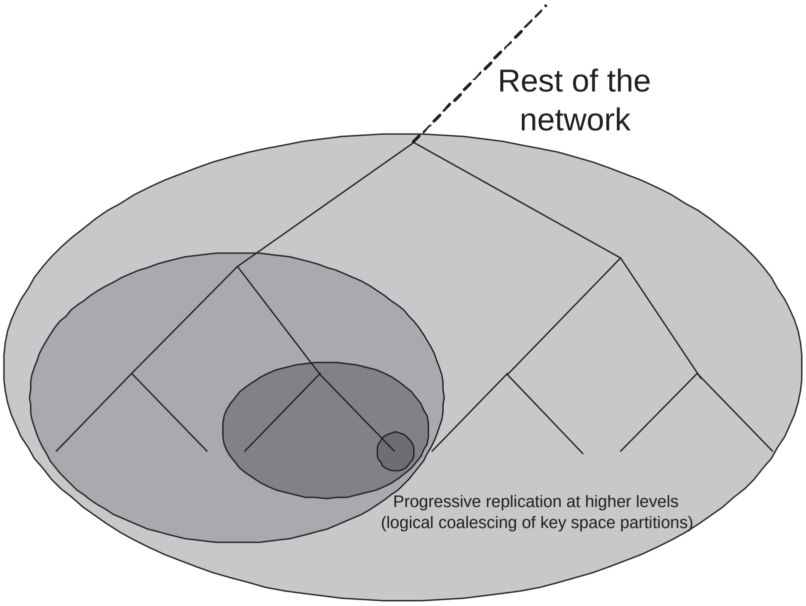

Rest of the network
Progressive replication at higher levels
(logical coalescing of key space partitions)

Fig. 5.1. Replication in tree structured networks. For many structured overlays there is no actual global tree abstraction, but the access from any peer to other peers in the network can still be locally viewed at as a tree.

query downstream) which CFS [46] already uses, and may have marginal influence in terms of search cost improvement. However, by choosing the closest incoming links at a peer (network maintenance protocols require these peers to communicate periodically in any case), instead of necessarily choosing the peers from the direction the last query arrived, the replicas are placed in a deterministic manner, based purely on locally available information. This placement strategy can also be used to improve fault tolerance in CFS [46]. Instead of having a separate policy for placing replicas for fault tolerance, and a separate policy for cache placement, thus making efficient use of storage and bandwidth required for.

So far we determined the replication factor purely in terms reducing expected access cost. We have to instead choose the larger of the two numbers based on load-balancing and fault-tolerance criteria, to be used as the replication factor. This argument is universally true for all structured overlays.

5.6 Results

5.6.1 Balancing in-degree in randomized routing networks

Multiple choices based reduction of variance in balls-into-bins problems is a standard solution technique. We evaluate its effectiveness to reduce variation of in-degree in randomized structured overlays like P-Grid for various network population sizes[3] N between 2^5 to 2^{15}, and provide the results in Figure 5.2.

[3] Here we consider number of peers to be the same as the number of key-space partitions $|\Pi|$ in P-Grid, that is to say, we ignore structural replication.

We observe from Figure 5.2(a) that the standard deviation of in-degree grows when using randomized choices, while with power-of-two choices, the standard deviation stays more or less the same, thus demonstrating the effectiveness of the power-of-two-choice based mechanism with scale. Figures 5.2(b)and 5.2(c) show histograms of in-degree distribution for networks of size 2^{15} peers without and with the use of load-balancing. Noteworthy is the fact that the in-degree is concentrated around $log(N) = 15$, unlike the case of randomized choice, where the bell-shaped distribution has a much wider spread. This clearly demonstrates the effectiveness of a simple and time-tested load-balancing mechanism to provide very good load-balancing without the need of any global information.

Deterministic networks have good in/out-degree distributions. While randomized networks also show good out-degree distribution, we have validated that use of a standard power-of-two-choice solution can be used to achieve good in-degree balance in randomized topologies. This ensures that for queries generated randomly at any peer, destined to any other peer, the routing network in itself will not cause any systematic bottle-necks. Note that this is the best that can be done with the routing network irrespective of the query distribution. Moreover balancing out/in-degree also balances the route maintenance load on each peer.

5.6.2 Numerical evaluation: Square-root vs. Proportional replication

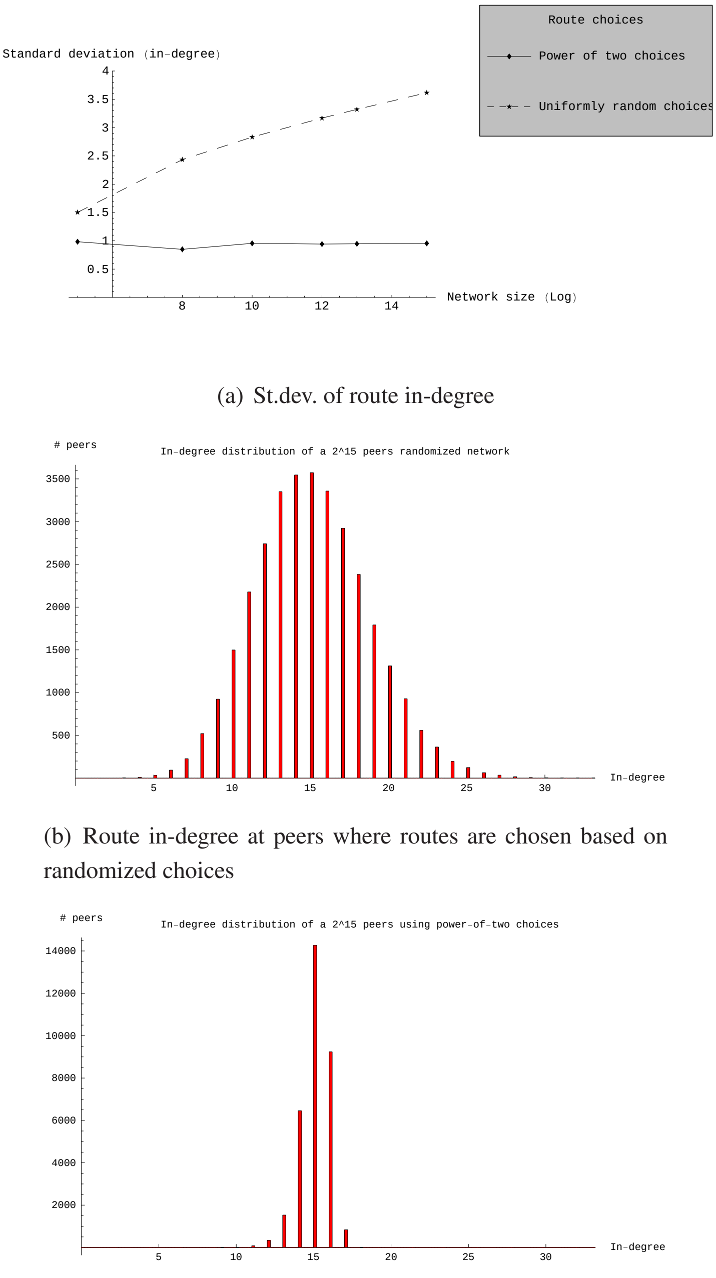
Replication proportional to the square-root of the query frequency has been shown to minimize expected search cost for unstructured P2P systems [111]. We choose this strategy as the baseline to compare with our proposal of replicating directly proportional to the query frequency. We consider a Zipf-distribution (parameter 0.8614) of queries. We consider that there are $M = 4096$ unique keys d_i that are queried with relative frequencies q_i which vary between 1 and 256. We further assume an overlay with 1024 partitions of equal size (e.g., for a tree-structured overlay, a balanced tree). That is, ignoring any structural replication, its a network of $N = 1024$ peers. We additionally assume that the keys are distributed over these partitions in such a manner that it would be possible to optimally place the replicas by exploiting only the available storage in the appropriate peers (as determined by optimal placement criterion). So to say, this numerical analysis makes several idealizing assumptions.

Consider that there is storage capacity for an average replication factor of $R \geq 1$ in the system[4]. For the proportional replication case, there are thus $r_i = Max(1, \alpha_1 q_i)$ replicas and for the square-root replication strategy, there are $r_i = Max(1, \alpha_2 q_i^{0.5})$ replicas for data item d_i, where α_1 and α_2 are determined under the constraint of limited available storage in the system, that is $\sum_{i=1}^{M} r_i = RM$.

We numerically evaluate and show in Figure 5.3 (y-axis) the expected search latency and cost $0.5 \sum_{i=1}^{M} q_i log_2(\frac{N}{r_i})$ for various values of overall storage capacities (x-axis).

This demonstrates that for the same average storage capacity R, in structured overlay networks with logarithmic search cost (in terms of network size), proportional replication achieves a lower search cost in

[4] Note that $R = 1$ actually means there is no replication, but only the original copy is stored.



(a) St.dev. of route in-degree

(b) Route in-degree at peers where routes are chosen based on randomized choices

(c) Route in-degree at peers where routes are chosen based on power-of-two choices

Fig. 5.2. Route in-degree balancing in randomized topologies

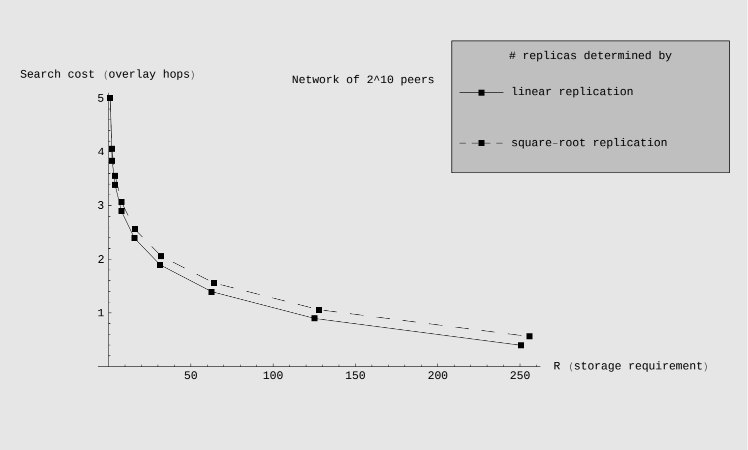


Fig. 5.3. Numerical evaluation: Proportional (linear) vs. Square-root replication for Zipf (parameter 0.8614) distributed queries for various storage capacities

comparison to the square-root replication strategy which was optimal for unstructured networks [111], thus highlighting a fundamental difference of the effect of replication in these two broad classes of P2P systems - structured and unstructured.

We also observe that search cost reduction consumes storage exponentially, which further shows the fundamental limitation of replication/caching to improve search latency in structured overlays, making systems like Beehive [137] which tries to achieve $O(1)$ lookup using replication impractical.

5.6.3 Simulations: Optimal query-adaptive replication

Setup and workload: We simulated a randomized tree-structured network of 2^8 peers (specifically using the P-Grid routing topology but without any structural replication), with the routes chosen either (i) randomly or (ii) using the power-of-two-choices to balance the in-degree at peers.

Queried objects were replicated according to the optimal placement strategy described earlier, with one additional replica created for each query received by any current replica. In case of lack of storage space, least recently queried object (locally perceived at individual peers) was removed in order to replicate a newly queried object. This is an approximate way to achieve proportional replication[5].

Each peer initially held 5 unique data items, thus there were 1280 unique keys. Each peer had a capacity for R_{pot} data items (including the original), where R_{pot} was varied between 1.2 to 20. Queries with relative frequencies Zipf-distributed (parameter 0.8614) were originated at random peers chosen uniformly. Approximately 16700 queries were issued.

[5] Note that the constant of proportionality here is implicitly chosen as 1 which may result in enormous data movement load as well, and in practice a less aggressive caching may be viable.

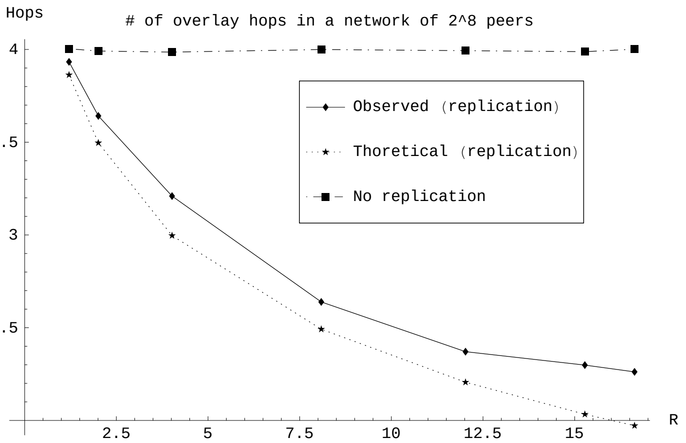
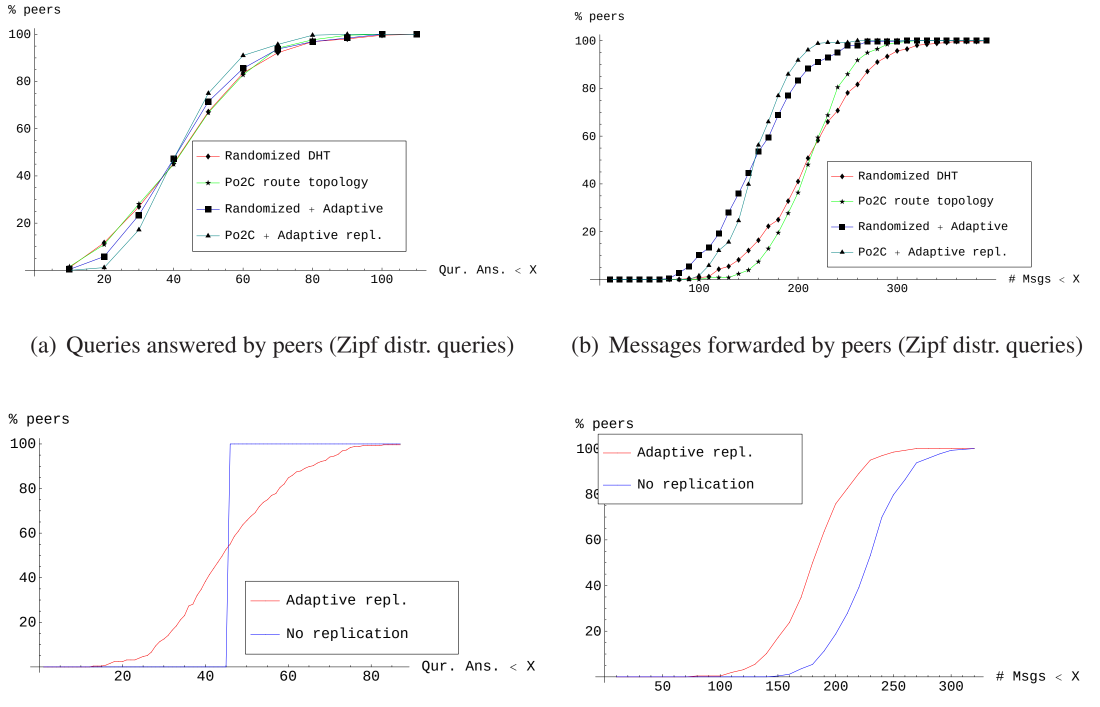


Fig. 5.4. Storage space vs. search latency trade-off under Zipf-distributed (parameter 0.8614) queries

Experiment results: In Figure 5.4 we show the average number of hops required to answer the queries for different values of average replications R. First thing we noticed in the experiments was that the value of R stay smaller than available storage space R_{pot} since replication placement is constrained by the placement optimality criterion and thus any arbitrary available space can not be used. For example, in our experiments, with $R_{pot} = 20$, only $R \approx 16$ was used. We also notice that for the storage space used, the average search cost is slightly higher than as expected from theory, even though it follows the same trend. This is because before enough replication is done, the queries can not leverage the advantage of replication and hence require more hops. The analysis in Section 5.4 assumed the replicas were already in place. This is a practical limitation of a caching scheme in a realistic setting.

We show *cumulative distribution functions* in Figure 5.5 to summarize the load-balancing results, where two different measures of load are used: (i) the number of query messages forwarded by peers, as well as the (ii) the number of queries actually answered by peers (possible when the peer has the corresponding key stored/cached locally). The *cumulative distribution* plots are to be interpreted as follows: The x-axis represents the load and the y-axis the percentage of the peer population which has a load less than or equal to this specific (x-axis) load. Thus steeper ascent of the curve represent smaller variation of load among peers, while gradual ascent results from greater variation (poorer load-balance). Two sets of experiments were conducted, once with queries with relative frequency Zipf-distributed, another where all keys were queried exactly the same number of times. Queries were issued at random peers.

The fact that the curve for adaptive replication based search (in Figure 5.5(b)& 5.5(d)) is above the one without replication implies, that the adaptive replication based strategy requires fewer number of messages per peer for search. A steeper slope in Figure 5.5(a) shows that the deviation in the number of queries answered using the replication based strategy is lower than without replication, that is, replication leads to better query-answering load-balance. We also notice that balancing the in-degree based on power-of-two choices (Po2C) leads to improvement in load-balance. The improvements are discernable, but limited. We attribute it to the statistical noise. The huge effect of statistical noise becomes apparent in Figure 5.5(c) for the experiment where all keys are queried the same number of times. In this case, the query-answering load is balanced if no replication is done, since each peer receives queries for its own keys and all keys are queried equally. However, with adaptive replication, as keys are replicated - the effect of statistical noise kicks in, thus in fact leading to load-imbalance.



(a) Queries answered by peers (Zipf distr. queries)

(b) Messages forwarded by peers (Zipf distr. queries)

(c) Queries answered by peers (Same # queries per key)

(d) Messages forwarded by peers (Same # queries per key)

Fig. 5.5. Cumulative distribution of query forwarding and query answering loads at peers

This experiment where keys were queried equally was more to put in context the effect of statistical noise. Under realistic work-loads where access to different keys is often heavily skewed, we'll need the query-adaptive replication as a means to achieve first-order load-balancing. However, a first-order load-balancing is in itself inadequate unless complemented with a second order mechanism to reduce the variance.

5.7 Conclusion and future work

There is a déjà-vu from what we observe from a first objective look at caching based query-load balancing in structured overlays. Initial research in overlay design [163, 140, 154] hoped to achieve good balancing of key-distributions among peers by using uniform distribution. The effect of statistical noise [135] was recognized only later, and had to be fixed using second-order load-balancing mechanisms [32]. Still, one can find in literature that in order to deal with hot-spots, caching is proposed (which is necessary!), and presumed sufficient, which is not the case. In some sense, it is unfortunate that despite dealing with the effect of randomization for storage load balancing, the same effects of randomization for more critical resources - bandwidth and peer's answering capacity under hot-spot conditions, were totally ignored, presuming caching itself will solve the problem. One may speculate several reasons for overlooking such an important issue: (i) Initial work based on simulations could observe the imbalance of key distribution, since it accumulates over time. Bandwidth consumption is however temporary, and if only the average is measured (as has often been reported in most published results), the imbalance goes unnoticed. (ii) It is only recently that some structured overlay implementations have matured enough to be deployed and is dealing with moderate query loads, and hence the effect of imbalance has not been observed. But as the volume of traffic in structured overlays increase, second-order balancing of query-load will become critical, since otherwise it'll cause congestion (and IP layer congestion control mechanisms won't be useful if the overlay systematically causes the congestion at end-nodes) even while other peers would have their resources under-utilized.

In that context, our work rediscovers the ghost of statistical noise. The way to reduce the variance may(not) be straightforward, requiring slight modification of the existing greedy routing mechanism used in most structured overlays. Overlays often maintain multiple routes to a destination for fault-tolerance and during routing many systems send acknowledgement to the previous peer which had forwarded a query. Along with the acknowledgement, peers may piggy-back the load as perceived by themselves in a time-window,[6] and the forwarding peer may use a power-of-two choice to decide the next peer to forward the next query to. Alternatively peers can also measure the delay in the acknowledgement messages to guess not only the load at the next hop peer, but rather the combined effect of load at the next hop peer as well as the underlying physical network for a specific connection e.g., if a particular connection is congested, there will be greater delays. Using such an approach even allows the possibility to implicitly take care of and better exploit the underlying physical network, since it avoids overloading the physical network (which has so far not been studied in the context of peer-to-peer systems), as well as accounts for proximity (proximity route selection [75]).

Such mechanisms are expected to reduce the variance in the query-forwarding load at peers. Coupled with optimal adaptive replication, we speculate that it'll also reduce the variance in query-answering load.

Apart these fundamental outstanding issues to achieve good query-load balancing in structured overlays, there are some practical aspects of incorporating the first-order query-load balancing in a real overlay in a distributed manner which we have overlooked. Particularly, in this chapter we have ignored that the (P-Grid) overlay will already be structurally replicated. Thus, even though the results are valid in principle,

[6] Since bandwidth load does not accumulate assuming flat pricing for internet connection for end users.

performing proportional replication and coalescing the search space will first require to replicate at different structural replicas of the other key-space partition before coalescing larger key-space. Moreover, in the simulations we have replicated a key once for each access/query for the key. In practice, a slower strategy will be necessary in order to reduce data movement every time. Ad-hoc solutions to these practical problems are fairly straightforward, however it'd be interesting to find ways to optimize the amortized cost of querying, caching and cache replacement.

We expect self-organizational principles, based on a proper understanding of the involved interplay of different system properties and design decisions like query and caching costs and rates and routing strategy, to play an important role in achieving practical query-load balancing in structured overlays. In that context (of query-load balancing, which compliments other load-balancing issues studied in this thesis), this chapter is a first step in identifying the design space and the challenges.

6. A self-referential directory

"Most instant message or communication software requires some form of centralized directory for the purposes of establishing a connection between end users in order to associate a static username and identity with an IP number that is likely to change. This change can occur when a user relocates or reconnects to a network with a dynamic IP address. Most Internet-based communication tools track users with a central directory which logs each username and IP number and keeps track of whether users are online or not. Central directories are extremely costly when the user base scales into the millions. By decentralizing this resource-hungry infrastructure, Skype is able to focus all of our resources on developing cutting-edge functionality."
— Skype [160]

6.1 Introduction

Identification provides an essential building block for a large number of services and functionalities in distributed information systems. In its simplest form identification is used to uniquely denote computers on the Internet by IP addresses in combination with the Domain Name System (DNS) as a mapping service between symbolic names and IP addresses. Thus computers can conveniently be referred to by their symbolic names, whereas in the routing process their IP addresses must be used. Higher-level directories, such as X.500/LDAP, consistently map properties to objects which are uniquely identified by their *distinguished name* (DN), i.e., their position in the X.500 tree. Other directories, such as UDDI, map names onto service descriptions and vice versa. These are just a few examples among many others that map sets of attributes onto objects, and that are essential to provide basic functionalities, such as routing of IP packets, searching distributed databases and retrieving certificates from public key authorities to conduct secure e-commerce.

Although the quality and purpose of identification may differ in the various domains, due to varying requirements and levels of abstraction, the basic underlying problem is always the one of binding a set of attributes to an identifier in a unique and deterministic way. Name/directory services such as DNS, X.500, or UDDI are a well-established concept to address this problem in distributed information systems. Usually these services are optimized towards the targeted problem area and differ in the degree of (de-)centralization, security guarantees, descriptive power, and flexibility. However, none of these pre-existing services addresses the specific requirements of peer-to-peer systems.

Peers temporarily depart the system, rejoin with potentially different physical address because of dynamic IP address assignment and/or mobility. It is desirable that the peer retains its identity across sessions so that other peers can re-establish contact with the peer across sessions. The primary functionality we are aiming at here is thus that a peer be able to retain its identity across sessions even if its physical address changes, and other peers can easily re-locate the peer whenever it is available in the network, and prevent other peers from impersonating an existing peer.[1]

Persistent peer identifiers across sessions is a departure from the traditional practice in some other overlays where peers assume a new and random identity in a new session, totally discarding their previous role in the overlay. Using persistent peer identity in the overlay routing level has some interesting uses for even just the overlay infrastructure: (i) In conjunction with structural replication, persisting peer identity can be used to devise a novel and efficient family of overlay maintenance mechanism (as will be elaborated in this chapter). (ii) When a peer rejoins the system, instead of triggering a lot of content movement, it instead needs to only synchronize its content with its online structural replicas. In P-Grid, a pull mechanism is used by peers to get up-to-date, as described later in Chapter 9.

Persistent peer identity across sessions can be exploited by various other peer-to-peer applications. For instance, in a storage system which uses redundancy to guarantee availability and persistence, returning peers can bring back the content stored in previous sessions, thus compensating for loss of redundancy because of (temporary) departure of some other peers. Exploiting the fact that peers have relatively short sessions but a long lifetime in the system can be used in designing efficient redundancy maintenance schemes for collaborative/P2P storage systems, as we will study in Chapter 8.[2]

Peer-to-peer systems are inherently decentralized and thus identification management should be decentralized as well to avoid scalability problems. Peer-to-peer systems are rather dynamic, with nodes frequently joining and leaving the system, and a centralized identification service may easily become a bottleneck. Interestingly, and presumably independently, this observation has been made and exploited by what can be called one of the biggest success stories of legitimate use of peer-to-peer technology, the Skype [160] VoIP network. Additionally, it is desirable not to depend on a third-party infrastructure because if this external service ceases to exist, the peer-to-peer system would no longer be operable. Such a centralized provider can also become a point of censorship or breach of privacy. Thus the peers should be able to manage identification issues themselves.

A system of peers managing their own identity-to-address mapping without any external directory service is self-referential. On the one hand, in order to correctly communicate, peers need to know the correct addresses of other peers. In order to know the correct address of other peers even as peers change their

[1] Our mechanism aims at preventing impersonation, but not Sybil attacks [56], which means that the same peer may have multiple identities in the system.

[2] The TotalRecall [25] storage system was the first system to exploit such returning peers to devise a lazy and efficient maintenance scheme, however, the strategy used there has poor resilience characteristics.

address over time, peers need to communicate with other peers while querying the self-referential directory. Thus there are several forces in action: peers' address changes, update of such address changes and location of the latest information and communication among peers in general (for all other purposes). The repair operations can be proactive, trying to repair irrespective of being necessary or not, or may be adaptive, determining the repair rate based on the overall state of the system.

From a systems design perspective, we were specifically interested in algorithms which will make the system adaptive, so that it can be deployed in a wide range of environments with minimal administrative burden to fine-tune system parameters to judiciously use available resources while providing resilience guarantees. To be more specific, we wanted to devise route maintenance strategies which will not waste resources to maintain routes when there is no need to, while gracefully adapt the rate of repairs as and when the membership dynamics increases in the system. Depending on the rate at which changes occur and how the latest changes are propagated among affected peers (repair of stale information), such a self-referential system under continuous changes and maintenance operation is expected to operate in a dynamic equilibrium (corresponding to a specific rate of changes) or collapse if the repair rate is too low to compensate for the changes.

There are a multitude of design choices in engineering such a self-referential directory. In order to understand the implications of such design choices as well as quantitatively compare different options, we study the system (modeled as a probabilistic system) using a Markov model, where the state transitions are triggered either when peers change their addresses, or update the locally cached addresses of other peers (depending on the route maintenance policy), and account also for peers being either off-line or online. We look into the system's steady-state (dynamic equilibrium) to evaluate system performance, particularly looking at the overheads of maintaining the overlay as a metric to compare different maintenance schemes. We propose interesting metrics like the operational zone based on contour maps to also evaluate the dynamics that the system can tolerate given a budget (network usage) for a chosen maintenance scheme.

While the presented system design as well as analytical study are restricted to the P-Grid overlay network and only some specific overlay maintenance schemes, the ideas and analysis approach in themselves are generally applicable to study overlays under churn. For instance, the dynamic equilibrium behavior of the Chord overlay network under continuous churn while using its self-stabilization algorithms [163] has subsequently been studied by others [101, 102].

6.1.1 A separation of concern from the underlying physical network

Overlay networks use a logical identification of the peers, realizing a logical independence from the physical address in the underlying physical network. For routing this logical identification is mapped onto an IP address in the routing tables. Since IP addresses are scarce most peers will have dynamic IP addresses that may change over time. This problem would be solved if Mobile IP [130] or IPv6 [155] were in place already and available at a large scale, because they take into account mobility (dynamism) and offer a much larger address space. However, this requires considerable changes in the basic networking infrastructure of the complete Internet and the ground reality is that such an infrastructure is yet to be widely deployed. Our approach of using a self-referential directory not only bridges this gap but also disentangles the overlay infrastructure from the peculiarities of the underlying physical network.

6.1.2 What are some of the other overlays doing?

In Chord [163], peers that (re-)join the overlay with a new IP address adopt a new identity and introduce themselves into the routing infrastructure like a completely new node. This is partly due to the fact that in Chord the logical identifier depends on the IP address. To repair faulty entries in routing tables resulting from node departures, the approach in [106] devises a periodic maintenance protocol. The execution of the periodic stabilization protocol is independent of changes to the network and the adaptation resulting from repairs may compromise structural properties of the routing infrastructure, which may have been established in order to address some other properties, like dealing with non-uniform workloads [138]. DKS(N,k,f) [16], a generalization of the Chord model, proposes a correction-on-use protocol to maintain routing tables. The authors show that their protocol is self-stabilizing and more efficient than the Chord maintenance protocol. In their approach peers can maintain logical identifiers with changing IP addresses, but routing tables need to be reorganized at all depths with the occurrence of every update.

In Pastry [154] nodes have an independent logical ID, and upon re-entering the overlay peers may enter their new ID-to-IP binding into the routing tables of peers, encountered when executing the node join protocol. However, as these peers are typically different from those already storing such bindings, stale mappings will be encountered by other peers during query routing. These stale entries are replaced by new routing entries [112], irrespective of whether the peer identified by the stale entry has rejoined with a new IP address or not.

6.1.3 Motivation for a new approach

We see the management of dynamic IP addresses in overlay networks as an instance of the more general problem of identification. In this chapter we will present our decentralized, self-maintaining approach to identification and prove its applicability and validity by applying it to the basic problem of changing IP addresses. In contrast to the approaches of Chord, Pastry and DKS, our strategy which we use in P-Grid can track changes in the mapping. This is important as soon as information on the characteristics of specific peers is exploited for routing purposes or needed by applications which exploit specific information of individual peers, such as their trustworthiness, quality of service or locality. Thus our approach is in particular relevant for applications such as e-commerce [50], trust management [9], collaborative storage networks,[3] social networks (clusters) such as semantic overlay networks [3] and potentially for mobility management in ad-hoc networks. If information about peers is gathered from earlier interactions and the choice of routing table entries is made dependent on such properties, changes to the structure of the overlay network due to modification of routing tables should be avoided if possible (unless triggered by the application). This is in particular true for changes resulting from a peer's physical address, which may be completely independent of a peer's other properties. The approaches of changing the logical IDs or restructuring routing tables as a result of changes to the peers' physical IDs as used in most other overlays would incur a loss of information. Thus our approach makes the overlay network logically independent of the underlying physical network. It is worth to mention that this functional advantage comes at no specific additional cost. All approaches (including the one we introduce here) incur $O(\log^2 n)$ message costs for maintenance.

File sharing applications brought the potential of a peer-to-peer paradigm to the forefront. In file sharing applications user response time is a major issue and identification is viewed to be only of subordinate importance. However, this is already changing, since reputation, data authenticity, and fair use of resources have become major issues in P2P systems and require identification as a service to address them. Also, quickly finding the peers with which some trust relationship has been established (say about quality of information) reduces response time and provides the appropriate information - such is the case in clusters of peers with similar interests, e.g., semantic overlay networks [3]. If peers have dynamic network addresses, this again requires an identification service as described here. As mentioned earlier, collaborative storage systems can also exploit the persistence of peer identity across sessions to devise better storage redundancy maintenance schemes as will be described in Chapter 8.

6.1.4 Problem statement and overview of the approach

To support dynamic IP addresses as an application of identification, it is necessary to address the following problems: (1) How can universally unique identifiers be mapped onto physical addresses in a secure, decentralized and efficient way, and be maintained securely by the owner? (2) With the possibility of changes of the mapping, i.e., the physical addresses, how can a peer detect whether it is still talking with the intended entity? This means that, (a) if peer p_1 goes offline and a different peer p_2 gets associated with p_1's old IP address, the other peers in the system must be able to detect this change and react accordingly, and (b) if a peer goes online again with a new IP address, the other peers must be able to detect this, re-locate the peer and verify its identity and update their routing tables accordingly.

We originally proposed the self-referential directory in order to maintain the P-Grid overlay in presence of logical mobility of the participating peers, e.g., to keep track of changing IP addresses. However, the

[3] Some P2P storage systems (see TotalRecall [25] as well as Chapter 8) exploit the fact that most peers have short session time but long life-time in the system, and thus use lazy maintenance of redundantly stored content. In such a system even if the peer changes its physical address, it is nonetheless desirable to be able to locate the peer at its new address.

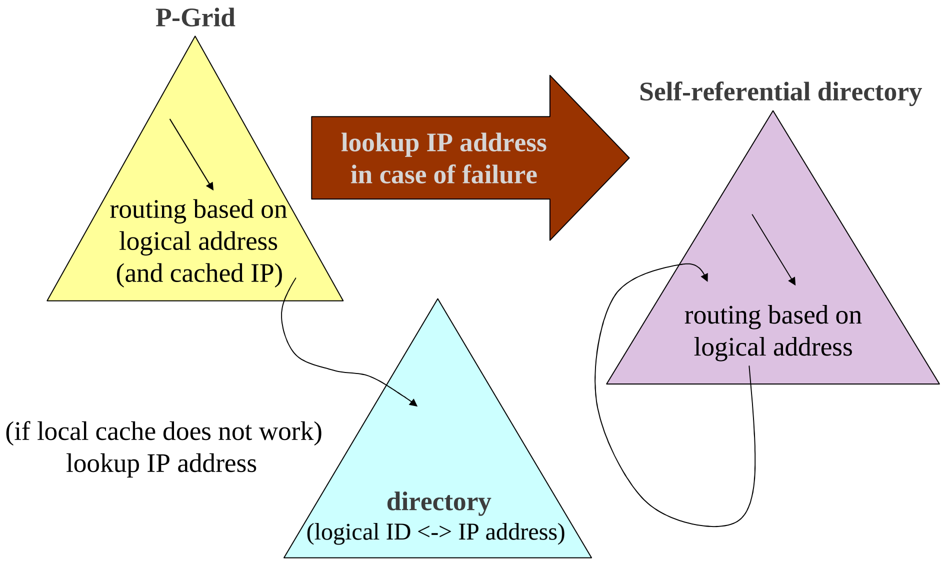


Fig. 6.1. Use of a self-referential directory instead of an external directory service

same approach may be used to maintain any other information about the peers as well and in principle, in any structured overlay. We explain here the basic idea of how ID-to-IP mappings are managed.

Peers use their public key as their logical identifier (ID). Mapping of information (like ID-to-IP mapping) about a peer is stored in the overlay corresponding to a key generated using the peer's ID. e.g., the ID itself can be used as the key. This information is redundantly stored in the overlay on a certain number of peers, e.g., at structural replicas of P-Grid. Any peer trying to look up the ID-to-IP mapping will issue a query corresponding to the key generated using the peer's ID. Similarly, when a peer needs to reinsert a new value for the mapping, it signs this latest value with its private key, and inserts it corresponding to the key. Use of public key cryptography and an external directory service could be used in order to maintain and retrieve such mappings corresponding to any ID. However, since a structured overlay network itself is a directory service, we choose to use the overlay as its own directory to manage the ID-to-IP mapping of participating peers. Following this simple strategy a self-referential directory is realized which is resilient against impersonation attacks. However, other attacks, particularly denial-of-service (D)DoS attacks are possible on such a directory service, and to that end, we consider a collaborative environment. Some preliminary security assessment looking into the possible attacks including (D)DoS has been done [50].

A self-referential directory for self-healing routing. At an abstract level, in a structured overlay the logical IDs are used for querying and routing, but still peers need to know and use physical addresses for the actual communication.

Contacting a peer fails if the peer has either changed its network address or because it is offline. The requester can now either assume that the peer is offline (unreachable) and give up or query a directory service maintaining ID-to-IP mappings to determine the peer's latest network address. In using a self-referential directory, what happens is that when an unusable routing entry is encountered, in order to retrieve the peer's latest IP address, a new query is issued in the overlay itself. Thus queries are issued recursively. Upon completion of such a child query a routing entry may be corrected based on the latest retrieved ID-to-IP mapping, and the parent query may be continued. That is why we call this routing process self-healing. Since there are redundant routing entries at each peer, it is also not necessary to always issue a recursive child query when an unusable routing entry is encountered. Depending on how aggressively the recursive queries are issued, we have a whole family of route maintenance strategies using the basic principle of self-healing routing. The idea is illustrated in Figure 6.1.

If and when contacting a peer from the routing table succeeds, its public key can be used to determine whether the contacted peer is really the one, whether a different peer reuses the address, or a malicious peer tries an impersonation attack.

There is an apparent hen-egg problem because of the recurrent nature of self-reference, as we use the overlay that depends on using the mappings for effective routing also for storing and maintaining the mappings. The main focus of this work is to demonstrate that despite this recursive dependency, it is in fact possible to realize a self-contained service which is completely decentralized and self-maintaining. We evaluate the performance of the overlay (and the maintenance scheme) under continuous churn and self-healing.

The rest of the chapter is organized as follows. In Section 6.2 we describe the protocols used by peers participating in a self-referential directory, particularly looking into how peers (re-)join the system, and how they operate. We provide an example of a self-referential directory realized using the P-Grid overlay in Section 6.3, where we also give a concrete example of self-healing routing. We furnish details of the self-healing routing which is algorithmically a slight variation of the greedy routing algorithm in Section 6.4. We analyze the two extreme variants of the self-healing family of routing algorithms, where the recursive self-healing queries are triggered as lazily or aggressively as possible in Section 6.5. We validate the analysis and evaluate the performance of self-healing routing with simulations and report our results in Section 6.6. We take a detailed look of various related works in Section 6.7 before concluding in Section 6.8.

Before proceeding further, a brief note on the security implications of our approach is due. Note that for a peer joining the network for the first time, there is no notion of an existing identity, and it is accepted as a new user, and its public key stored at the responsible peers. When it comes back, it signs its latest mapping with the corresponding private key. This ensures prevention against impersonation attacks. A peer may however create multiple identities (Sybil attack [56]). In an internet setting it is hard to prevent such an attack. Some preliminary study of using the self-referential directory for realizing a PGP-like decentralized public key infrastructure has been done in [50]. With Sybil attacks, the malicious users can in fact be the same, and hence collaborating. Effective prevention of Sybil attacks, as well as other known or even unknown sophisticated attacks in a decentralized setting thus clearly stays an open challenge, and only some approaches based on centralized certification authorities exist so far [33]. The prevention of impersonation that our directory service provides guarantees that no other peer can impersonate an existing peer. Avoiding impersonation has at least the advantage that if trust has been built in a given identity or resources are assigned to a specific identity, then it can be safely maintained and not be abused by others. Other attacks are out of the scope of this work.

6.2 Self-referential directory service protocols

This section defines the algorithm and protocol for maintaining ID-to-IP mappings in P-Grid. Generalizing this specific example to other mappings is implicit and straight-forward.

As is common in various peer-to-peer systems, each peer generates a private/public key pair D_p/E_p locally once in the bootstrap phase by applying a cryptographically secure hash function using random inputs including, for example, its current IP address $addr_p$. Each peer p is uniquely identified by its public key E_p.

In the following we use the notation $D_p(x)$ and $E_p(x)$, where $D_p(E_p(x)) = E_p(D_p(x)) = x$, to denote the application of the private and public keys in an asymmetric encryption scheme.

In P-Grid routing tables and the index hold only these identifiers. Each peer p additionally has a cache of mappings $(E_i, addr_i, TS_i)$ for peers that it already knows, including at least its routing table entries. TS_i denotes a timestamp which must be included to prevent replay attacks, i.e., the recording of transmitted information by a malicious party and replaying it at a later time (the timestamp guarantees the freshness of messages). When a peer joins or rejoins the network, it inserts a signed version of the tuple $(E_i, addr_i, TS_i)$ in the overlays. This tuple is stored in the network at replicas (to recall from Chapter 3, $\Re(\kappa)$ represents the set of peers replicating the object corresponding to key κ) corresponding to a key generated from E_i, that is at $\Re(\mathrm{f}(E_i))$, where the function to generate the key f() can be any globally defined one-to-one mapping function from the ID-space to the key-space (binary string). Inserts and updates within the replicas is done using a push/pull-based gossiping communication primitive [51] which is explained in Chapter 9.

Peer (re-)joining the system.

1. p starts up and checks whether its $addr_p$ has changed. If not, the system already has the latest address. Otherwise, the following steps are taken.

2. When peer p wants to (re-)join it sends an insert/update message $(E_p, addr_p, TS_p, D_p(E_p, addr_p, TS_p))$ to the P-Grid, i.e., a new mapping and a signature for this mapping. Multiple redundant insert messages can be made in order to ascertain that multiple peers get the insert messages to form a probabilistic quorum. Using such an approach, some simple faults and denial-of-service attacks can be avoided [50].
3. Upon receiving the update request, the responsible peer(s) $r_i \in \Re(\mathrm{f}(E_p))$ check the signature. Thus only p can update its mapping. The new time-stamp is checked in order to prevent replay attacks. An error message is returned if there is any discrepancy.
4. Each peer from $\Re(\mathrm{f}(E_p))$ who gets the update request directly from p also initiates an unique gossip to synchronize the replica subset using the aforementioned push/pull mechanism.
   Some simple mechanisms of security can be augmented to this scheme. In order to make sure that the update is correctly registered in presence of churn as well as some free-riding replicas who do not initiate gossips, the peer p may actually also try to send the update message to several of the replicas from $\Re(\mathrm{f}(E_p))$, thus achieving a probabilistic quorum.[4]

Operation phase.
In the operation phase p is up and running, has registered an up-to-date mapping $(E_p, addr_p, TS_p)$, and is ready to process queries (issue, answer or forward them) and update requests.

1. p receives a request $Query(\kappa, p)$ from some peer q.
2. In case p can answer $Query(\kappa, p)$ locally, the result is returned to q. Otherwise p decides which peers p_f from its routing table it can forward the query according to P-Grid's routing strategy. Then it checks its routing table and retrieves $(E_{p_f}, addr_{p_f}, TS_{p_f})$ which had been cached since the last successful communication with p_f.
3. p generates a random number ρ, contacts p_f and sends $E_{p_f}(\rho)$. As an answer p_f must send $(D_{p_f}(E_{p_f}(\rho)))$ and p can check whether $D_{p_f}(E_{p_f}(\rho)) = \rho$. If yes, p_f is correctly identified, i.e., p really talks to the peer it intends to, and $Query$ is forwarded to p_f.
4. If not, then p_f has a new IP address (the case that somebody tries to impersonate p_f is covered implicitly by the signature check above), and p issues a new query to P-Grid to retrieve the current $addr_{p_f}$ using $\mathrm{f}(E_{p_f})$ as the key. Since this query is a child query created while processing the original one, and in fact can in turn lead to more children queries, we call this mechanism as *recursive-query*,[5] and provide the details of the recursive-query in Algorithm 6. Moreover, a maximum number of recursions can be defined in order to restrict long or infinitely running recursions, though simulations for small-sized networks did not need such a hard-coded restriction for depth of recursion and terminated successfully rather fast.

[4] Only very basic assessment of the security implications has been done [50].

[5] Note that this notion of recursive query is completely different from the similar terminology (Section 3.4.4) used to describe the actual way to process isolated query in an iterative or recursive manner.

5. In order to avoid chance errors because of replica inconsistency, p can use replies from multiple distinct peers from $\Re(\mathrm{f}(E_{p_f}))$ and choose the latest time-stamped information.
6. Now p can proceed with step 3. In case this is successful, p updates its local cache with $(E_{p_f}, addr_{p_f}, TS_{p_f})$. Thus using the P-Grid itself as a directory service, the recursive-querying mechanism corrects stale routing entries arising from changes in peers' physical address, i.e., the overlay network self-heals while processing queries.

6.3 Processing queries using self-healing routing: An example

Figure 6.2 shows a typical snapshot of a P-Grid network, and how it is used as a self-referential directory.

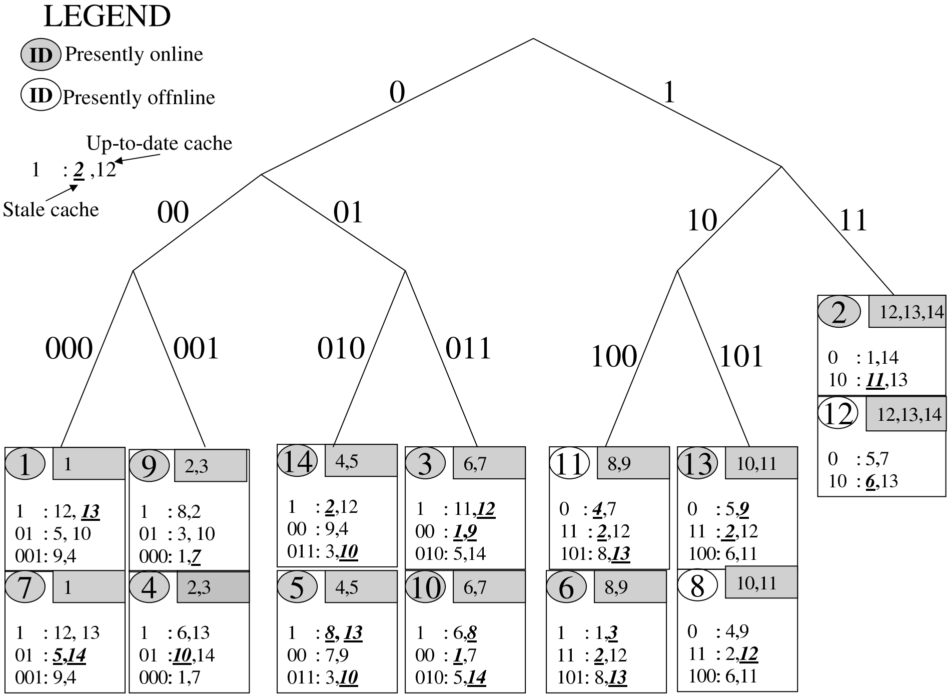


Fig. 6.2. A self-referential directory realized using P-Grid: Before $Query(01\cdots)$ at p_7

Peer p_i is denoted by i inside an oval. Online peers are indicated by shaded ovals, offline peers by unshaded ovals. Peers at the same leaf-node are replicas. For example, p_1 and p_7 are both responsible for keys with prefix 000, i.e. $\Re(000) = \{p_1, p_7\}$. Without loss of generality we assume that Id_p has a length of 4 bits in this example. Thus, for instance, p_7 holds the public key and latest physical address mapping about p_1 (updated by p_1) because p_7 is responsible for the paths 0000 and 0001. The shaded rectangle in the upper-right corner of each peer shows the peer IDs that a peer is responsible for, i.e., whose public key and physical address mapping it manages. There exists no dependency between the peer identity ($id_{p_7} = 0111$) and the key-space partition path it is associated with ($\pi(p_7) = 000$). In its routing table p_7 stores references for paths starting with 1, 01 and 001, so that queries with these prefixes can be forwarded closer to the peers holding the searched information. The cached physical addresses of these references may be up-to-date (for example, p_{13}'s) or be stale (denoted by underlining, for example, $\underline{p_5}$).

A peer p_q decides that it has failed to contact a peer p_s, if one of the following happens: (1) No peer is available at the cached address (trivial case). (2) The contacted peer fails in the authentication as described in the previous section on the operation phase of the algorithm (step 3). p_q will use p_s's public key to verify p_s's identity. Since only p_s knows its private key which must have been used for the signature, it is the only peer that can pass the identity test. In either of the above two cases an up-to-date mapping must be obtained by querying the P-Grid. We have investigated two querying strategies:

Isolated-Query (with greedy routing): Upon receiving a query a peer checks whether it can answer the request. If not, it forwards the query to at least one of the peers in its routing table according to P-Grid's greedy routing algorithm 1. If none of these peers can be contacted, the query is abandoned and fails.

Recursive-Query (with a self-healing variant of the greedy routing): If a peer fails to contact peers in its routing table, it initiates a new query to discover the latest ID-to-IP mapping of any of those peers. If this is successful it forwards the query.

Note that the $Query$ operation in Algorithm 6 is an extension of the Retrieve Algorithm 1 which was introduced in Chapter 3. Note also that we are overloading the notation $Query(x, p)$ to also mean $Query(\pi(x), p)$ where x is a peer identifier, and using only $Query(\kappa)$ when the context as to which peer the query for any key κ is being processed is clear. Furthermore, the actual query may be for a normal key corresponding to some data or for the ID-to-IP mapping, but here we mask the application level semantics, and only focus on the routing.

The protocol in Section 6.2 will work as follows: While the P-Grid is in the state as shown in Figure 6.2, assume that p_7 receives a query $Query(01\cdots)$. p_7 fails to forward the query to either of p_5 or p_{14} since their cache entries are stale. Here, the *Isolated-Query* algorithm fails immediately.

In contrast, the *Recursive-Query* algorithm would try to discover the latest addresses for the stale entries. p_7 initiates *Recursive-Query*(p_5), using $Query(0101)$, which needs to be forwarded to either p_5 or p_{14}. This fails again. p_7 then initiates *Recursive-Query*(p_{14}), i.e., $Query(1110)$, which needs to be forwarded to p_{12} and (or) p_{13}. p_{12} is off-line, so irrespective of the cache being stale or up-to-date, the query cannot be forwarded to p_{12}. p_{13} is online, and the cached physical address of p_{13} at p_7 is up-to-date, so the query is forwarded to p_{13}. p_{13} needs to forward $Query(p_{14})$ to either p_2 or p_{12}. Forwarding to p_{12} fails and so does the attempt to forward the query to p_2 because p_{13}'s cache entry for p_2 is stale. Thus p_{13} initiates another sub-query, *Recursive-Query*(p_2), i.e., $Query(0010)$. Additionally, it may initiate *Recursive-Query*(p_{12}). p_{13} sends $Query(p_2)$ to p_5 which forwards it to p_7 and/or p_9. Let us assume p_9 replies. Thus p_{13} learns p_2's address and updates its cache. p_{13} also starts processing and forwards the parent query *Recursive-Query*(p_{14}) to p_2. p_2 provides p_{14}'s up-to-date address, and p_7 updates its cache.

Having learned p_{14}'s current physical address, p_7 now forwards the original query $Query(01\cdots)$ to p_{14}. This does not only satisfy the original query but p_7 also has the opportunity to learn and update physical addresses p_{14} knows and p_7 needs, for example, p_5's latest physical address (we assume that peers synchronize their routing tables during communication since this does not incur any overhead). In the end, the query $Query(01\cdots)$ is answered successfully and additionally p_7 gets to know the up-to-date physical addresses of p_{14} and possibly of p_5. Furthermore, due to child queries, p_{13} updates its cached address for p_2. Figure 6.3 shows the final state of the P-Grid with several caches updated after the the completion of $Query(01\cdots)$ at p_7.

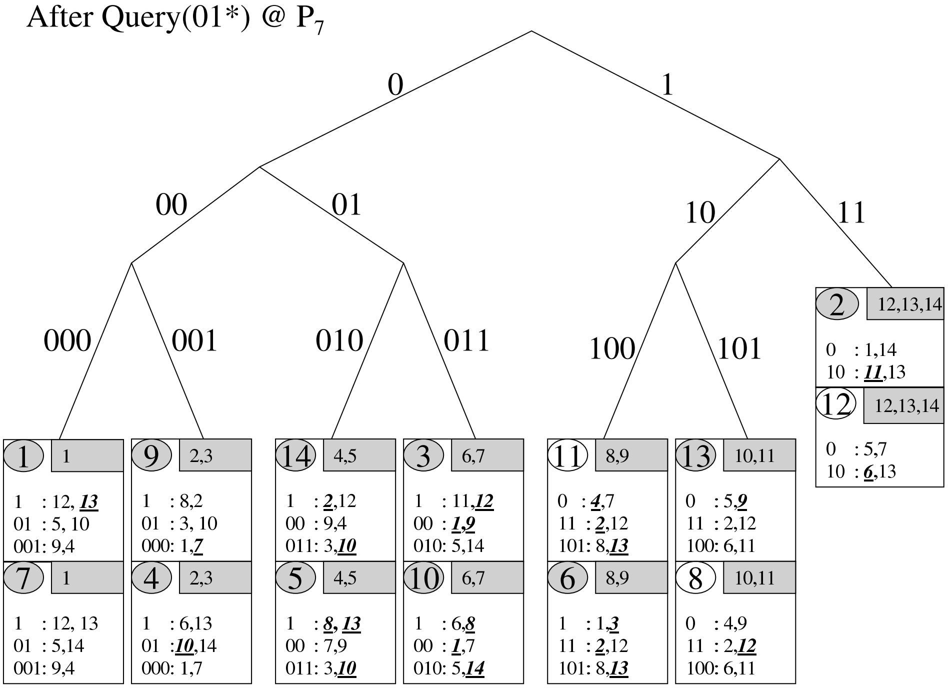


Fig. 6.3. P-Grid after $Query(01\cdots)$ at p_7

We have not explicitly mentioned concurrency issues in the example above because those are either addressed by the networking layer or do not cause problems since we support only lazy consistency. For example, if the IP addresses change during authentication, the networking layer would drop the connection and the authentication would start anew without causing security or concurrency problems. If the IP address of a peer to be contacted would change after retrieving a mapping, i.e., a query would return a "stale" entry, this would be recognized, because the authentication would fail upon contacting the peer present at the retrieved address. Since only the owner can update a mapping, concurrency control is not needed here and if replicas have not been updated correctly when being queried, this would also be recognized upon contacting the peer and can be accounted for by re-issuing the same query. Additionally, we can assume that IP addresses do not change at a high frequency (normally once a peer goes online it keeps the same address for at least some hours in typical ISPs' DHCP setups). However, other "hidden costs," for example, updates to the mappings and rectifying stale cache entries (self-healing) need to be taken into account as discussed in the following sections.

6.4 Self-healing routing algorithm for a query (search)

Algorithm 6 shows the recursive query (search) algorithm in pseudo-code. To recall notations introduced in Chapter 3, $\Re(\kappa)$ denotes the set of peers (replicas) which has the result to a query for path κ. If a peer p receives a query $Query(\kappa)$ and $p \notin \Re(\kappa)$, then it tries to route (forward) the query to peers in its routing table according to P-Grid's routing strategy using the routing entries in $\rho(p)$.

Algorithm 6 Query at peer p for an object corresponding to key κ: $Query(\kappa, p)$

```
 1: if $\pi(p) \subseteq \kappa$ i.e., $p \in \Re(\kappa)$ then
 2:   reply to query; {$p$ has the requested information}
 3: else
 4:   $\rho_{ch} = \{q \in \rho(p,l) \wedge q$ is contactable (using ping) at cached address for $l$ s.t. $\pi(p,l) = \overline{\pi(\kappa,l)}\}$
 5:   if ($\rho_{ch} = \emptyset$ and lazy repair strategy) or eager repair strategy then
 6:     {Only for the self-healing routing used by recursive queries, since they try to repair routes.}
 7:     for all $p'' \in \rho(p,l) \wedge p'' \notin \rho_{ch}$ do
 8:       $Query(Id_{p''}, p)$; {The actual semantics of a child query is somewhat different depending on the object being searched,
          but here we are interested in the involved routing process only.}
 9:     end for
10:     Re-determine $\rho_{ch}$ as in step 4; {New entries expected because of repairs.}
11:   end if
12:   if $\rho_{ch} \neq \emptyset$ then
13:     $Query(\kappa, p')$ for some $p' \in \rho_{ch}$; {Query for $\kappa$ forwarded to $p'$.}
14:   else
15:     return failure;
16:   end if
17: end if
```

A family of route maintenance mechanisms can be realized by triggering the self-healing recursive queries based on the number of unusable routing references (out of r possible) for any given level at a peer. For the sake of simplicity, we consider the two extremes, where a recursive query is triggered whenever an unusable entry is encountered. We call this the eager repair or Correction on Use (CoU). The other extreme is to trigger repairs only after all of the r references become usable, which we call - lazy repair or Correction on Failure (CoF). These two recursive variants may trigger child queries at any stage in the routing process until they succeed. The recursive strategies affect the rectification of stale caches at various peers, thereby "self-healing" the overall routing network. To avoid cyclic recursions the queries bear unique identifiers and recursion does not occur when it would apply to a reference that is under repair in a parent query. Additionally, this helps to prevent concurrent repair requests for the same stale entry at one peer.

Deadlock situations, where none of the recursive queries terminates successfully and all entries at one level are stale, may occur. But experiments show that their influence on performance decreases with an increasing network size.

The pseudo-code in Algorithm 6 summarizes different variants of routing in the overlay including the non-recursive isolated query (essentially Algorithm 1), as well as the lazy and eager recursive query strategies as introduced above.

The non-recursive variant (isolated query) uses the standard greedy routing as used by most structured overlays and thus provides a result for a query if at least one routing entry is up-to-date for each routing step and the corresponding peer(s) are online. Thus it only works, if the routing tables are sufficiently redundant.

6.5 Analysis of the algorithms

6.5.1 Models for analyzing the overlay under churn

Prior and contemporary to our work, the models used to study overlays under churn included static resilience [75] and half-life [106]. However a more detailed study of the continuous and combined effect of churn and maintenance operations which depends specifically on the maintenance strategy, provides a more accurate performance/resilience versus cost metric of the dynamic system. To that end, we propose to study the system's time evolution. This is a general analysis framework adapted from cybernetics [85] and is used not only in studying overlays under churn, but we will reuse the same basic framework in Chapter 8 for studying P2P storage systems as well. In Section 8.4, we will recapitulate the models for analyzing P2P (storage) systems under churn.

Static resilience: The basic question answered using a static resilience analysis is: *If a certain fraction of the peers has left the network and a corresponding fraction of information is unavailable, what is the performance of the system? For example, in an overlay, what fraction of the queries can still be routed?* This model does not take into account the role of the maintenance operations, but provides a good idea about the system's resilience even without/before the maintenance operations.

Δ-life: This model looks into the lower bound of the cost that is needed to completely repair the system essentially answering the following: *As the network membership changes over time, such that only Δ (say* 0.5*) fraction of the original peers remain in the system, what is the minimal number of repairs that are required to restore a fully consistent state (get all information up-to-date)?* This model too does not look into the performance of any given maintenance scheme.

Time-evolution: The Markov model [85] is traditionally used to study the dynamics of large probabilistic systems. Any change (event) in the system is considered as a transition from one possible system state to another. We adapt this well established methodology in the context of studying the dynamics of P2P storage systems under churn by modeling the system as a Markov process and looking into the time evolution of the probability density function of all the possible states the system can be in, and hence to see if this distribution function converges to a *steady state/dynamic equilibrium* in the long run. If such a steady state exists, then *it determines the operational state of the system under the given churn and adopted maintenance strategy, which in turn is necessary to determine the performance vs. operational cost trade-offs in the system.*

6.5.2 Analysis of an isolated search/query (static resilience of the overlay)

In the analysis of the algorithms, we use the following notation: $\mathcal{P}_{on}$ denotes the probability of peers being online; $\mathcal{P}_{dyn}$ defines the probability that a randomly selected entry of a routing table is stale; μ is the probability that an isolated attempt to contact any particular peer p_i by peer p_j using its local cache information fails; ϵ_h denotes the failure probability of forwarding a query to any other peer specialized for the other half of the search-subtree; ϵ defines the failure probability of a query; $|\Pi|$ is the number of leaves; r is the number of references for the other half of the subtree in P-Grid routing tables for each depth. It is important to note, that we can assume these r references to be independent due to the randomized construction process of P-Grid. A (A_ϵ) denotes the expected number of attempts (message exchanges) required for a query (along with the achieved failure rate).

We first analyze the effect on P-Grid searches of peers going off-line and then rejoining the P-Grid with a possibly different physical address. When a peer p_q needs to forward a query $Query(p_i)$, it may fail to do so, because all the peers in $\Re_{p_i,q}$, to which the query may be forwarded, are off-line or their cached physical addresses are stale (or both). If the overall offline probability of peers is $1 - \mathcal{P}_{on}$, and the probability that a cache entry at p_q is stale is $\mathcal{P}_{dyn}$, then the probability that an isolated attempt at P_q to reach a particular peer in $\Re_{p_i,q}$ is successful equals $1 - \mu = \mathcal{P}_{on}(1 - \mathcal{P}_{dyn})$. Likewise, the failure probability of an isolated attempt to forward a query equals $\mu = 1 - \mathcal{P}_{on}(1 - \mathcal{P}_{dyn})$.

Thus μ represents the coupled probability that a peer is off-line and/or the physical address associated with any peer p_s cached at p_q has changed. Consequently, when attempts are made to contact r random peers from the references $\Re_{p_i,q}$ at p_q, the probability that all r attempts fail is μ^r. So, given a per-hop failure tolerance ϵ_h, we need a minimum of r references to which a search may be forwarded, such that $\mu^r < \epsilon_h$. Thus we need at least $A_{\epsilon_h} = \lceil \frac{\log \epsilon_h}{\log \mu} \rceil$ references for the other half of the P-Grid subtree at any depth to achieve a given ϵ_h (and vice versa).

With a ϵ_{h_i} failure probability for query routing at hop i, the probability of successful routing to a desired leaf node is $\prod_{i=1}^{H}(1 - \epsilon_{h_i})$ where H is the expected number of hops to reach the particular leaf node in question. If there are at least A_{ϵ_h} references available at any hop then $\epsilon_h \geq \epsilon_{h_i} \forall i$, and thus ϵ_h determines the worst per-hop failure probability. We use this ϵ_h for all hops, thus determining a worst case average performance in the remaining analysis. We thus obtain the effort for one hop of the non-recursive version of the query as $A_h^{iso} = A_{\epsilon_h}$. The expected total cost to process a query in a balanced P-Grid is then $A_{iso} = \frac{\log_2 |\Pi|}{2} A_h^{iso}$.

If ϵ_h is achievable at every hop (enough references available) then the success probability is $1 - \epsilon = (1 - \epsilon_h)^H$ where H is the number of times the query needs to be forwarded to reach the leaf node. Thus, the expected value of the achievable success probability is $1 - \epsilon = E_H[(1 - \epsilon_h)^H]$. For a general P-Grid, the distribution of H is not known and thus the expectation difficult to evaluate, but for a balanced P-Grid, H is a binomial random variable of size $\log_2 |\Pi|$ and parameter 0.5. Hence, $1 - \epsilon = (1 - \frac{\epsilon_h}{2})^{\log_2 |\Pi|}$. This analysis gives the static resilience of the network - given a certain state of correctness of the routing tables - the probability of resolving queries successfully.

6.5.3 Recursive queries and dynamic equilibrium

While cached entries continuously get stale owing to network dynamics, they trigger recursive queries in order to update the stale mappings. Hence the recursive version of querying in P-Grid has an inherent self-healing property. With few stale mappings, there is hardly any deterioration in answering the queries, but as the stale entries accumulate over time, they lead to more frequent recursions. Thus it is expected that the system will reach a dynamic equilibrium, such that the rate of changes will equal the rate at which self-maintenance is done due to recursions. If the rate of changes in the system is very high, then the system's self-maintenance will be unable to catch up with the changes, and the system breaks down. In this section, we analyze if a dynamic equilibrium for a given rate of changes is achievable, thus determining the operational zone with respect to the rate of change, and if such an equilibrium is achievable, we evaluate the system behavior (dynamic resilience) at the equilibrium.

In the analysis and the results we quantify the rate of change considering two kinds of events in the network. With probability r_{up} an event consists of peers independently updating their address. With probability $1 - r_{up}$ an event consists of peers issuing queries. Since updates necessarily imply the execution of one query to locate a node to which the update is going to be stored, we assume that $r_{up} \leq 0.5$.

Eager recursion. In the eager recursion, all references are checked and the algorithm tries to rectify all stale cache entries immediately. The effort for a single hop is $A_h = A_h^{iso} + r\mu A_{rec}$ since an expected $r\mu$ recursions will be initiated at each hop, even if the original query is forwarded, where $\mu = 1 - \mathcal{P}_{on}(1 - \mathcal{P}_{dyn})$. The effort for a recursive query is $A_{rec} = \frac{\log_2 |\Pi|}{2} A_h$. Thus the expected number of recursions $N_{rec} = \frac{A_{rec}}{A_h^{iso}} = \frac{1}{1 - \frac{\log_2 |\Pi|}{2} r\mu}$.

A single hop fails when none of the references can be contacted, either because the recursive query fails or the concerned peer is offline. If ϵ_h is the probability of failure of a single hop (forwarding) in the querying process, then $1 - \epsilon_r = (1 - \frac{\epsilon_h}{2})^{\log_2 |\Pi|}$ (as derived in Section 6.5.2). The probability that a single hop fails equals the probability that recursions are initiated and none of the r children responses are usable, either because the recursions themselves fail, or even if they do not fail, the concerned routing reference is offline. Thus $\epsilon_h = \mathcal{P}_{rec}(1 - (1 - \epsilon_r)\mathcal{P}_{on})^r$.

The dynamic equilibrium equation then is $(1 - r_{up})(N_{rec} - 1)\mathcal{P}_{on}(1 - \epsilon_r) = r_{up}(1 - \mathcal{P}_{dyn})r \log_2 |\Pi|$. The left hand side is the rate at which successful repairs of stale routing table entries occur, due to the $N_{rec} - 1$ additional recursive queries. Repairs can only lead to a successful update, if the peer to be repaired is online, therefore an additional factor of $\mathcal{P}_{on}$. This reflects the underlying assumption that queries and thus repairs occur at a substantially higher rate than the peers are switching between offline and online state. The right hand side is the rate at which routing table entries turn stale due to changes.

We solve the above equations numerically for $\mathcal{P}_{dyn}$, ϵ_r and N_{rec} to determine the system performance at the dynamic equilibrium given the system parameters $\mathcal{P}_{on}$, r_{up}, r and n. For the case $\mathcal{P}_{on} = 1$ it is also relatively easy to see from the equations that $N_{rec} \leq 1 + \frac{r_{up}}{1-r_{up}} r \log_2 |\Pi|$. This shows that eager recursion exhibits also excellent scalability with $|\Pi|$.

Lazy Recursion. For lazy recursion we need to analyze the states of routing tables in more detail. There are r routing table entries at each peer at each depth. Let S_i denote the probability that i out of r routing table entries at a given depth are stale. In the following we will also say that a routing table at a given depth is in state S_i.

When using lazy recursion, queries are triggered if a peer cannot use any of its corresponding routing tables to forward a query. This happens either because the routing entries are stale and the latest ID-to-IP mappings have to be found (using recursive queries) or because the concerned peers are offline. We assume that in this case the peer issues parallel queries for all r references in order to minimize response time. Recursive queries occur with a probability $\mathcal{P}_{rec} = \sum_{i=0}^{r} S_{r-i}(1 - \mathcal{P}_{on})^i$ where $\mathcal{P}_{on}$ is the probability that any particular peer is online.

The effort to forward any query (one hop) is $A_h = A_h^{iso} + r\mathcal{P}_{rec}A_{rec}$, where A_{rec} is the total effort in terms of number of messages for a newly issued recursive query. From the properties of P-Grid, any query requires $\frac{\log_2 |\Pi|}{2}$ forwarding steps on an average to successfully answer the query. Hence, $A_{rec} = \frac{\log_2 |\Pi|}{2} A_h$. Solving these recursive equations, we obtain the total number of queries including the original and children queries invoked per original (isolated) query $N_{rec} = \frac{1}{1 - \frac{r\mathcal{P}_{rec} \log_2 |\Pi|}{2}}$.

As mentioned earlier, r_{up} fraction of the network events are ID-to-IP mapping changes in comparison to $1 - r_{up}$ fraction of queries. Routing tables are created based on P-Grid's randomized construction algorithm, and each peer is equally likely to be a routing table entry for other peers. If there are N peers populating a P-Grid with $|\Pi|$ leaves (replication factor of $N/|\Pi|$), then each peer has on an average $r \log_2 |\Pi|$ routing references, r for each of the $\log_2 |\Pi|$ depth of the search tree. Hence, each peer is used as a routing reference $\frac{Nr \log_2 |\Pi|}{N} = r \log_2 |\Pi|$ times in the whole P-Grid network. The probability that an update of a specific ID-to-IP mapping affects a routing table entry with i stale entries then is $r_{up} S_i \frac{r-i}{r} r \log_2 |\Pi|$, since the original queries are uniformly distributed and $r - i$ out of the r entries are susceptible to become stale.

The self-healing comes into action while processing the original queries, each of which leads to $N_{rec}-1$ recursive queries on average, and each of these recursive queries tries to update one routing reference. When recursive queries are triggered in state S_i, which occurs with probability $S_i(1-\mathcal{P}_{on})^{r-i}$, i out of the $N_{rec}-1$ recursive queries are initiated. Thus in state S_i, the probability that self-healing occurs due to an initial query is $(1-r_{up})\frac{1}{\mathcal{P}_{rec}}S_i(1-\mathcal{P}_{on})^{r-i}\frac{1}{i}(N_{rec}-1)$.

However, recursive queries may not always succeed. If the recursion is triggered when i cached references are stale and the $r-i$ others are offline, then i/r fraction of references can be updated at most. If ϵ_r is the probability that a recursive query fails, then $\binom{j}{i-j}(1-\epsilon_r)^{i-j}\epsilon_r^j$ is the probability that of the i possible repairs, only $i-j$ are successful, such that $j \leq i$ stale entries are left in the routing table even after recursive queries. $\epsilon_h = (\mu(1-(1-\epsilon_r)\mathcal{P}_{on}))^r$ and $1-\epsilon_r = (1-\frac{\epsilon_h}{2})^{\log_2|\mathit{\Pi}|}$ are derived as in Section 6.5.3.

For the dynamic equilibrium, the inflow to any state S_i should equal the outflow from S_i. Hence, the dynamic equilibrium equation is

$$\begin{aligned} r_{up} \quad & S_i \quad \frac{r-i}{r} r \log_2|\mathit{\Pi}| \\ + \quad & (1-r_{up}) \sum_{j=i+2}^{r} \frac{1}{\mathcal{P}_{rec}} S_j (1-\mathcal{P}_{on})^{r-j} \frac{1}{j}(N_{rec}-1) \binom{i}{j-i-1} (1-\epsilon_r)^{j-i-1}\epsilon_r^{i+1} \\ = \quad & r_{up} S_{i+1} \frac{r-i-1}{r} r \log_2 + (1-r_{up}) \frac{1}{\mathcal{P}_{rec}} S_{i+1}(1-\mathcal{P}_{on})^{r-i-1} \frac{1}{i+1}(N_{rec}-1)(1-\epsilon_r^{i+1}) \end{aligned}$$

The left hand side of the equation is the inflow into state S_{i+1} from S_i as well as from $S_j \forall j > i+1$, because of partial repairs. The right hand side is the outflow from S_{i+1}. The outflow is caused by two factors: The first is because additional entries turn stale; the second occurs whenever recursions are initiated and at least one cached entry is repaired.

We solve the above equations numerically for S_i, N_{rec}, $\mathcal{P}_{dyn}$, ϵ_r and thus determine the system's performance (dynamic resilience) at the dynamic equilibrium given the system parameters $\mathcal{P}_{on}$, r_{up}, r and $|\mathit{\Pi}|$.

6.6 Analytical and simulation results

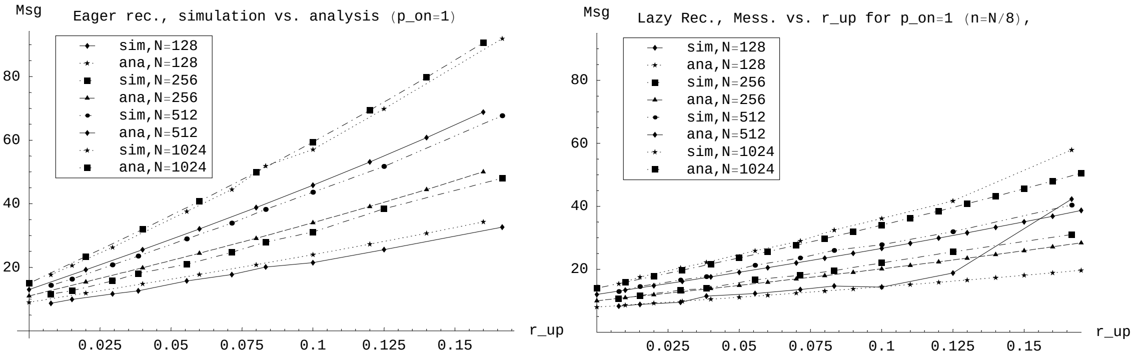
We implemented the algorithms described in this chapter in order to verify the analytical models by simulation and to demonstrate their scalability. We primarily report on results where $\mathcal{P}_{on} = 1$, i.e., peers only change their IP addresses but stay available, apart from some analytical results for the more general case of $\mathcal{P}_{on} \leq 1$. As in the analysis, we do not consider the cost of establishing a quorum and refer the reader to [50].

For the underlying storage, we construct P-Grid overlay networks of variable population sizes $N = 128, 256, 512$, and 1024 peers. We set the average replication factor to 8, such that the paths have an average length of 4, 5, 6 and 7 ($\log_2|\mathit{\Pi}|$ where $|\mathit{\Pi}| = N/8$). At each depth of the routing tables we maintain $r = 4$

references. The simulation is performed in rounds, where in each round we issue a random query at a random peer with probability $1 - r_{up}$, and with probability r_{up} we change the identity of a random peer and perform the necessary update. To reach a dynamic equilibrium state, we run a sufficiently large number of rounds (increasing from $25N$ to $100N$ for decreasing values r_{up}) and take the mean over the last $25N$ rounds to obtain values of the measured parameters.

For both the lazy and eager recursive query mechanisms we could show that the performance of the simulated system matches the predicted performance very well. We summarize the results in Figure 6.4, where we show the number of messages generated as a function of the frequency of updates, both when using the eager and the lazy algorithm.

We see that the message cost in the simulation is slightly higher than the predicted cost. This is due to the variation of the staleness of references. Since the message cost depends non-linearly on the staleness, variations inevitably lead to an increase as opposed to our average case analysis.



(a) Messages vs. r_{up}, eager algorithm

(b) Messages vs. r_{up}, lazy algorithm

Fig. 6.4. Simulation based validation of the analytical model

We observe that for the lazy algorithm for small N ($N = 128$) the model starts to break down when the value of r_{up} grows. This is so because for very small networks combinatorial effects such as cycles and deadlocks, which are not accounted for in the analysis, start to take effect, thus making the model inaccurate. On the other hand, we see that for a larger network population the predictions are increasingly accurate, as it is the case for any statistical model. The message cost scales gracefully. Thus for large networks our analytical model can be used to reliably predict its performance. We also observe that the lazy algorithm overall consumes slightly fewer messages than the eager algorithm.

Figure 6.5 shows the analytical predictions and the observed $\mathcal{P}_{rec}$ values from simulations for varying r_{up}, which may also be used to verify the accuracy of the analytical model. For N=128, the probability of recursion $\mathcal{P}_{rec}$ starts to increase dramatically, which also implies increase in message cost, and the simulation results deviate from the analytical predictions. Even for moderately large network sizes (N = 256 and higher), the results obtained from both simulations and analysis match well, which shows that the independence assumptions, and statistical results of the analysis are correct, once the system has a moderately large peer population. This is as expected from any statistical model.

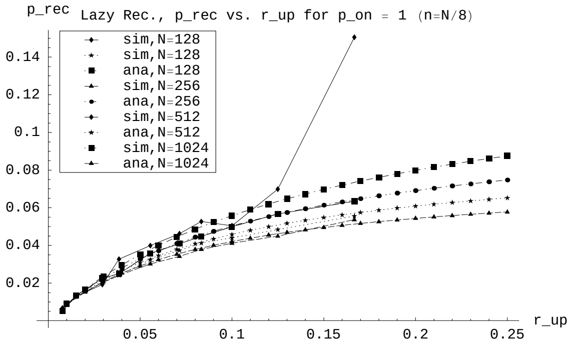


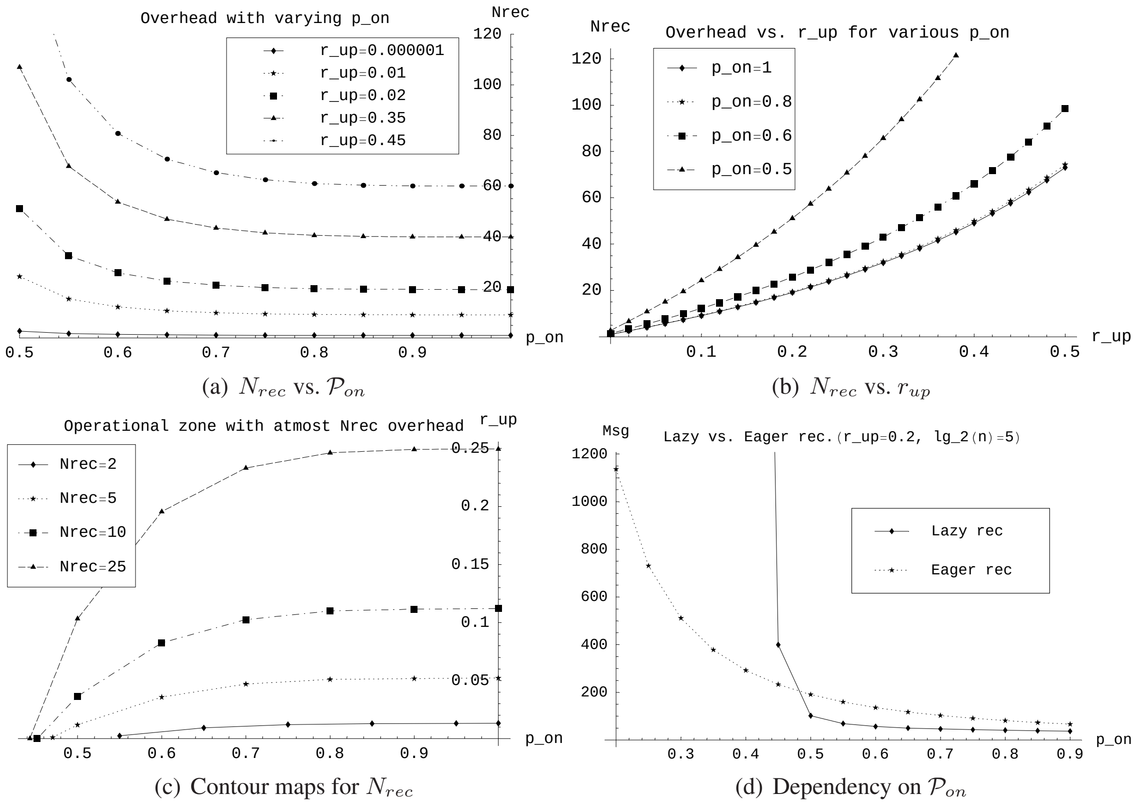
Fig. 6.5. $\mathcal{P}_{rec}$ vs. r_{up}, lazy

We use the analytical model to further explore the properties of the system in dynamic equilibrium. In Figure 6.6(a) we show how N_{rec} varies with varying $\mathcal{P}_{on}$ for any r_{up} value when using lazy recursion. We observe that the algorithm is very robust, and the message overhead is stable for a wide range of $\mathcal{P}_{on}$ values. This is so because for lower $\mathcal{P}_{on}$ values, even fewer stale entries render the routing table unusable, and trigger recursions. The intuitive expectation will thus be an increase in N_{rec}. However, such recursions also have the effect of quickly repairing the routing table, such that fewer recursions are triggered later. These two opposite effects balance, hence the wide stretch of $\mathcal{P}_{on}$ values where the overhead stays stable.

Figure 6.6(b) shows how the overhead varies with increasing network dynamics (increasing r_{up}), and we observe that it is more sensitive to r_{up} at lower $\mathcal{P}_{on}$ values.

While the use of recursion almost eliminates failures, tolerating even very low $\mathcal{P}_{on}$ values and moderately high network dynamics (high r_{up}), the incurred effort may not be affordable in a realistic network. In Figure 6.6(c) we thus provide contour maps corresponding to N_{rec} values, with $\mathcal{P}_{on}$ in the X-axis and r_{up} in the Y-axis. The interpretation of the plot is that if a system (participating peers) is willing to incur an N_{rec} fold increase of effort per query with respect to the ideal case ($\mathcal{P}_{on} = 1$ and $r_{up} = 0$), the network will operate for all $\mathcal{P}_{on}, r_{up}$ combinations below the curve, with the success probability being 1. If the system is unwilling to use more than N_{rec} effort and if the system operates in the region above the curves of Figure 6.6(c), there is a non-zero failure probability, which starts increasing with the increase of distance from the curve. Figure 6.6(c) thus captures two important tradeoffs in the system. The first tradeoff is that of efficiency versus probabilistic success guarantee of queries. The second tradeoff is the system's resilience against the two "demons" of the network, the network dynamics r_{up} versus average availability of peers in the network $\mathcal{P}_{on}$.

Finally, we analyze the dependency on varying values of $\mathcal{P}_{on}$. In Figure 6.6(d) we show the number of messages for a fixed $r_{up} = 0.2$ and path length 5. We see that for networks with more peers being online, the



(a) N_{rec} vs. $\mathcal{P}_{on}$

(b) N_{rec} vs. r_{up}

(c) Contour maps for N_{rec}

(d) Dependency on $\mathcal{P}_{on}$

Fig. 6.6. Analytical results

lazy strategy is advantageous. The tradeoff is that the lazy strategy collapses earlier. Thus the eager strategy is more resilient in the case of low availability. This suggests that combined adaptive strategies with various degrees of "eagerness" are an interesting approach for environments with varying online characteristics.

6.7 Related Work

6.7.1 Identity management

For unstructured P2P systems such as Gnutella [40] and hierarchical systems such as FastTrack-based (http://www.fasttrack.nu/) systems like Kazaa/Skype, dynamic IP addresses are less of a problem. For example, Gnutella builds an unstructured graph of peers in which each peer typically has 4 permanent connections to other peers. In the case that a connection drops, a peer simply tries to reconnect or tries to connect to another peer, it has learned about implicitly through Gnutella's routing process. Since no routing tables are maintained no inconsistencies can occur. However, this comes at the expense of very high network traffic. In hierarchical systems "routing tables" (in fact they are rather simple) can become inconsistent but their scope is limited, so the effect can be compensated easily with existing methods.

Freenet [38, 39] suggests the use of a third-party DNS service that allows the peer to update its name-IP mapping in special DNS domains. However, this introduces a dependency on a third-party service and an element of centralization into the architecture which is in contrast to the principles of decentralization to ensure scalability. Skype uses a central server to manage peers' identity.

DNS's original specification was extended by several RFCs (RFC2136, RFC2846, RFC2535), so that in theory it could maintain dynamic IP addresses through secure nameserver updates. However, this is very heavy-weight, requires very elaborate configurations, and is not intended for allowing a large number of peers to change the DNS database. Also an alternative DNS-based approach presented in [90] is still way too heavy for P2P systems and does not address security and unique identity of peers. To some extent our approach for recursive queries is similar to DNS's recursive lookup strategy which also updates caches during a name lookup. However, DNS's strategy is much simpler since DNS servers change their IP addresses very infrequently and thus the tree structure is basically static which simplifies routing a lot. Additionally, the number of participating DNS servers is considerably lower than the number of peers in a P2P system, the depth of the DNS tree is small, and, in contrast to DNS, our approach is self-contained, i.e., does not depend on a third-party infrastructure.

Other work using a DNS-like hierarchy without a single root has been done in the context of decentralized identification, such that some peers authorize other peers to use particular resources they provide. Any peer can authorize other peers to use its local resources as well as possibly delegate the authority to authorize other peers to do so. Systems following this approach are [43] and [14] which are based on [163].

6.7.2 Security issues

For security we devise a self-organizing public key infrastructure [50] which is comparable to PGP [66] which uses a similar, decentralized approach. PGP uses transitivity of trust, whereby, if P_A trusts that K_B is P_B's public key, and also relies on P_B (personally determined) to certify a third party's public key, then P_A will use K_C as P_C's public key, if P_B certifies it. The strength of such chains is determined by its weakest link and thus highly vulnerable. So [143] suggests to include multiple paths which, however, still offers only limited liability due to intersecting paths. Thus additionally authentication metrics [144] are required to quantify the reliability of such multiple paths. This approach, however, is heavy-weight, for example, finding multiple paths, loads the network considerably and both the multiple paths and the metrics need to be evaluated at each peer, and thus the effort is not shared. In contrast to that, our approach does not suffer from these problems at all. In P-Grid random (independent) peers replicate identity information (mappings) and thus our approach does not incur any costs in finding independent paths, the use of a quorum mitigates malicious behavior, and storage and search costs are distributed among the peers and require substantially lower computing and network resources. Additionally, since a subset of peers (to which searches are routed efficiently) are responsible for a given key, it is also simple to revoke or update mappings which is superior to PGP-based schemes. Further discussions are provided in [50]. However, security being a complex issue, and the security evaluation done so far being rather superficial with respect to the plethora of possible attacks, we would like to emphasize that a more rigorous evaluation is necessary. Particularly, there are other more critical attacks on the overlay, like Sybil attack [56], route-poisoning and Eclipse attacks. The problems looked into in these studies presupposes the concept of peer identity as well. These are more difficult attacks, and most proposed solutions require some centralized provisioning, particularly to prevent multiple identities by the same peer. As previously noted, these issues were beyond the scope of the work presented here.

6.7.3 Overlay route maintenance

The approaches of Chord [163], DKS [16] and Pastry [154] to deal with dynamic addresses of peers in an overlay network are the closest one to ours. As discussed previously in Section 6.1, these approaches are either more expensive; e.g., as in DKS where changes need to be immediately updated at all routing table levels, or stale entries are replaced instead of correcting them, as in Pastry, or there may not be any explicit notion of binding a peer to an identity over multiple sessions as in Chord.

The approach presented in [86] extends Tapestry [175] to address the joining and leaving of peers. In absence of self-healing, network maintenance is very expensive in this approach in terms of traffic (multicast-based partial flooding of the network), and there are no results on how the approach will cope with network dynamics ($\mathcal{P}_{on}$ and $\mathcal{P}_{dyn}$).

The self-maintenance mechanism proposed and studied in this chapter is considerably different from other route maintenance schemes in various respects: (1) Our proposal is the first one to apply a self-contained directory and (2) we explicitly model and address two sources of unreliability of the network, the dynamics of the underlying network (r_{up}), and the average peer unavailability ($1 - p_{on}$), and demonstrate that the self-contained directory may be used to heal the stale routing entries and operate at a dynamic equilibrium. (3) We propose a new family of reactive route maintenance schemes - the two extremes of which are called the eager reactive strategy or Correction on Use (CoU) and lazy reactive strategy or Correction of Failure (CoF), which have lower overheads than existing (pro-)active strategies like Correction on Change (CoC [163]) and proactive periodic correction (PC [112]). The essential idea in a correction on change strategy is to immediately propagate the changes (because of peer leave or join) to all affected peers. This is what is done in the self-stabilization algorithm of Chord. Periodic correction probes all routing entries at each peer periodically, and tries to rectify them. The main difference between CoC and PC is that typically CoC will be triggered by the node causing the change, or its immediate neighbors, who will then propagate the change, while in PC, every node tries to detect the changes locally.

Since the ring needs to be always maintained correctly in a Chord-like system, updating the predecessor and successor links immediately upon the occurrence of change is necessary. In P-Grid, as discussed previously (Section 4.5), new node joins do not require existing peers to change their routing entries. Similarly, because of structural replication, even if a node leaves the network (temporarily), alternate routes are typically available in P-Grid, thus making the use of lazier changes possible. Thus nodes joining, rejoining or leaving the network do not immediately affect the structure of the P-Grid network (key-to-peer associations do not change) even if query forwarding paths change and adapt to changes in the network topology. It has been noted in Chord as well as other ring-based topologies, that for the long-links the routing choices are rather flexible, and hence there is no real need of reflecting the changes immediately either, which can make use of lazier techniques possible (apart for the maintenance of the ring, which has to be corrected immediately upon the occurrence of changes).

Figure 6.7.3 summarizes the qualitative difference of the various route maintenance schemes - PC (Proactive or periodic Correction), CoC (Correction on Change), CoU (Correction on Use) and CoF (Correction on Failure).

The simple probing mechanism (PC) of [112] is very inefficient if there are infrequent changes. In comparison, CoC which is used in Chord (for node insertions) is a pragmatic downright reactive mechanism. CoC exploits the fact that *if there are no changes, there is no need for corrections*. CoC approaches devote their effort uniformly to all changes in the network. More directed maintenance effort proportional to the utility of a specific part of the network makes a more judicious use of resources. It calls for a reactive mechanism like CoU which initiates route maintenance only if a reference is indeed required but stale. CoF is even more pragmatic since it relies on the overlay's routing redundancy (similar to [174]) and hence resilience of the overlay, such that as long as a query may be answered, albeit with increased latency and effort, no repair is done. CoU and CoF need some mechanism to rediscover usable routing entries, for which we can use the self-contained directory. While CoF has the least overhead, and typically acceptable latency for a wide range of network conditions, it is unsuitable in an environment where network reliability and query frequency are very low. This is because, by allowing unusable entries to accumulate and not repairing them as long as there is no failure, CoF allows the network to reach a "point of no return." Intuitively, there is a *phase transition*, and the network gets totally disconnected. Figure 6.6(d) shows the effect of increased churn (by varying p_{on}).

A relatively recent experimental empirical study [146] also observed the deterioration of one specific reactive scheme in a variant of Pastry (named Bamboo) attributing the deterioration to positive feedback cycles. The conclusion drawn there was that periodic stabilization/recovery is more efficient. In that context, there are several things that needs to be pointed out here to disambiguate the results. First of all, this study focussed on maintenance of essentially the ring structure - i.e, successors/predecessors nodes.[6] In absence of structural replication, maintenance of the ring is indeed critical, and use of an active strategy is indeed desirable. Secondly, while our analysis is for a different class of reactive strategies, such a positive feedback

[6] For redundancy and recovery, multiple successors are required in a ring.

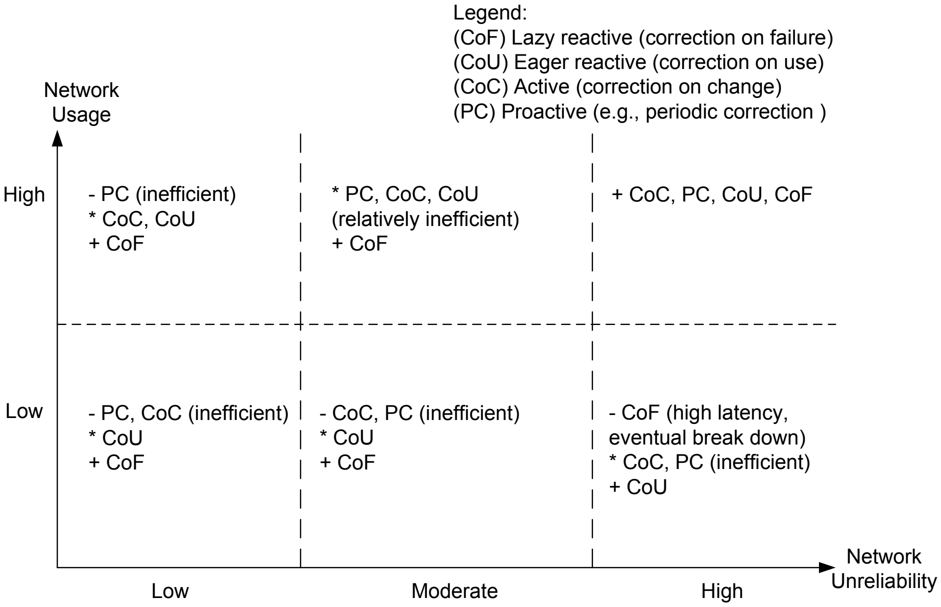
cycle is indeed observed for the case of CoF. In that respect the analysis in this chapter complements the empirical study [146].

A recent radical approach to deal with possible changes in (long-range) routing table entries is to preempt churn. So to say, try to speculate when a particular routing entry will potentially become unusable (based on statistical and historical information) and replace it with an entry which is speculated to be more reliable. This approach is used in a Kleinberg style small-world [98] choice of long-range links in a ring based topology (Accordion [105]). Another novelty in this approach is to pre-allocate the bandwidth budget for maintaining routing entries - which is used in order to determine the outdegree/indegree of individual peers. Simulations using synthetic work-loads show that the system provides better performance (in terms of maintenance cost, query cost) for a wide range of environments (churn) in comparison to the self-stabilization mechanism used in Chord. This approach is relatively new, and not compared with the existing reactive maintenance schemes, and may in fact well be used complementarily with lazy repair strategies in general. This is so because even while using a preemptive mechanism, unusable routes will be encountered. These (temporarily) unusable entries can then be repaired using a lazier mechanism. Moreover, none of these other route maintenance schemes exploit and provide persistence of peer identifiers, and hence (as mentioned previously) applications using meta-information about peers lose the opportunity to accumulate and use such information.

From Figure 6.7 and the discussion above we conclude that none of the existing approaches is suitable for all network conditions and usage patterns. Hence hybrid mechanisms are required. The self-tuning mechanism in [112] is an example where the probing period of PC is adapted based on the change rate, thus realizing aspects of CoC. However, neither PC nor CoC exploit the fact that maintenance is needed for only what is used, and incur higher overheads. Thus, from a route maintenance perspective, the family of self-healing routing (a hybrid of CoU/CoF) based route maintenance mechanisms are promising candidates, and complement well the recent preemptive mechanism [105].

6.7.4 Analysis of overlays under churn

Performance of overlays under membership dynamics has been of immense interest to overlay designers since the early days. Early proposals like Chord [163] looked at the probability of queries failing because of inconsistent routing tables. A more exhaustive work investigating the effect of the geometry of an overlay on its static resilience [75] was an important step towards a formal study of overlays under churn. A lower bound for overlay maintenance cost was also studied [106] prior to our work. These approaches do not look at any specific maintenance strategy and hence do not quantify the actual performance of the overlay under continuous effect of maintenance operations and membership changes. Our work pioneered the use of such a dynamic equilibrium study for overlay networks, using the P-Grid overlay and studying the Correction on Use and Correction on Failure based route maintenance schemes. Subsequently, similar to our dynamic



Legend:
(CoF) Lazy reactive (correction on failure)
(CoU) Eager reactive (correction on use)
(CoC) Active (correction on change)
(PC) Proactive (e.g., periodic correction)

Network Usage \ Network Unreliability	Low	Moderate	High
High	- PC (inefficient) * CoC, CoU + CoF	* PC, CoC, CoU (relatively inefficient) + CoF	+ CoC, PC, CoU, CoF
Low	- PC, CoC (inefficient) * CoU + CoF	- CoC, PC (inefficient) * CoU + CoF	- CoF (high latency, eventual break down) * CoC, PC (inefficient) + CoU

Fig. 6.7. Route maintenance taxonomy and qualitative comparison. A preemptive strategy like [105] is excluded here, since it is not exactly a route repairing strategy, but it can be used to complement any of the route maintenance strategies for the full spectrum of network churn characteristics.

equilibrium analysis other works [101, 102] use formal tools from statistical mechanics (master-equations) to study the Chord overlay under its self-stabilization algorithm. While the system parameters used in the analysis in [101, 102] and the details of the analysis are somewhat different than ours, again, such a dynamical equilibrium analysis accurately predicted the behavior of the Chord overlay under continuous churn and maintenance, vindicating our original proposition of use of a dynamic equilibrium analysis methodology to understand the dynamics of large-scale dynamic systems. It also exemplifies how, as we originally envisioned, such analytical tools from complex systems will be more widely adopted in the context of distributed systems in general, and particularly for peer-to-peer systems.

A critical decision when modeling and analyzing complex systems is to choose appropriate parameters. The existing dynamic equilibrium studies [5, 102, 101] choose these parameters in an intuitive but ad-hoc manner. A recent systematic modeling approach proposes the use of *intensive variables* [59]. Intensive variables is a concept inspired from physics. Intensive variables are system invariants which are not affected with the system's scale (as long as the system size is large enough to make statistical properties relevant). For instance, density of an object is an intensive variable, unlike its mass. Intensive variables are empirically obtained. In the context of computer science, an intensive variable is determined by proposing a hypothesis and validating it with simulations based on whether a "data collapse" is observed or not. A data collapse essentially encodes any functional relationship among different system parameters. The advantage of using intensive variables to model, analyze and express system properties is that then the system is described using parameters which are independent of the scale, and it obviates the need to do an exhaustive enumeration of the system properties for all parameters by eliminating redundancies because of functional relationships among such parameters. Some early study to identify intensive variables, looking into the Chord overlay network and its periodic stabilization algorithm has been done [59]. It has been shown in [59] that both the node density on the identifier space as well as the ratio of perturbation (churn) to repairs are intensive variables. It is not evident from this initial study if the intensive variables themselves change based on algorithmic changes in the system. For instance, the current analysis was assuming periodic stabilization of the Chord network. If peers adapt locally the period of stabilization itself (e.g., as proposed in [112]), this may change the qualification of certain variables to be intensive or not. Nevertheless, such a systematic approach to choose appropriate system model parameters based on intensive variables is generally interesting for distributed systems research, and complements the analysis methodology of studying dynamic equilibrium in overlays under churn by identifying appropriate modeling parameters systematically rather than heuristically, as has been the practice so far.

6.8 Conclusions

This chapter described a decentralized, self-maintaining, light-weight directory service. We have demonstrated that our algorithm is robust and applicable in unreliable environments such as current peer-to-peer systems and that it operates well, even if we assume low online probabilities. Our approach has six major contributions: (1) We separate identity from network properties and thus introduce the concept of logical independence into overlay networks, (2) we provide a general approach to identify entities and to bind arbitrary information to them, (3) we demonstrate that the approach does not corrupt structural properties of the used P2P system and retains existing knowledge and semantics, (4) we explicitly address security against impersonation to guarantee the correctness of identities, (5) we have explored a P2P system's dynamic resilience in the presence of changes in the underlying network, in contrast to other works that have only addressed static resilience of P2P systems and (6) we proposed a novel family of reactive route maintenance schemes exploiting structural replication of the overlay; Correction on Use and Correction on Failure being two special (extreme) instances.

The directory service is based on the P-Grid P2P system and applied in P-Grid itself to mitigate the problem of dynamic IP addresses. To prove the efficiency and applicability of our approach we have provided an analytical model for the dynamic equilibrium case and have evaluated our algorithm based on this model. Additionally, we have provided simulation results to verify the correctness of the model. Our infrastructure offers a sufficient level of security against impersonation attacks.

7. Experimental evaluation on PlanetLab

“All models are wrong, but some are useful.” — George E.P. Box

7.1 PlanetLab as an experiment testbed

We used the PlanetLab infrastructure [37] to obtain results from moderately large-scale experiments based on a real implementation of some of the algorithms introduced so far under realistic networking conditions and to verify the validity of our theoretical predictions and simulation results.[1]

PlanetLab [37] (http://www.planet-lab.org/) is a global testbed for large-scale experiments with distributed systems. At the time of the experiments in the first half of 2005, PlanetLab consisted of approximately 530 computers (nodes) geographically distributed over the whole planet running a modified version of Linux to support efficient administration and resource sharing for large-scale experiments. Nodes in PlanetLab are connected via a diverse collection of links.

In PlanetLab, computers are spread over different administrative domains and are subject to different network adminstration policies and management. So PlanetLab is inherently indeterministic. It is not meant to run controlled experiments to obtain reproducible results [161]. Nor does the PlanetLab topology reflect the internet topology. Moreover, often multiple experiments are run simultaneously on the same computers influencing the results. Nonetheless, PlanetLab has recently become an interesting test-bed for deployment and experimentation with large-scale distributed system, because it provides a real geographically distributed environment where actual implementations using networking protocols can be deployed, not only for bench-marking and performance evaluation but also to validate the functional correctness of prototype and full-fledged softwares.

[1] A disclaimer and an acknowledgement: The PlanetLab experiment results are based on the actual Java implementation of some of the algorithms introduced in the previous chapters which have been integrated with the rest of the Gridella/P-Grid software. The implementation as well as the experimentation is a joint work spearheaded by Roman Schmidt and substantially supported by Renault John. Without their contribution neither the software nor the experiment results in a real internet environment would have existed. I thank them both for the collaboration we had in developing the system from the conceptual embryo to the actual functional software, and also for permitting me to include the experimental results. The PlanetLab experimental results have been included in this thesis in order to show the practicality of some of the previously introduced algorithms.

7.2 Objectives and scope of the experiments

The deployment and experiments were conducted with a Java based P-Grid software implementation, which incorporates several of the algorithms introduced earlier. Particularly the P-Grid construction algorithm (recursive use of the adaptive eager partitioning **(AEP)** algorithm described in Section 4.3.1) and the basic search/retrieve Algorithm 1 and range-query Algorithms 2&3 were tested, and the findings are reported next.

Our experiments on PlanetLab ran on up to 300 computers depending on the number of available nodes. Each node executed one instance of a P-Grid node. When interpreting the results presented in the following, it is important to consider that PlanetLab is shared by a large number of research groups for experiments that are executed in parallel and thus mutually influence the performance considerably especially with respect to absolute latency. Absolute latency is also affected by the implementation artifacts like whether a proper garbage collection mechanism is in place or if there are memory leaks. Subsequent to the experiments the software has undergone several bug fixes and fine-tuning to improve performance as well as new features based on other algorithms introduced in this thesis as well as others are being integrated.

7.3 Experimental setup for overlay construction by recursive re-partitioning

We deployed the P-Grid software, i.e., the peers, on all available nodes at the times the experiments were conducted and assigned 10 keys chosen uniformly randomly from a real text collection (taken from the Alvis information retrieval EU project [30]) to each peer. This relatively low number of keys was chosen to speed up experiments and as we have already speculated based on simulation results, sample size has little influence on load balancing. To validate our experiments, we also performed tests with larger numbers (up to 2000 keys per peer) and used various distributions, including uniform random distribution and Pareto distribution.

The time-line of the experiments was as follows: In an initial phase starting at time t, peers joined the system by contacting a bootstrap peer (until $t + 30min$) and formed an unstructured overlay network (from t until $t + 45min$) which was used later to replicate data a fixed number of times (from $t + 45min$ until $t + 60min$). In the replication phase peers randomly chose 5 peers from the unstructured overlay network to replicate their data. Subsequently, from $t + 60min$ to $t + 300min$, the structured overlay network was constructed using the approach presented in Chapter 4. We were especially interested in evaluating the bandwidth consumption during this phase and to verify whether the theoretically predicted load balancing properties of the algorithm are achieved under realistic networking conditions. Then we ran queries on the constructed overlay network ($t + 300min$ to $t + 400min$) to analyze search performance. Each peer performed a search every 1–2 minutes. In the final phase ($t + 400min$ to $t + 500min$) network churn was emulated to evaluate the resilience of the overlay network. Each peer independently decided to go offline for a duration of 1–5 minutes every 5–10 minutes which caused considerable churn that the system had to withstand and compensate.

7.3.1 Experimental evaluation

We first verified that the system behavior matched the theoretical predictions and the simulations. The experiment was performed with 296 peers and compared to simulation results using the same number of peers and the same key set.

The quality of load balancing was evaluated as defined in Section 4.8.1 and was practically identical for simulations and experiments, with an average of 0.38 for 10 simulations (the standard deviation is 0.05) respectively a value of 0.39 for the experiment. This indicates that the theoretically predicted load distribution properties were met quite accurately by the implementation even under realistic network conditions with slow connections and communication failures.

We now report some system measurements that we made to evaluate the performance of the overlay network, both during the construction phase, as well as in its operational lifetime both in a static situation (no change in peer population) as well as under churn.

Figure 7.1 shows the number of peers in the overlay during the whole period of experimentation. We see how peers first joined the network and the number of peers in the network increased to the maximal number. Then during the construction phase this number was stable (approx. 300 peers) while decreasing again in the final phase where we emulated network churn and a substantial dynamic fraction (around 25%-30%) of peers became unavailable.

Figure 7.2 shows the aggregate bandwidth consumption of all peers (maintenance and queries) in Bytes/sec. During the construction phase the bandwidth consumption reached a peak of 250 Bytes/sec per peer. The maintenance consumption decreased quickly down to less than 100 Bytes/sec and became negligible compared to the bandwidth consumed by queries.

Figure 7.3 shows the average query latency and its standard deviation. The absolute values were relatively high and essentially reflected the poor response time of PlanetLab nodes. The response time was slightly higher with a larger deviation under network churn because requested peers were sometimes offline which had to be compensated by forwarding the queries to others.

We observed that the number of query hops per query was as low as theoretically expected, i.e, approximately half of the mean path length, even under churn. The average path length was slightly below 6 and the average number of query hops per query was approximately 3. Moreover after the construction phase had led to full evolution of the overlay network, all peers discovered all their replicas, and the system had an expected mean replication factor of 5, as intended, and success rate for queries was between 95% and 100% even under network churn. Queries were mainly unsuccessful because of network problems such as lost or corrupted messages.

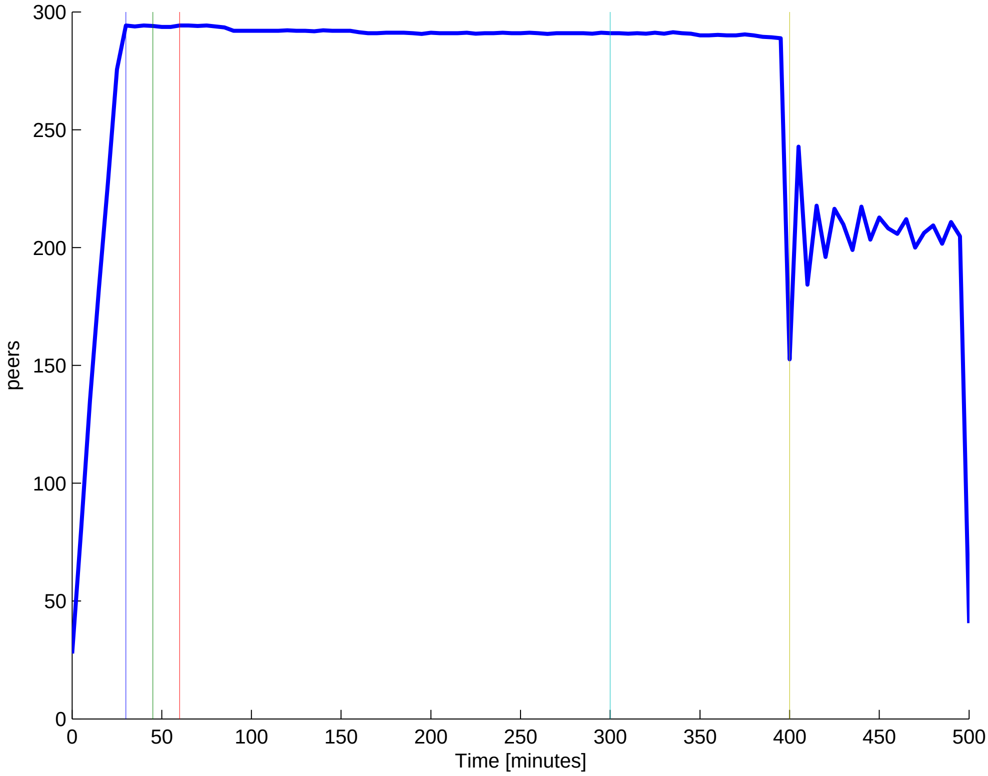


Fig. 7.1. Number of participating peers

Finally, we would like to point out that the presented experimental evaluation is still limited in the following sense: The moderate number of peers does not allow us to obtain significant results on the reduction of latency during bootstrapping as predicted by our theoretical analysis in Section 4.4.4; which is one of the main properties of our approach.

7.4 Experimental setup for evaluation of the range query algorithms

In the experiments to evaluate the min-max and shower range query algorithms which were introduced in Section 3.3 we used a network of 250 peers each running on a dedicated physical PlanetLab node. We inserted 2500 unique data items into the system and required an average replication factor of 5. Thus initially we had a total of $5 * 2500 = 12500$ data items in the system and each peer was responsible for $5\frac{2500}{250} = 50$ data items. The real number of the data items in the system in fact was higher as for load-balancing each peer was required to manage a minimum of 50 and a maximum of 100 data items, and given the randomized construction approach of P-Grid, each peer would thus hold on average 75 data items, i.e., the total number of data items in the system was $250 * 75 = 18750$.

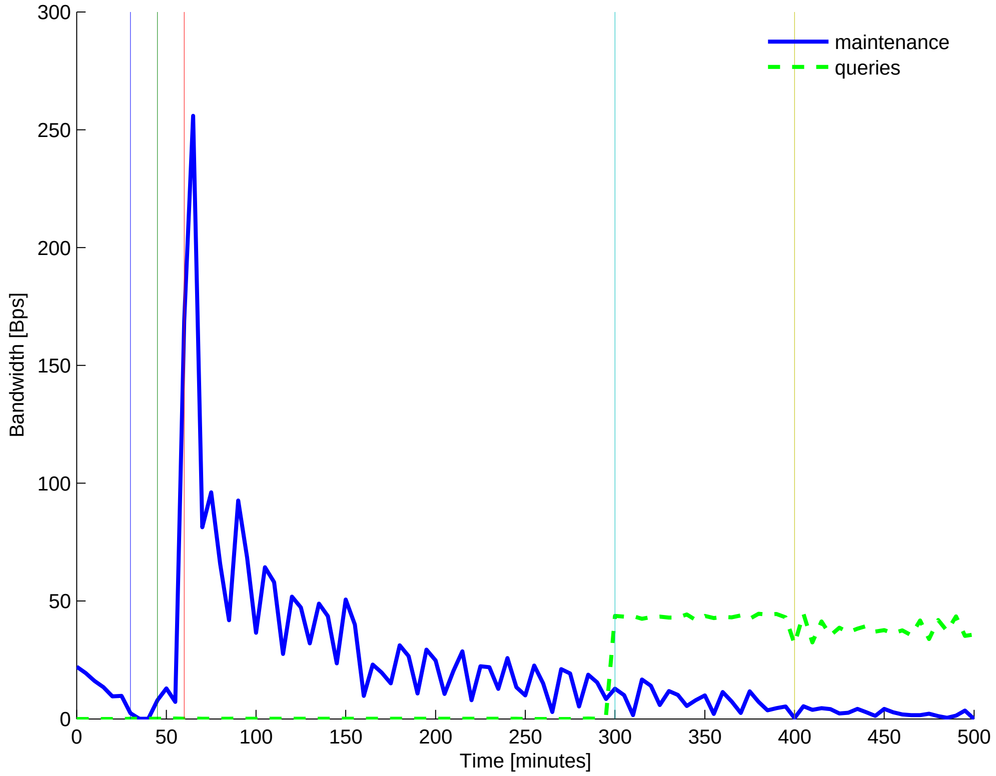


Fig. 7.2. Aggregate bandwidth consumption

To show that range query on P-Grid network is efficient for diverse data distributions, we used two very different data sets, one uniformly distributed and one Pareto distributed (with a probability density function of $\frac{a\,k^a}{x^{1+a}}$ and parameters $k = 1$ and $a = 2.0$) as shown in Figure 7.4.

Pareto is a typical long-tail distribution which occurs frequently. We observed in the previous set of experiments that P-Grid copes well with such distributions because of the underlying load-balancing algorithm which balances both storage and replication load. We thus safely infer that if the results are good for a Pareto distribution, the system will perform equally well for other distributions including other long-tailed ones like Zipf distribution.

In the experiments each peer selected randomly 10 data items of a data global set according to one of these distributions. The peers then constructed a P-Grid which had an average height of $\log_2 \frac{2500}{10*5} = 5.6$. Then range queries which affected data from all partitions of the data sets were issued. The queries were started from randomly chosen peers with random lower range bounds, and were constructed in a way, such that they should ideally return 50, 100, 150, 200, 400, and 800 unique data items. For each of the six answer set sizes, each of the two distributions, and each of the two algorithms, one query was issued by each of the 250 peers, i.e., a total of $6 * 2 * 2 * 250 = 6000$ queries resulting in 250 values per data point in Figures 7.5–7.8.

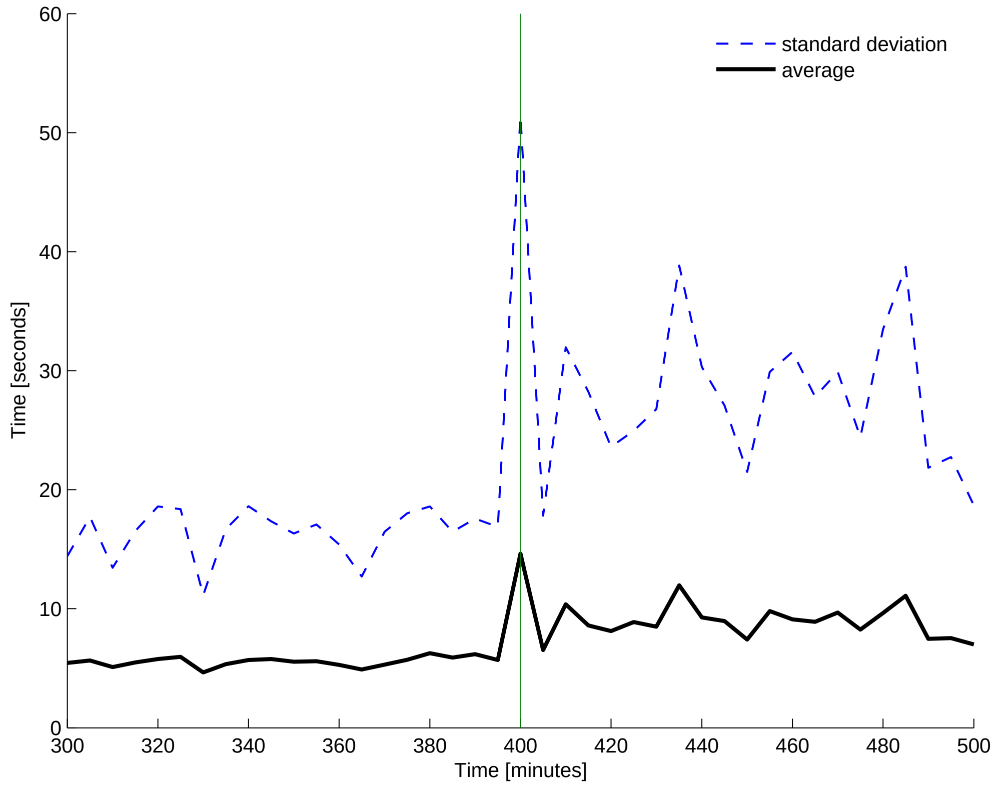


Fig. 7.3. Query latency

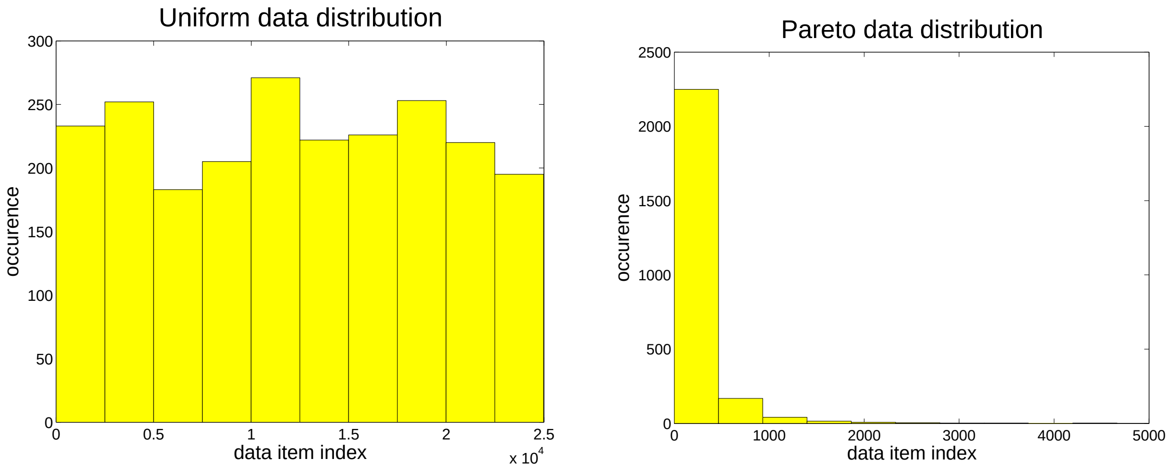


Fig. 7.4. Data set distributions used for range query experiments

7.4.1 Experimental evaluation

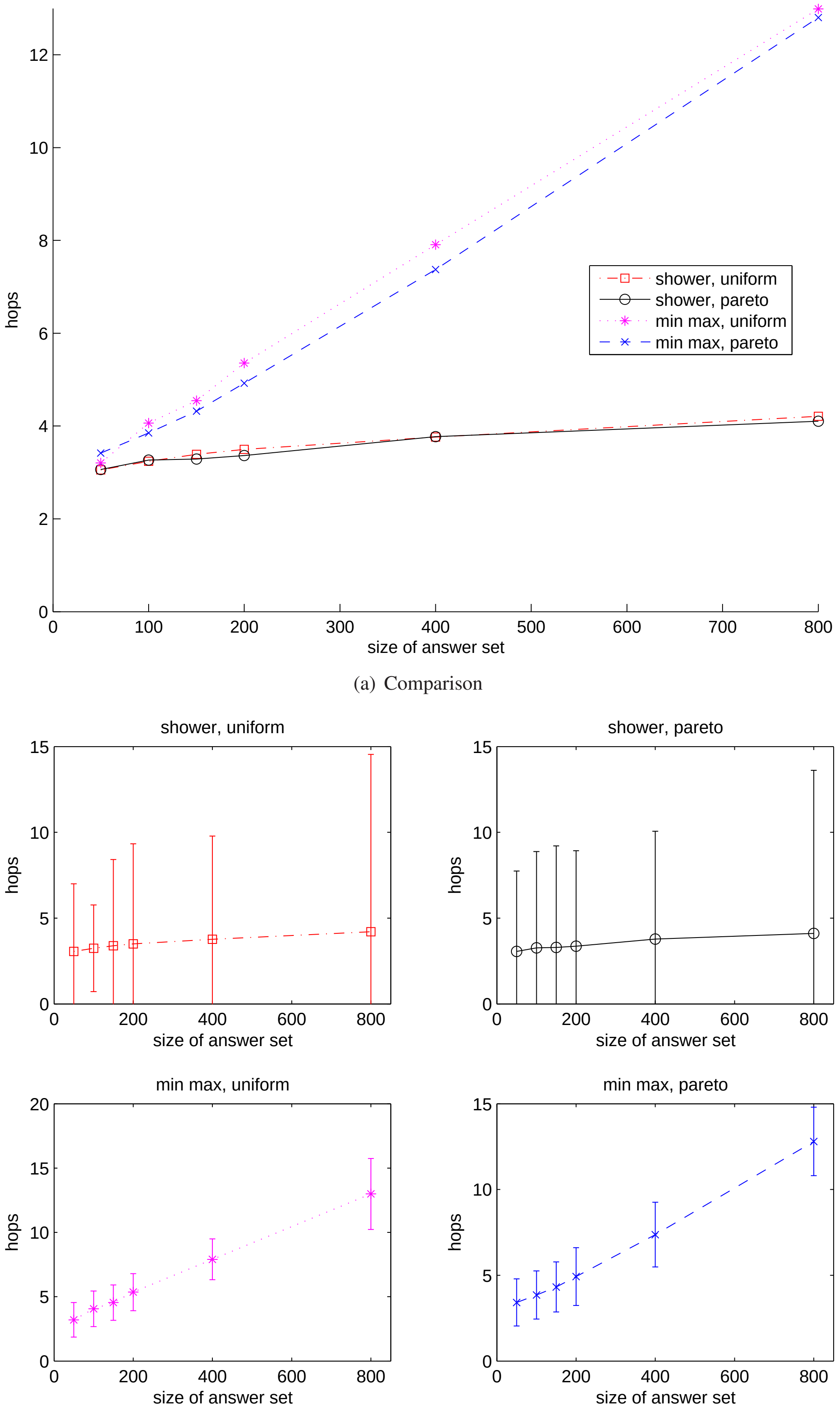
There are several performance metrics of interest to evaluate the P-Grid system as well as the algorithms for their suitability to support range queries. This includes load-balance characteristics (storage, replication, and query load), data fragmentation, as well as message costs and latency for various data distributions. P-Grid's efficient multi-faceted load-balancing characteristics for diverse workloads have already been discussed. This allows us to use order-preserving hashing to ensure low data fragmentation, while the underlying network transparently takes care of load-balancing despite the skew in key distribution over the key-space. Earlier in the chapter we already reported the load-balancing results for the implementation also.

Thus the main objectives of the experiments here was to demonstrate the cost/latency trade-off of the range query algorithms, and to show that because of the use of a load-balanced trie-structured overlay network, the cost of range queries is independent of the data distribution and the size of the range, but only dependent on the used algorithm and the size of the answer set - as expected from the theory (Section 3.3). From the experimental results presented in the following, we can observe that the cost and latencies are indeed independent of the distribution and indirectly prove that the overlay network had good storage-load balancing characteristics.

Figure 7.5 shows the costs incurred by range queries in terms of message latency (hops), i.e., the maximum number of messages required to hit each sub-partition of the range, i.e., one peer in each sub-partition. Figure 7.5(a) shows a direct comparison of the experimental results for the four combinations (2 data distributions and 2 range query algorithms) and Figure 7.5(b) gives the standard deviations of each of the four types of experiments as error bars.

On an average we needed 3 hops to reach first a peer responsible for some key-space within the range for both types of algorithms. But the min-max algorithm suffers from the sequential traversal of the range to reach all sub-partitions after having reached the peer responsible for the lower bound. This leads to increasing hop counts with increasing range sizes whereas for the shower algorithm the number of hops remained constant, i.e., it was rather insensitive to the size of the answer set as an increase in the number of hops for this algorithm basically means that the range had exceeded one level in the tree and an additional hop was necessary as the “shower” had to start at the next higher level. However, this benefit came at the cost of an increase in the overall messages as shown in Figure 7.6. Figure 7.6(a) shows a direct comparison of the experimental results and Figure 7.6(b) gives the standard deviations of each of the four types of experiments as error bars.

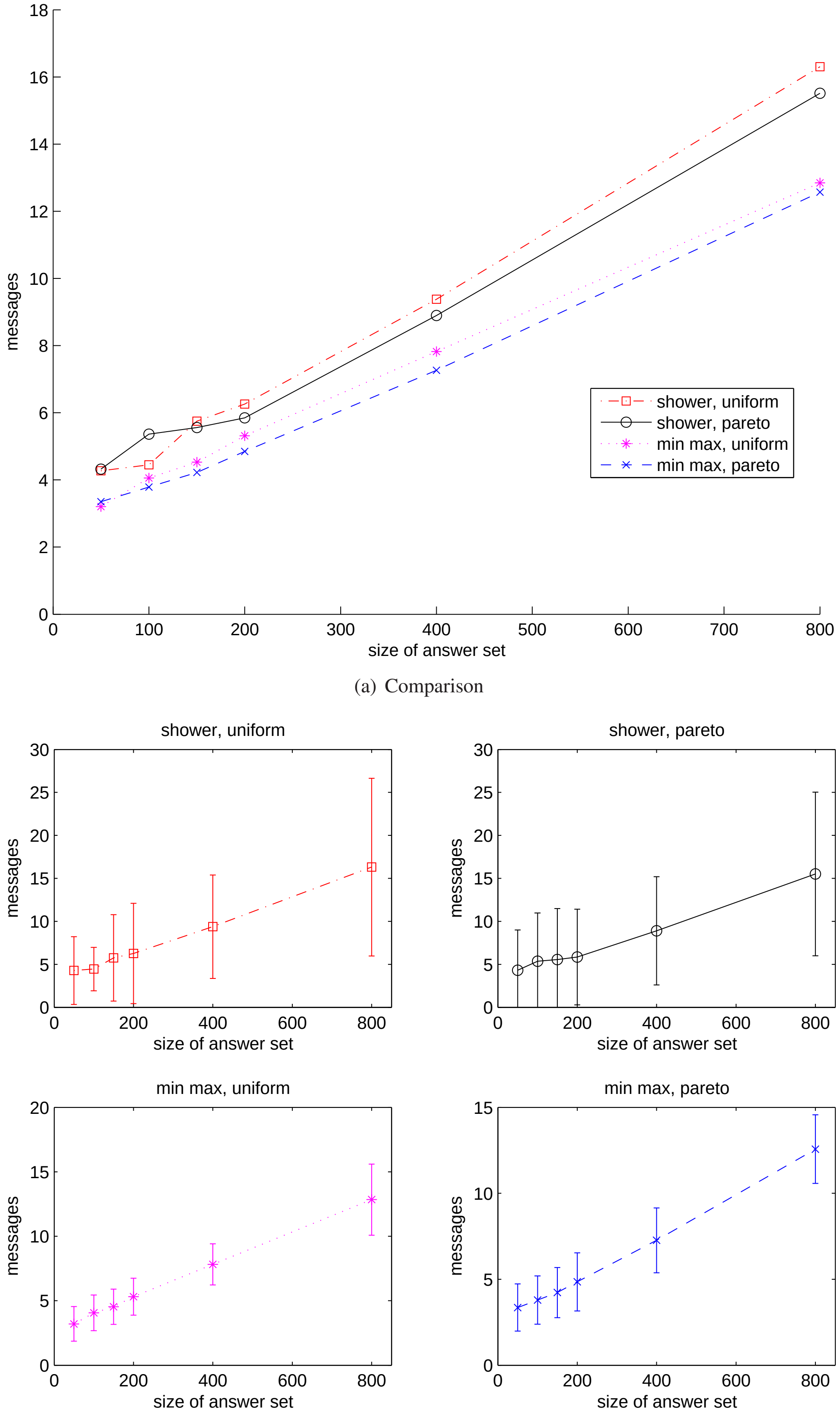
The shower algorithm requires a slightly higher number of messages but improves latency as it sends them to the responsible peers in parallel. Therefore all peers responsible for a range section were reached after 3 hops (in the experiment's setup) independent of the range size. Range queries with an answer set size of 50 were answered mostly by one peer because peers on an average were responsible for 75 data items. It can further be seen that both algorithms performed well for both data distributions and scaled as expected



(a) Comparison

(b) Standard deviation

Fig. 7.5. Message latency (hops)



(a) Comparison

(b) Standard deviation

Fig. 7.6. Message cost

from theory. An increase of the answer set size by a multiplicative factor of the average peer storage size yielded an additional message on average which is the best possible result achievable with limited storage available at the peers and again indirectly proves the effectiveness of the storage-load balancing.

Figure 7.6 also shows the total number of peers involved in a range query, i.e., the number of peers forwarding or replying to a range query. For the min-max algorithm this number was equal to the number of messages because only one message is first routed to the lower bound and then forwarded to the higher bound. Therefore the number of peers forwarding a query to a peer of the desired range is smaller than for the shower algorithm. More peers are involved during the shower algorithm because messages are sent in parallel to reach desired peers.

In terms of query latency, it is interesting to see that the shower algorithm is almost insensible towards answer set sizes, this can be seen in Figure 7.7, where the latency grows very slowly.

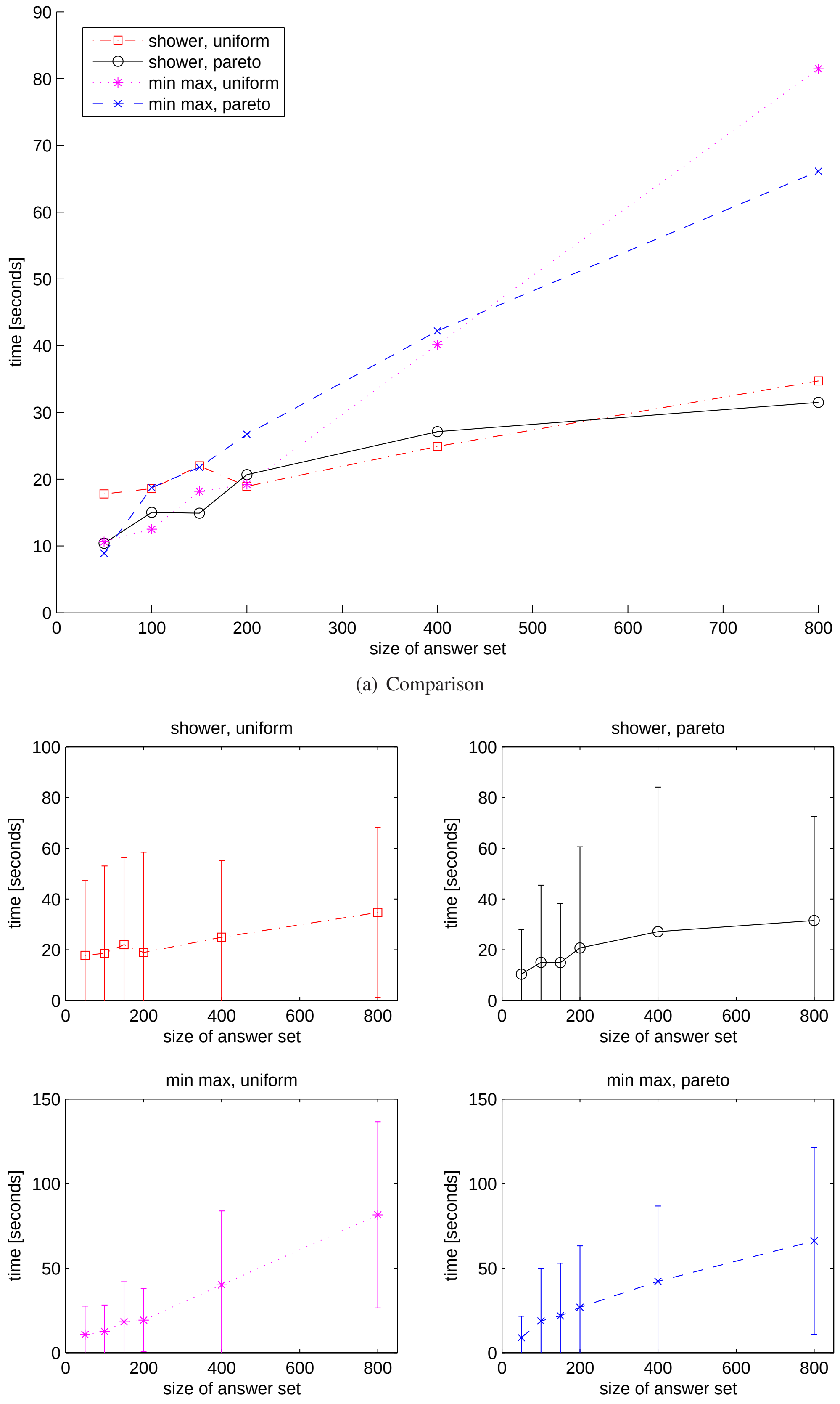
This can be explained by the fact that a considerable number of data items would have to be added before the trie increases its height which is the dominant contribution to the latency for this algorithm. For the min-max case the latency increased for obvious reasons as messages are forwarded sequentially which increased the latency. Here an increase of the height of the trie has a much more dramatic influence as the min-max algorithm heavily depends on the width of the interval. While increasing the height of the trie means only an additional hop for the shower-algorithm which is processed largely in parallel, for the min-max algorithm the number of sequential messages increases by a factor of 2 on average. Note that this is expected from theory, since the height of the tree will increase by 1 only if approximately twice the data items are in the same range, and in the min-max algorithm, both latency and message costs are proportional to the number of data-items in the answer-set.

A side result which can be inferred from these plots is that the smallest range queries involving 3–5 peers took approximately 10–20 seconds on an average. Larger range queries using the min-max algorithm took a multiple of that. This can be explained by the success and adoption of PlanetLab as an experimental testbed, since a large number of experiments are conducted concurrently which considerably slows down PlanetLab's overall performance.

Finally, in Figure 7.8 we show what level of result completeness we could achieve with our range queries.

This measure represents the percentage of received data items as answers to a range query with respect to the actual number of data items inserted (present) in the specific range. The result completeness is around 90% and is mainly independent of the range sizes and the data distributions.

We observed several problems during our experiments in respect to the PlanetLab environment, for example, communication problems and crashes of PlanetLab nodes (not of the tested P-Grid system but the physical PlanetLab computers), which explain the non-exhaustive results. Such failures because of unreliable peers are characteristic of any deployed P2P system, the relatively high success rate in fact demonstrates the robustness of P-Grid under churn. Smaller scale experiments in a local environment with lower numbers



(a) Comparison

(b) Standard deviation

Fig. 7.7. Query latency (time)

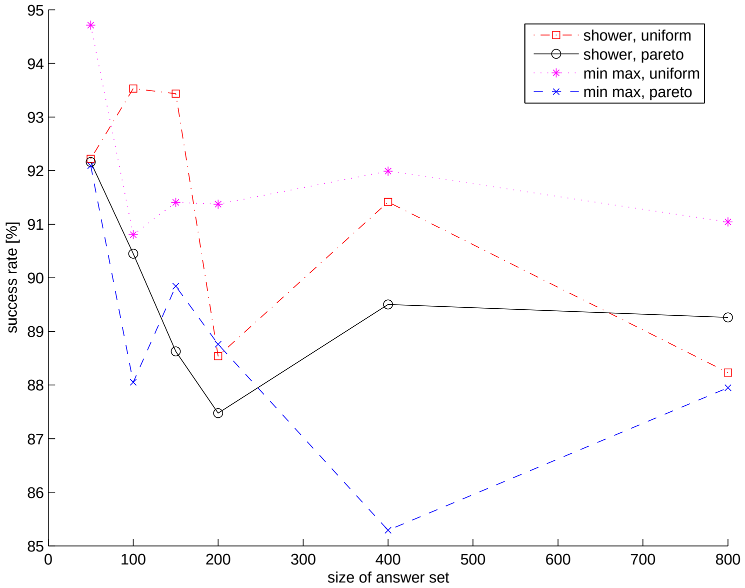


Fig. 7.8. Result completeness of the range query algorithms

of nodes and fewer emulated node failures have proven the functional correctness of our implementation and provided a 100% success rate. To increase the success rate on PlanetLab or in a real-life deployment we need to increase the replication factor, i.e., data is replicated more often, and thus node failures could be possibly compensated better. This will increase the maintenance overhead but should provide better results. However, due to the duration of the experiments and the lack of possibility to assess the conditions on PlanetLab that caused a certain experimental result and behavior, we have no experimental evaluation using higher replication yet. In the experiments discussed above we used a replication factor of 5 on average. Each key space partition (and hence data item) was actually replicated between 1 and 10 times. Taking this into account and the very dynamic situation on PlanetLab a success rate of 90% seems reasonable. In future work, we will explore the possibility to adapt replication to the dynamic situation on the physical network, apart integrating the structural replication re-balancing algorithm introduced in Section 4.6 to improve the overlay's resilience and performance.

7.5 Conclusion

The evaluation of the actual implementation based on some of the algorithms presented in this dissertation not only shows the practicality of the algorithms, the experimental evaluations also provide a final validation of some of the analytical predictions. The implementation and the experiment results thus help covering the full spectrum starting from system design and analysis and finally culminating in an implementation and evaluation in a realistic environment. The deployment of full-fledged functional software in a highly distributed environment (like PlanetLab) also is a stepping stone towards making the software available to a wider audience of end users as well as developers of other systems and applications that may use some of the functionalities provided by the systems designed in this dissertation.

Part III

Content management in internet-scale systems

8. Efficient redundancy maintenance in storage systems

"I think you're begging the question," said Haydock, "and I can see looming ahead one of those terrible exercises in probability where six men have white hats and six men have black hats and you have to work it out by mathematics how likely it is that the hats will get mixed up and in what proportion. If you start thinking about things like that, you would go round the bend. Let me assure you of that!" — Agatha Christie, The Mirror Crack'd

8.1 Introduction

In the recent years there has been an increasing trend to use resources at the edge of the network - typically desktop computers interconnected across the internet provide services and run applications in a peer-to-peer manner, as an alternative to the traditional paradigm of using dedicated infrastructure and centralized coordination and control. Similar services in a relatively more dedicated infrastructure like PlanetLab [131] is also a growing trend. One such extensively studied application is that of collaborative storage systems, where free storage space of individual computers is used in order to realize persistent and highly available data storage [25, 46, 80, 103, 145, 168]. Such collaborative storage systems can be used by a wide range of applications - as backup service for individual users [42], public services like digital library or an internet archive [114] or file systems [46] - to name a few. Moreover, the peer-to-peer paradigm need not necessarily be used only out in the open in the internet. Private enterprises spread geographically over various sites can use the same peer-to-peer paradigm for a cost effective automated back-up service within their own corporate intranets with dedicated and much more reliable infrastructure.

Large-scale systems in general, and peer-to-peer systems in particular, are prone to the unreliability of individual participants. Particularly in a peer-to-peer environment, individual peers often leave and rejoin the network for relatively shorter session times over a long period of time (lifetime), before leaving the network permanently. Also, some peers may become temporarily unreachable from other peers because of communication network problems. Irrespective of such autonomous and whimsical behavior of individual participants, for any practical usability, it is necessary to design systems which are stable and reliable, even if the individual participants are unreliable. Such reliability is achieved using redundancy. In order to maintain the redundancy over a long period of time, it is also necessary to continuously compensate for the lost redundancy in presence of continuous membership dynamics (churn). The maintenance operation needs to be sufficiently aggressive in order to provide a minimal redundancy and resilience, but at the same time the maintenance overheads need to be kept low. This needs prudent design of the maintenance scheme to better explore the cost-performance tradeoffs. If the system is to be deployed in diverse environments, or to operate in an environment where the dynamics can change substantially over time, it is also desirable that the system adapts automatically to the environment, with as less human administration to tune the parameters as possible.

In this chapter we propose and analyze such an adaptive self-organizing maintenance algorithm for collaborative storage systems which is both efficient in terms of maintenance cost as well as robust so that the storage system can tolerate a wide range of churn as well as rarer but likely correlated failures.

The algorithm we propose is a *randomized lazy maintenance* scheme which has quantitatively superior resilience against both a wide range of churn level as well as rarer but inevitable (in large-scale distributed systems) correlated failures than known maintenance scheme [25]. Our maintenance scheme achieves this resilience at a comparable (and often lower) maintenance overhead. The basic idea is to probe randomly for a minimal number of redundant fragments of the stored content. Depending on the current available redundancy the sample size thus automatically adapts, as does the effort to replace the detected lost redundancy. So to say, as availability decreases replacement rate increases smoothly for our scheme.

We evaluate our maintenance scheme analytically, validating the results using simulations. To that end we study *time-evolution and steady state characteristics* of storage systems under churn by introducing analytical tools used in the study of dynamical systems [85]. We use a Markov model for studying P2P storage systems under churn [47], similar to our analysis of structured overlays under churn [5]. We specifically investigate whether, given a specific amount of churn and a chosen maintenance mechanism the system works in a steady-state and if so, how stable is the steady state to additional failures (possibly caused abruptly by correlated failures) and what is the maintenance cost.

Despite the immense interest in building reliable P2P storage systems, the issue of churn for storage systems is not properly studied, apart resorting to simulations [168] which do not capture the interplay between - churn, specific properties of the kind of redundancy used and the specific maintenance operations. OceanStore [103, 169] looks only into resilience against disk failures, and is not designed for a dynamic setting as is more likely in a peer-to-peer setting. Other systems like CFS [46] and Glacier [80] eagerly maintains redundancy - which however does not explore the trade-offs of cost and resilience. TotalRecall [25] proposed a heuristic lazy maintenance scheme to reduce the maintenance cost, and used simulations to demonstrate the tremendous cost savings using their heuristic in comparison to the use of eager repairs. Buoyed by our successful use of steady-state analysis for overlays under churn, we wanted to reuse the analysis methodology for P2P storage systems. Our analysis of the existing (deterministic) lazy maintenance scheme [25] not only provided an exact picture of the interplay of churn and repair operations, but also prompted us to design a better (randomized) lazy maintenance scheme. While our pragmatic lazy scheme could have been designed accidentally, historically the fact remains that it was inspired by the analysis based precise understanding of the system's dynamics, moreover the analysis provided a framework for quantitative evaluation of each of these schemes as well as comparative study (similar to the case of structured overlays as studied earlier in Chapter 6).

From such a comparative study, we determine that the (randomized) lazy maintenance scheme we propose achieves substantially better resilience against churn as well as correlated failures than the existing (deterministic) lazy maintenance scheme [25] for comparable maintenance overhead averaged over time. Our randomized lazy maintenance scheme exploits the advantage of continuous eager repair strategies by spreading the maintenance effort over time, while still doing only partial repairs (based on randomization) thus enjoying low maintenance overheads.

This chapter is organized as follows: In Section 8.2 we discern the subtle differences in the kind of redundancy one can have for content storage, and the practical implications of these subtle differences. We describe in Section 8.3 the specifics of the maintenance strategies that can be employed to maintain the redundancy of the stored content, and introduce our randomized lazy maintenance scheme. In Section 8.4 we review the various models for analyzing churn and advocate the use of time-evolution analysis - a well established methodology (developed and used in diverse domains including physics and cybernetics [85]) for studying large dynamic systems. In Section 8.5 we describe the specific model of churn which we study in this chapter. In Section 8.6 we perform the time evolution analysis for our randomized lazy maintenance scheme and that of the existing deterministic lazy maintenance scheme. We present our results in Section 8.7 where we show how our randomized lazy maintenance scheme outperforms the existing maintenance scheme. We summarize some ongoing and future work in Section 8.8 before concluding in Section 8.9.

8.2 Redundancy mechanisms: Replication, Erasures and Digital Fountains

Redundancy is essential to fault-tolerance. Moreover, in a peer-to-peer setting characterized by churn, unavailability of any individual peer is more the rule than the exception. Irrespective of the behavior of individual peers, collaborative storage systems endeavor to provide reliable - i.e., highly available and persistent, storage.

Storage redundancy is typically realized by either purely replicating objects (e.g., CFS [46]), or using erasure codes (e.g., RAID [129]). Hybrid strategies in order to improve access efficiency using replication while providing persistence in a memory efficient fashion using erasure codes is also a standard practice (e.g., HP AutoRAID [172]), which has more recently been used in a P2P setting in various systems like Oceanstore [103] and TotalRecall [25].

Erasure codes (e.g., Reed-Solomon codes [142]) have the property that any *M out-of N* fragments can be used to decode and reconstruct an object O. At a storage overhead slightly more than N/M (since in practice size of the object $|O|$ is slightly smaller than size of M fragments) erasure codes provide much better redundancy than what may be achieved using the same storage overhead if pure mirroring (replication) is used [28, 169]. There are however performance trade-offs in actively accessing data [152], and hence erasure coded redundancy is practical only for providing persistence to relatively larger objects in a storage efficient manner, both because of the direct overheads of decoding (reconstructing the object) as well as other practical considerations in such collaborative storage systems - for instance, managing the information about all the fragments and accessing them, among others [168].

Moreover, even though in principle replication is a special case of erasure codes: *1 out-of N*, there is a subtle practical difference typically ignored in the existing analyses [24, 169]. Note that for non-trivial erasure codes, any M *(but) distinct* fragments are required. Thus, if a node goes offline and rejoins, and in the meanwhile the missing fragment has been replenished by the system's maintenance operation, this replica of the erasure coded fragment does not enhance the availability of the whole object.[1] In contrast, in a pure replication based system, obviously the duplicates indeed enhance the availability. In that respect rateless or Digital Fountain (DFs) codes [110, 157] is more like replication. Using DFs lead to generation of random and unique fragments, so that whenever a particular fragment is lost/unavailable, there is neither any need to identify specifically which one fragment is missing, nor is there any risk of having duplicate fragments if and when the missing fragment returns to the system (because of peer rejoins).

The above discussion highlights the subtle differences in redundancy realized by replication, traditional finite rate erasure codes and rateless digital fountains. This has implications on the time-evolution. The current work is a first steop to bridge the gap in existing literature. We'll restrict our analysis to only traditional finite rate erasure codes. For pure replication or DF based redundancy, the same analytical tool can be reused, however the precise details of the analysis will differ, since unlike the case of finite rate erasure codes, using either of pure replication or DFs implies we can potentially have infinite redundancy, even if that's neither necessary nor practical.

While the precise details of the analysis thus depends on the specific properties of the kind of redundancy used, the qualitative results are not expected to be affected. Which is to say, the randomized lazy maintenance scheme we propose will in all cases be more efficient and resilient than the deterministic lazy maintenance scheme.

[1] It potentially can be exploited to enhance the availability of individual fragments but that would lead to higher implementation complexity as well as operational overheads, the benefits of which may be marginal or even detrimental. Or else the duplicate needs to be garbage collected. Whichever be the case, the issue is relevant for system implementation but can be abstracted out in our steady-state analysis, where we use availability of individual distinct fragments (whichsoever way the specific implementation deals with duplicate fragments).

8.3 Maintenance strategies

Since individual participants in a peer-to-peer system can autonomously leave the system - either temporarily or permanently, just storing an object with some redundancy is not sufficient. A minimal redundancy needs to be maintained for that object as long as we'd like it to persist in the system, irrespective of whether the original peers which stored the object (fragments) stay or not. This necessitates a suitable maintenance strategy.

Existing P2P storage systems employ either an eager repair strategy, or a deterministic lazy repair strategy [25], as elaborated below. The existing lazy repair strategy outperforms the eager repair strategy in terms of maintenance cost, however, as we'll subsequently show, this strategy actually is rather vulnerable against churn and correlated failures. We propose a randomized lazy maintenance scheme. For the same redundancy (same M out-of N erasure code) and comparable maintenance cost to the deterministic lazy mechanism, our maintenance strategy provides much better resilience against both regular churn, as well as has better resilience against correlated failures. Next we explain what the existing eager and deterministic lazy maintenance strategies do. Then we introduce our randomized lazy maintenance scheme.

Eager repair: In this strategy the storage system periodically probes for availability of each peer, and replaces any (possibly and most likely temporarily) unavailable data. Such a proactive maintenance mechanism means the system always operates in a state where redundancy level remains constant apart from temporary reduction between repair periods. However as expected intuitively, and has also been observed using simulations by others [25], such a maintenance strategy is very expensive.

Deterministic lazy repair (Strategy-A): The life-time of participants in a peer-to-peer network is often much longer than its session times - that is to say peers often leave the system temporarily only to rejoin back. Consequently, it is not necessary to always replace all fragments which appear to be unavailable (of a stored object) as done in a eager repair strategy, and instead lazier repair strategies can be used - particularly for large objects where periodic repairs is prohibitive. TotalRecall [25] exploits this to propose a lazy repair strategy which we call "*deterministic procrastination*". In order to exploit the returning peers and use lazy maintenance, it is necessary to be able to locate these peers even if they change their physical address. The self-referential directory introduced earlier in Chapter 6 can be used for that.

In the *Deterministic procrastination* approach all peers storing fragments for an object are probed periodically. Repairs are triggered only when a certain threshold T_a of nodes (and corresponding data) becomes unavailable for that specific object. Thus to say, when an object has no more than $T_a > M$ fragments available in the system, then a repair process for the object is initiated so that at the end of the repair process all N fragments are again available. This maintenance strategy is proposed and simulated for the TotalRecall [25] system but the dynamics has not been analyzed. This strategy allows a significant loss of redundancy before triggering many repairs all at the same time. This is undesirable because by waiting before losing a significant amount of redundancy, the system becomes vulnerable to sudden multiple (correlated) failures.

Randomized lazy repair (Strategy-B): An alternative strategy, which we introduce and call *sampling random subsets*, is to probe only a fraction of the stored fragments randomly (uniformly), until a minimal $T_b \geq M$ number of live fragments are detected. Thus a random number $T_b + X$ of probes (determined according to a probability distribution which depends on the actual number of live fragments) will be required to locate T_b live fragments. Then X fragments which were detected to be unavailable are replaced by the system. Note that X can be (and as we'll see from the analysis that it actually is) typically much smaller than the total number of unavailable fragments at that instant.

The advantages of our randomized lazy strategy include:

(i) The repair process is continuous and adaptive, and does not have knee-jerk reactions. When fewer fragments are available, more repairs take place, while when more fragments are available, fewer probing and repair operations are required. Thus this strategy repairs all object all the while a little bit adaptive to the rate of churn, unlike the deterministic procrastination Strategy-A, which repairs objects less frequently, but needs to do a lot of repair work every time it is repairing an object. We'll see in the following that such an approach makes Strategy-A much more vulnerable to both churn and correlated failures.

Note that even though our approach repairs continuously, it does so for only a small subset of unavailable fragments. The rate of repair smoothly adapts to the degree of unavailability. Thus it has the advantage of lazy maintenance strategy - lower maintenance cost, even as it also has the advantage of using the network resources spread over time and providing much better resilience against churn and correlated failures than the existing lazy maintenance approach (for comparable repair cost averaged over time).

(ii) Reduction in the number of probe messages even though cost of probing is not that critical a load on the system.

Obvious extensions of the sampling random subsets based maintenance strategy will include self-tuning the threshold as well as adapting the probing period, but we do not investigate such variants here. Also note that the eager repair strategy is a special case of either of the lazy strategies (corresponding to $T_a = N$ or $T_b = N$).

In principle, if less than M fragments of an object are left in the system, there is no more guarantee that the object will persist in the system. However there are two obvious out of band mechanisms apart the normal maintenance operations. (a) Owner of the object or any other user who has previously used the object and has a local copy may reintroduce it in the system. (b) In any practical implementation of the maintenance operations, the prober(s) for a specific object will keep a local cache of the object in order to optimize the repair process, which can still be used to reintroduce the lost object as well.

However, particularly since we are looking at a scenario with high churn, we ignore these out of band mechanisms, since the original source may be (even permanently) away from the system, as well as over time, different peers will act as the prober of an object since the previous ones might be off-line.

Hence, we will look into only the resilience that is guaranteed by the stand alone maintenance schemes themselves. Implementation issues, particularly which participant of the distributed system is responsible for the maintenance of a specific object is an orthogonal issue.

8.4 Markovian time-evolution analysis

Existing literature on P2P storage systems under churn fail to use a proper analytical tool to objectively determine the performance of the system given a churn and any specific maintenance strategy. Thus, a proper theoretical framework to compare two strategies too is non-existent, leaving no recourse to the P2P storage system researchers apart resorting to simulations. For instance, Bhagwan et. al [25] looks into system designing where they introduced the deterministic lazy maintenance mechanism and evaluate the system performance based on simulation as well as prototyping, however they [25] do not provide any new analytical insight into the system's behavior under continuous churn and repair processes. Similarly Weatherspoon et. al's recent work [168] benchmarks several existing storage systems through simulation experiments for some specific churn levels, but does not delve into analysis of the systems' dynamics. This leaves an important void in the objective understanding of such storage systems' behavior under churn, despite an abundance of empirical results from simulations and prototypes [25, 80, 168].

In order to evaluate our maintenance scheme, following our general approach we identify and develop the right analytical tools based on Markov model of the storage system and looking for a steady-state corresponding to a given churn rate and maintenance strategy, and validate the analytical predictions with simulations.

Churn has been more exhaustively studied in the context of peer-to-peer overlay routing networks using various models as we explained earlier in Section 6.5.1 - (i) Static resilience e.g., Gummadi et. al [75], (ii) Half-life e.g., Liben-Nowell et. al [106] and (iii) Markovian time-evolution analysis, e.g., Aberer et. al [5]. In the following we argue why the later is the only way to comprehensively compare maintenance schemes.

Existing analyses [24, 169] (also reused in [25, 80, 152]) of P2P storage systems at best study the system's static resilience. Weatherspoon et. al [169] look into permanent disk failures as the dominant model for unavailability, and hence completely ignore temporal effects of churn. Thus it is actually Bhagwan et. al [24] who investigate the static resilience of the system. This model essentially looks at a snap-shot of the system, completely ignoring any new failures or repairs. Since this model does not at all look into the repair/maintenance process it is not suitable to compare different maintenance schemes.

As previously argued for studying overlays under churn, we study the storage system's time-evolution particularly looking into its long run *steady state/dynamic equilibrium* behavior under continuous churn for any specific maintenance scheme. If such a steady state exists, then it determines the operational state of the system under the given churn and adopted maintenance strategy, which in turn is necessary to determine the performance vs. operational cost trade-offs in the system.

We evaluate and compare our proposed randomized maintenance scheme with the existing deterministic maintenance scheme by studying the time-evolution of the system for both of these maintenance schemes. We also validate our analysis with simulations.

In that respect, apart proposing a better maintenance scheme, we also employ the time-evolution analysis in the context of P2P storage systems. Having such an analytical model has several other benefits. (a) Wider parameter ranges can be explored accurately much faster than running experiments. (b) Unlike simulation results which are vulnerable to implementation artifacts and whose interpretation is essentially open to speculation, the analysis provides a precise cause-and-effect picture of the system dynamics.

Its worth mentioning that even for the existing empirical studies [25, 80, 168], where the information must have been available, no one looked into the frequency distribution of the system states but only at the mean value. This is possibly a consequence of the fact that without a proper abstraction (as is required for the analysis), even obtainable information has been ignored in the existing literature, simply because it was not well understood as to what to look for and how to use this information.

8.5 Churn model

We model churn according to an exponential lifetime distribution for each online session of any peer as well as the period a peer stays off-line (for a given total peer population). Thus we assume that irrespective of the history at any time instant t, an online node will become unavailable with a probability $\delta_{\downarrow}$ at time $t+1$. Similarly an offline peer will rejoin the system with all its locally stored content at time $t+1$ with probability $\mu_{\uparrow}$. The fraction of available (online) peers is then given by $p_{on} = \frac{\mu_{\uparrow}}{\mu_{\uparrow}+\delta_{\downarrow}}$. Only this average availability is used in existing analysis [24] to study the effect of churn on static resilience in storage systems. Instead we take into account the dynamicity of the system, particularly studying its time evolution using a Markov model.

New peers joining the system will also bring additional storage space (and new objects to be stored), however these peers would not “bring back” the lost fragments, and hence not explicitly accounted for in the analysis. So to say, even if $\mu_{\uparrow} = 0$, the overall network size may stay stable or shrink or even expand, depending on the rate of new peer arrivals. While these new peers do not bring back missing fragments, the continuous maintenance operation will of-course exploit these new peers storage space while replenishing lost redundancy. Similarly peers departing permanently from the network will no more influence the system or its dynamics. Thus the above expression for the parameter p_{on} needs to be understood in the context: roughly speaking, it is the average availability of the existing peers in the network for a mid-term future (in comparison to the period for maintenance operations). Thus, the implicit assumption is that even if all the current peers eventually leave the network in the long run, the maintenance operation will in the meanwhile replace the stored objects at the newly arriving peers. This in turn implicitly assumes that there is actually sufficient storage space in the network. If the network capacity does not increase over time (which is likely since there will be only limited number of nodes, each with storage limitations) but the content volume increases (with proliferation of various devices to produce huge volume of digital content this is also quite likely), the overall storage capacity of the network can become a bottleneck. This is however a more general problem for p2p storage systems and applications designer and is completely orthogonal to the focus of this chapter where we only look at the impact of continuous and simultaneous churn and maintenance operations on the performance, and specifically availability/durability of the stored content. One practical way to deal with exhaustion of storage capacity is to lease storage space for a specific time-span and the storage layer provides availability and persistence of the stored object only for this lease-period, after which the object is gradually garbage collected. The application layer is responsible for lease renewal.

Since the existing maintenance operations (whichever strategy) are invoked periodically, the real time is disentangled from the analysis - more frequent maintenance operations will mean lower perceived churn and vice-versa. Though not done by existing systems like Glacier [80] or TotalRecall [25], one can expect future generations of P2P storage systems to adapt the period of maintenance operations adapted to actual churn conditions, as well as differentiating the importance of various stored objects and various application requirements (while using the same underlying storage system). In these terms, such a disentanglement from the real time provides us the right abstraction, so that the analysis stays generic and only the system parameters $\mu_\uparrow$ and $\delta_\downarrow$ would be different for different scenarios. This approach is similar in spirit to physicists' use of intensive variables (i.e., scale invariant metrics) to study large scale systems, and has also been used in studying the properties of overlay routing networks under churn [59].

There may be some concerns with the churn model we use, but we argue why these are not critical: (i) This model does not look into the effect of permanent departure of nodes from the system nor new peer joins. In the context of storage systems, new peers joining the system do not change availability of already existing objects. But if we assume that the mean session time of a peer is relatively smaller than its life-time in the system, then the availability is threatened more by temporary departures. Since the system's repair mechanism will replenish the lost redundancy, we assume that relatively infrequent permanent departures do not influence the system's availability, particularly since the maintenance scheme will use other peers to compensate for the gradual permanent departures. Such an assumption is justified by measurement studies (such as [23]) on mean life-time and session-time. (ii) In reality, the rate of churn itself varies over time. In such a situation, the system continuously tries to converge to the corresponding steady state. Even then our simplistic analysis continues to provide a holistic insight into the system's behavior - particularly the interplay of churn and maintenance operations and the system's stability and performance for a chosen maintenance strategy. Qualitative comparison of different maintenance schemes hold irrespective of the peculiarities of churn. Particularly, even if churn varies over time, the general relative inferences of lower maintenance cost or better resilience of one strategy over another are expected to hold, and hence such a study provides an objective framework to compare maintenance schemes.

8.6 Analysis: Erasure code based redundancy, lazy maintenance

In order to quantify the performance and compare maintenance schemes, we are typically interested in: *"What is the operational cost of such a system vis-à-vis its resilience?"*

To answer this, we need to better understand the system's dynamics: *"What is the ensemble state of the system (e.g., the time evolution of the probability density function of actual redundancy of the stored objects)?"* and *"Whether it converges to a steady state?"*.

We'll restrict our study to only finite rate erasure code based redundancy (*M out-of N* erasure code). Analysis for other redundancy mechanisms - replication and rateless (Digital Fountain) erasure codes - as well as more sophisticated strategies, e.g., with self-tuning probing periods and repair thresholds, remain as part of our future work, but will rely on the same analysis methodology.

Implicit assumptions: The probing is done according to the maintenance strategy periodically, represented as discrete time t. As previously mentioned, the churn in the system is defined by two parameters: $\delta_{\downarrow}$ and $\mu_{\uparrow}$, representing the probability that an online peer goes off-line or an off-line peer rejoins the system between time t and $t+1$. Fluctuations because of any peer going offline and returning within this period (or vice-versa) is of course transparent to the maintenance operation. Furthermore, we assume that the maintenance operation of different objects is synchronized, and the whole system goes cyclically through two distinct phases: churn and maintenance. Beside the simplification of the analysis, such an approach provides a modular model discerning the separation of concerns in the analysis, such that we obtain two sets of equations: one dependent solely on the churn model, another dependent only on the maintenance strategy, hence making them reusable in the overall analysis when only one aspect of the system (say maintenance strategy) changes. Time for reconstruction of fragment is ignored in our model. This can induce a physical limitation on the repair period which in turn will naturally lead to a minimum amount of churn the system will have to tolerate irrespective of however aggressive a repair strategy one may wish to implement.

Notation and terminology: We say that an object is in state i at any given time if i out of the N erasure encoded fragments of the object are available at that given time. We define $S_i(t)$ as the probability that i out of the N possible fragments of any object are online at time t just after the maintenance operations. $\widetilde{S}_i(t)$ is the probability that i fragments of any object are online at time t just before the maintenance operations (and after churn since time $t-1$). $\sum_i S_i(t) = 1$ and $\sum_i \widetilde{S}_i(t) = 1$ are the standard normalization for probability distributions. We also define $\widehat{S}_i(t) = (S_i(t) + \widetilde{S}_i(t))/2$ as the average of these two states. In reality, since churn is continuous and repair of different objects can not be synchronized, nor is it necessary and in fact undesirable since its better to use the network all the while to maximize its use, we expect that when repairs are indeed not synchronized, the system will actually reside in this state instead of oscillating between the two artificially introduced (before and after churn/maintenance) phases for analytical simplification and modularity. We revert back to this issue while validating our analysis in Section 8.7 (particularly Figures 8.1(c) and 8.1(d)).

Note that the whole system state is defined by these variables (S_i,$\widetilde{S}_i$). It does not matter how the system reaches a specific state till any time t, the system's time evolution from this point can then be modeled as a Markovian process. In particular, a recursive relationship between $S_i(t)$s and $\widetilde{S}_i(t)$s can be defined as follows.

8.6.1 Effect of churn

Irrespective of the maintenance mechanism in use, because of churn (our specific model parameterized by $\delta_\downarrow$ and $\mu_\uparrow$) we obtain the following recursive relationship.

$$\begin{aligned}
\widetilde{S}_i(t+1) \;=\; & S_i(t) \\
& -S_i(t)\left[\sum_{l=0}^{i}\begin{pmatrix} i \\ l \end{pmatrix}\delta_\downarrow^l(1-\delta_\downarrow)^{i-l}\left(\sum_{g=0;g\neq l}^{N-i}\begin{pmatrix} N-i \\ g \end{pmatrix}\mu_\uparrow^g(1-\mu_\uparrow)^{N-i-g}\right)\right] && (8.1)\\
& +\sum_{j=0}^{i}\sum_{l=0}^{j} S_j\begin{pmatrix} j \\ l \end{pmatrix}\begin{pmatrix} N-j \\ g \end{pmatrix}\delta_\downarrow^l(1-\delta_\downarrow)^{j-l}\mu_\uparrow^g(1-\mu_\uparrow)^{N-j-g}\ \textbf{where g=i-j+l} && (8.2)\\
& +\sum_{j=i+1}^{N}\sum_{g=0}^{Min[N-j,i]} S_j\begin{pmatrix} j \\ l \end{pmatrix}\begin{pmatrix} N-j \\ g \end{pmatrix}\delta_\downarrow^l(1-\delta_\downarrow)^{j-l}\mu_\uparrow^g(1-\mu_\uparrow)^{N-j-g}\ \textbf{where l=j-i+g} && (8.3)
\end{aligned}$$

In the equation above, the term (8.1) represents the outflow from state i because of churn. This happens for any object in state i when any l of its i online fragments go off-line, and any $g \neq l$ of its $N-i$ off-line fragments come online. The term (8.2) is the inflow into state i from states $j \leq i$, where the number of fragments for the corresponding state j that go offline (l) and the number of fragments that come back online (g) are mutually related such that $g = i - j + l$. The corresponding object ends up into state i from states $j \leq i$. When $i < j$, similar combinatorial arguments hold - term (8.3). In addition, $i-g = j-l \geq 0 \Rightarrow g \leq i$ and $g \leq N - j$ determine the possible values of the number of fragments coming online (g) from states $j > i$ which can still cause inflow into state i because of simultaneous losses. The corresponding loss l is mutually related to g such that $l = j - i + g$.

8.6.2 Lazy Maintenance Strategy-A: Deterministic Procrastination

We have the following recurrence relationship for the deterministic procrastination based lazy maintenance Strategy-A introduced earlier in Section 8.3. We consider T_a to be the threshold defined in this maintenance strategy.

$$S_N(t+1) = \widetilde{S}_N(t+1) + \sum_{j=M}^{T_a}\widetilde{S}_j(t+1) \tag{8.4}$$

For $T_a \leq i < N$, $S_i(t+1) = \widetilde{S}_i(t+1)$, while for $M \leq i < T_a$ we have $S_i(t+1) = 0$ since if there are more than $M-1$ but less than T_a fragments available, all fragments will be repaired, and for $i < M$ we have $S_i(t+1) = \widetilde{S}_i(t+1)$ since normal repair operations can not reproduce and repair a data with fewer than M of its fragments available in the system. Some out-of-band mechanism to reintroduce the fragments, like by the owner of the object, is beyond the scope of this analysis. Some objects will always go to such states with a positive, even if very small probability. Thus, there is a “leak” in the probability mass, such that eventually all objects will be lost, unless we consider existence of such an out-of-band mechanism. Thus we normalize the probability distribution in each time round, compensating for the small nonetheless finite loss. This normalization process can also be viewed as if the probability distribution function we obtain from the analysis corresponds to the probability distribution corresponding to only the available objects in the system. Despite such a “*trick*”, our analytical model successfully captures the system behavior as validated in subsequent simulations. Particularly refer to the discussions in Sections 8.7.3 and 8.7.5.

The cost of repair operations per object per repair period at a time t is then:

$$C_a^r(t) = \sum_{j=M}^{T_a} \widehat{S}_j(t)(N-j) \tag{8.5}$$

We use $\widehat{S}_i = (S_i + \widetilde{S}_i)/2$ for calculating the cost since in a realistic setting with non-synchronized repair operations for different objects, the system will reside in such a mean state. Though, alternatively one may well use $\widetilde{S}_i$ in order to obtain a more pessimistic estimate of the costs.

Cost of probing per object per repair period is $C_a^p(t) = N$.

8.6.3 Lazy Maintenance Strategy-B: Sampling Random Subsets

For this case we have the following recurrence relationship. We assume T_b as the threshold defined in this strategy.

$$S_N(t+1) = \widetilde{S}_N(t+1) + \sum_{r=1}^{N-T_b} \widetilde{S}_{N-r}(t+1)P_{N-r}(x=r) + \sum_{j=M}^{T_b-1} \widetilde{S}_j(t+1) \tag{8.6}$$

For $T_b \leq i < N$:

$$S_i(t+1) \quad = \quad \widetilde{S}_i(t+1) - \widetilde{S}_i(t+1)\sum_{r=1}^{N-i} P_i(x=r) + \sum_{r=1}^{i-T_b} \widetilde{S}_{i-r}(t+1)P_{i-r}(x=r) \tag{8.7}$$

where $P_i(X=j)$ is the probability that $T_b + j$ fragments are randomly (and sequentially) accessed in order to find T_b available fragments (after which the probing is stopped in that time round), when i out of the possible N fragments are actually available.

$$P_i(X=j) = \binom{j+T_b-1}{j} \frac{i!}{N!} \frac{(N-T_b)!}{(i-T_b)!} \frac{(N-i)!}{(N-i-j)!} \frac{(N-T_b-j)!}{(N-T_b)!} \tag{8.8}$$

This expression comes about because, of the first $T_b + j - 1$ fragments probed exactly $T_b - 1$ must be online (the $T_b + jth$ fragment probed is also online, which is why the probing terminates). They might have been probed in any interleaved sequence along with the j offline fragments probed. There are $\left(\begin{array}{c} i \\ T_b \end{array} \right)$ possible ways of choosing the T_b live fragments out of a total of $\left(\begin{array}{c} N \\ T_b \end{array} \right)$ ways of choosing T_b fragments. Similarly, the j off-line fragments are chosen from the $N - i$ off-line fragments, while they could actually have been chosen from any of the other $N - T_b$ fragments.

For $M \leq i < T_b$ we have $S_i(t+1) = 0$ since if there are more than $M - 1$ but less than T_b fragments available, all fragments will be repaired and for $i < M$ we have $S_i(t+1) = \widetilde{S}_i(t+1)$ since normal repair operations can not reproduce and repair a data with fewer than M of its fragments available in the system.

The cost of repair operations per object per repair period at a time t is then:

$$C_b^r(t) = \sum_{j=M}^{T_b} \widehat{S}_j(t)(N-j) + \sum_{j=T_b+1}^{N-1} \widehat{S}_j(t) \sum_{r=1}^{N-j} rP_j(x=r) \tag{8.9}$$

Cost of probing per object per repair period can be given as[2]:

$$C_b^p(t) \;\; = \;\; N\sum_{i=0}^{T_b} \widehat{S}_i(t) + \sum_{i=T_b+1}^{N} \sum_{j=0}^{N-i} (T_b + j)\widehat{S}_i(t)P_i(x=j) \tag{8.10}$$

8.6.4 Correlated failures

Apart from being resilient to regular and continuous churn, a persistent storage system should also be able to deal with rarer nonetheless inevitable correlated/catastrophic failures.

A way to model correlated failures is to assume that a fraction f_{corr} of the peer population are affected by the correlated failure. However, since the different fragments of the same object are stored at randomly chosen peers, each fragment is lost because of the correlated failure with a probability f_{corr} independently of each other. This model for correlated failure has been used in studying Glacier [80].

In such an event, an object which had i live fragments available before the correlated failure affecting f_{corr} fraction of peers will survive the correlated failure with a probability $\sum_{j=0}^{i-M} \left(\begin{array}{c} i \\ j \end{array} \right) f_{corr}^{j}(1-f_{corr})^{i-j}$. Thus the overall probability for a single object to survive a correlated failure while using a specific lazy maintenance scheme under normal churn is given as:

$$\mathcal{D}_1 \;\; = \;\; \sum_{i=M}^{N} \widehat{S}_i(t) \sum_{j=0}^{i-M} \left(\begin{array}{c} i \\ j \end{array} \right) f_{corr}^{j}(1-f_{corr})^{i-j} \tag{8.11}$$

[2] Note that the actual cost of probing is in any case negligible in comparison to the cost of fragment replacement is thus not critical. Also if more than T_b fragments are available, the cost of probing is $T_b + C_b^r(t)$.

Finally, an end-user of such a storage system storing x objects will be concerned about not only the persistence of a single object with high probability, but also that none of the x objects are lost $\mathcal{D}_x = (\mathcal{D}_1)^x$ assuming that fragments for all the x objects are stored at different peers. Without this assumption, we'd have a higher probability of not losing any of the x objects, however, when objects will be lost, many objects will be lost simultaneously.

8.7 Results

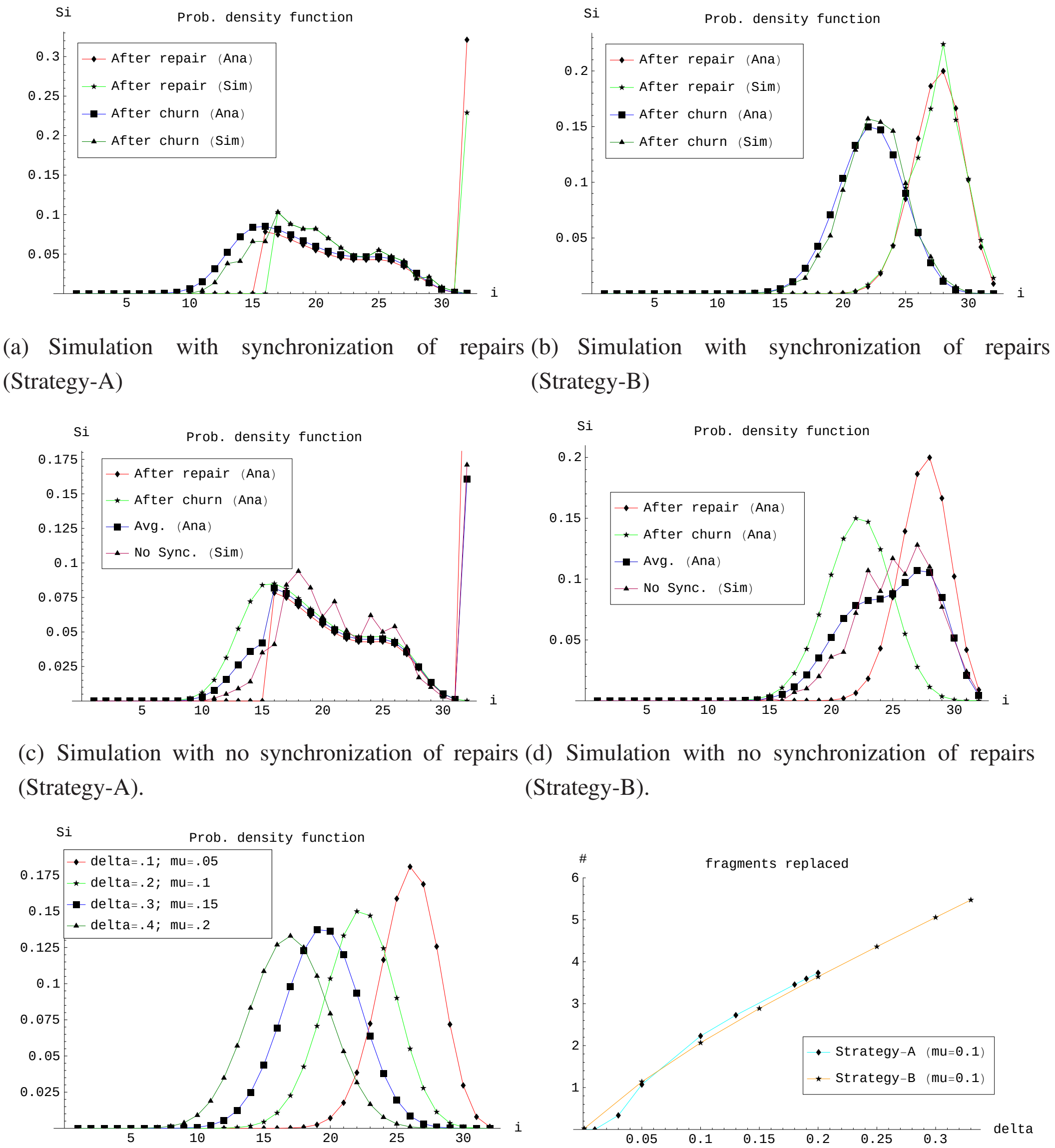
We validate our model with exhaustive simulations and briefly report some of the results here. We compare the two lazy maintenance schemes and observe that the sampling of random subsets based lazy maintenance strategy (Strategy-B) proposed by us has better performance than the deterministic procrastination based existing lazy maintenance strategy (Strategy-A). What is important in the simulations is to respect the independence of object fragments availability. This implies a large enough peer population, but that apart the peer population itself does not play any role. For the statistical properties to hold, we need at least a moderate number of objects to do the averaging across these objects in order to determine an observed distribution function of the states of the objects. Unless otherwise specified, our default experimental setting for the results presented subsequently was as follows:

We considered 200 distinct stored objects. The simulations were run for 200 time units (maintenance cycles) though the steady state is approached within a much shorter time span. Moreover the observed distribution is bound to fluctuate a bit from one time round to another, thus we average the simulation results over a time window of 5 time units. Even for such a small time-window for averaging led to a fairly stable probability distribution over time, demonstrating the low deviation of the system from the steady-state. We use a *8 out-of 32* (rate 0.25) erasure code. We used $T_a = 16$ in *Strategy-A* and $T_b = 12$ in *Strategy-B*. These parameters are chosen such that both the parameters have comparable maintenance cost over time. The experiments were conducted for both settings: synchronized as well as the more realistic non-synchronized repair cycles (but the same periodicity) for different objects. For churn, we typically used $\delta_\downarrow = 0.2$ and $\mu_\uparrow = 0.1$. The results obtained from the simulations matched well with the prediction from our analysis.

8.7.1 Validation of the analytical model

Based both on our analysis (equations solved numerically) and simulations, we observed that the probability distribution functions $S_i(t)$ and $\widetilde{S}_i(t)$ converge over time (and in fact fast) to the steady-state values, demonstrating that all other things being same, particularly the parameters determining the churn, the system indeed converges to a steady operational state[3]. We show some results from our analysis and experiments for deterministic procrastination *Strategy-A* (*Figures 8.1(a),8.1(c) and 8.1(f)*) and sampling random subsets

[3] Thus we'll use only the steady-state distributions S_i and $\widetilde{S}_i$ in the following, without anymore referring to the time t.



(a) Simulation with synchronization of repairs (Strategy-A)

(b) Simulation with synchronization of repairs (Strategy-B)

(c) Simulation with no synchronization of repairs (Strategy-A).

(d) Simulation with no synchronization of repairs (Strategy-B).

(e) $\widetilde{S}_i$ for different $\delta_\downarrow$ and $\mu_\uparrow$ for same $p_{on} = 1/3$ (Strategy-B)

(f) Maintenance overhead (fragments replaced) per maintenance cycle

Fig. 8.1. Simulation based validation of the analytical model (a,b,c,d) and some analytical results (e,f)

Strategy-B (*Figures 8.1(b), 8.1(d)*, 8.1(e) and 8.1(f)). The *x-axis* in the plots corresponds to the states i - the number of available fragments for any object out of the possible N. In Figures 8.1(a) to 8.1(e) the *y-axis* shows the probability mass associated with the corresponding states, just after repair operations are performed S_i, and just before maintenance operations are performed $\widetilde{S}_i$ i.e., after the churn phase. Figure 8.1(e) shows only $\widetilde{S}_i$ for various churn levels. A first setting of our simulation adhered to the analysis model where the repair operations for all objects were synchronized. This led to the two distributions for each state - one after the repair phase and one just before it (Figures 8.1(a) and 8.1(b)), and we see that the analytical prediction of the system state concurs with the experimental results. In practice, such synchronization will not be realistic, and thus the repair operations (replacement of unavailable object fragments) as well as churn (loss or regain of object fragments) will be continuous and randomly interleaved. We simulated the system where there's no synchronization of the repair operations for different objects, and compared it with the result obtained by averaging the two distributions obtained from the analysis. As can be seen from Figures 8.1(c) and 8.1(d), the simulation based result from the model without synchronization of repair operations matched very well with the average obtained from the analytical prediction (averaged).

These results validate that despite the simplifications, particularly with respect to the separation of concern of the effects of churn and repairs, we have an appropriate analytical model capturing the system dynamics.

8.7.2 Static resilience versus steady state analysis

Previously we had noted that $p_{on} = \frac{\mu_\uparrow}{\mu_\uparrow+\delta_\downarrow}$ is the fraction of online peers, and hence corresponds to the average peer availability in the system. In Figure 8.1(e) we show the $\widetilde{S}_i$ analytically obtained for various $\mu_\uparrow$ and $\delta_\downarrow$ but the same $p_{on} = 1/3$ (using maintenance Strategy-B). The more the probability mass shifts leftwards (lower values of i), the more vulnerable the system is. From the figure its clear that even if the peers' average availability is the same, with higher churn (characterized by higher values of $\mu_\uparrow$ and $\delta_\downarrow$), the system is less robust. Such inferences on the system's dynamic resilience is not captured by the static resilience study [24], since it fails to distinguish two systems with same average behavior but different dynamics. The system's actual state in turn has pronounced implications, and the dynamic equilibrium analysis provides us a better glimpse of the system's inner working and hence its actual resilience.

8.7.3 Overheads of lazy maintenance mechanisms

Sampling of random subsets always needs less (or at most same) probes per object as the deterministic procrastination, which always probes for all fragments. However the probing cost is not a dominant cost in the system and hence we do not show it here. The replacement of fragments is however expensive.

In Figure 8.1(f) we show the average number of fragments that are replaced for a churn represented by the parameters $\mu_\uparrow = 0.1$ and varying $\delta_\downarrow$ for the two maintenance schemes. The actual effort in terms of consumed bandwidth and CPU usage will of course depend on the size of the stored objects as well as the particulars of the implementation. In the plot we show for each maintenance strategies only the range of $\delta_{\downarrow}$ for which $\sum_{i=0}^{M-1} \widetilde{S}_i \leq 10^{-4}$. This condition guarantees that the availability of individual objects under the given churn and chosen maintenance strategy stays higher than 0.9999. The choice of the number 10^{-4} is arbitrary, and something else could have been chosen as well. However this number needs to be sufficiently small, both in order to ensure good availability guarantee of stored objects, as well as to ensure that the normalization argument used in Section 8.6 is rational. We'll revert back to this issue also in Section 8.7.5 while explaining results corresponding to Figure 8.3.

From Figure 8.1(f) we can infer two things. First of all, while the two maintenance strategies have very similar average cost of repairing fragments per maintenance round, deterministic procrastination has somewhat lower overheads only at very low churn rate ($\delta_{\downarrow}$) while mostly sampling of random subsets has lower average fragment replacement overhead. In fact the maintenance scheme parameters T_a and T_b for these results were chosen such that the two maintenance schemes have comparable expected repair overheads, so that the resilience achieved can then be compared.

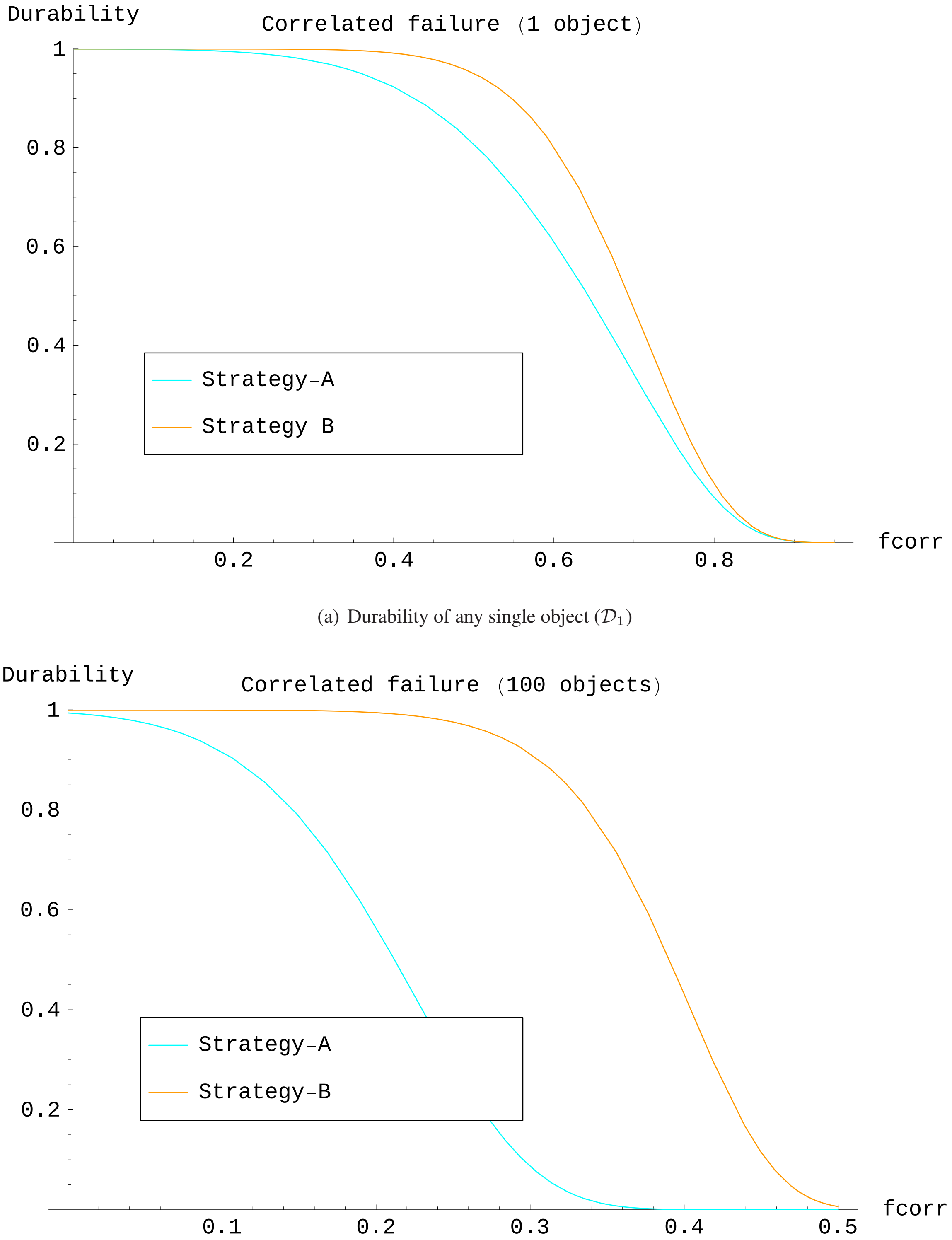
This is because once the churn is high enough, repairs are triggered frequently enough by the deterministic procrastination based scheme, it is just that it does the repairs for each object in impulses - repairing between $N - M$ to $N - T_a$ fragments for the same object in a single maintenance round, because it first lets many fragments to become unavailable. In contrast, the sampling of random subsets based mechanism naturally has to sample and repair more fragments for high churn and fewer for low churn (sample size is determined by the probability distribution as determined in Equation 8.8). Since the repair process for each object is continuous, that is, some of the unavailable fragments are replaced in each maintenance round, the same effort is more evenly distributed over time per object.

A consequence of such a continuous but lower effort per object per round is that almost all objects always retain much better redundancy, that is, the system is "*healthier*" and has a better resilience against churn. Thus for same $\mu_{\uparrow} = 0.1$, Strategy-A guarantees a 0.9999 availability for each object only for $\delta_{\downarrow} \leq 0.2$ while Strategy-B tolerates churn till $\delta_{\downarrow} \leq 0.33$.

Apart from higher resilience against normal churn, this also has implications on the robustness of the system against correlated faults, as is discussed next.

8.7.4 Surviving correlated failures while using lazy repairs

Glacier [80] uses a proactive repair strategy and high redundancy to deal with normal churn. Proactive strategies, particularly for large objects (which is exactly where use of erasure codes make sense) however have prohibitive maintenance cost, which motivated the use of a lazy maintenance scheme in TotalRecall [25]. However, the deterministic procrastination (Strategy-A) used in TotalRecall leaves it very vulnerable to even a small degree of correlated failures, an aspect not accounted for in its design (nor evaluated). In contrast, the



(a) Durability of any single object ($\mathcal{D}_1$)

(b) Durability of all "100" objects ($\mathcal{D}_{100}$)

Fig. 8.2. Durability under correlated failure f_{corr} in addition to regular churn ($\mu_{\uparrow} = 0.1$, $\delta_{\downarrow} = 0.2$)

randomized sampling based maintenance Strategy-B we introduced in this chapter, while having the benefits of being lazy also provides much better resilience against correlated failures. A lazy mechanism can never compete with an eager one in terms of resilience, but still, the randomized lazy maintenance mechanism provides a much a better compromise between maintenance cost and resilience, unlike the deterministic procrastination based scheme which has marginal tolerance against correlated failures. In Figure 8.2 we show the durability of any individual object, as well as the durability of a collection of 100 objects (that is, the probability that none of these 100 objects are lost) for the two lazy maintenance schemes.

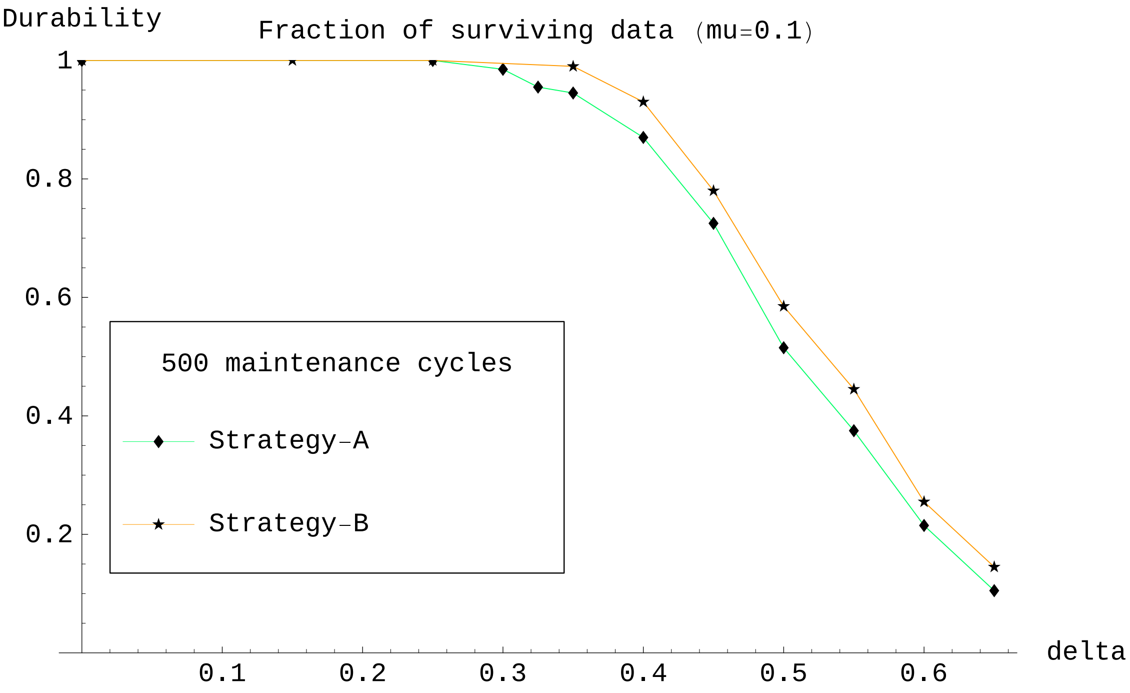
The stark difference in resilience of the two lazy maintenance schemes despite similar repair costs (under regular churn) is readily explained from the the probability distributions $\widehat{S}_i$ corresponding to the two maintenance strategies as observed in Figures 8.1(c) and 8.1(d). Deterministic procrastination allows a large number of objects to concentrate close to the state T_a, while randomized sampling keeps the system far away from the edge even while spending comparable maintenance effort even for a smaller T_b (than T_a) during normal churn. This essentially means that using the randomized sampling based scheme, the system has a much better "*health*" and hence has greater resilience against regular churn as well as occasional correlated failures.

8.7.5 Convergence, uniqueness and stability (of the system) and validity of the model

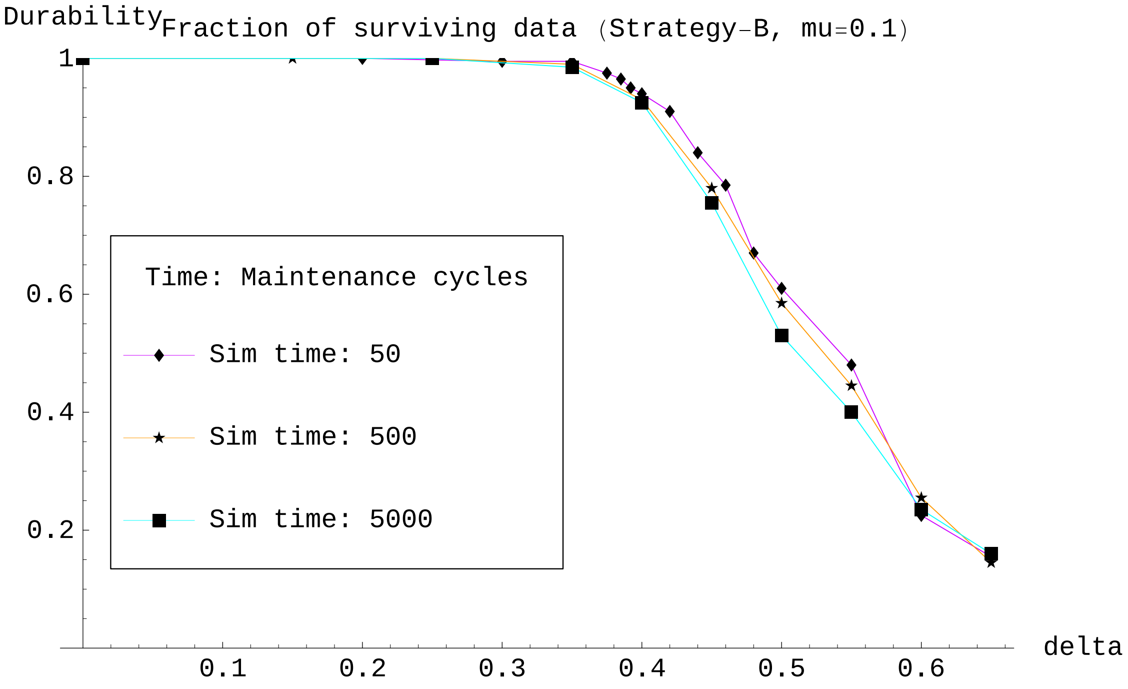
Finally, we show some simulation results to look into the fraction of data that survive normal churn as well as investigate the settings for which our model is appropriate, and when the simplifying assumptions that we made no more hold. The simulations are for $\mu_\uparrow = 0.1$ and varying $\delta_\downarrow$. The x-axis in these plots is $\delta_\downarrow$, and the y-axis is the durability of the objects - fraction of objects which stay available for the whole simulation period. For each setting, the experiments were conducted 5 times, and the worst performance was chosen. Moreover, even as durability and availability are strictly different, we considered the worst availability during the whole simulation as the indicator for the durability under the premise that if a specific object is once unavailable, it may never be recovered using the maintenance schemes themselves (even though some re-joining nodes may make the object available again).

We observe in Figure 8.3(a) that Strategy-B is more robust (tolerates larger $\delta_\downarrow$) than Strategy-A. We also run the experiments for various period, measured in terms of maintenance cycles, to ensure that the system's behavior is correctly captured. In Figure 8.3(b) we show results corresponding to the use of Strategy-B for various simulation durations.

From these results shown in Figure 8.3 we also observe that the system exhibits a threshold (*phase-transition*) behavior at a certain churn value. Beyond this threshold which depends on the maintenance scheme, the system is unstable, and given the chosen maintenance scheme and the churn conditions, stored objects would be lost. The analyses presented earlier in this chapter cease to hold beyond this threshold simply because the transition probabilities and the *normalization trick* (described in Section 8.6.2) does not



(a) Comparison of the two maintenance strategies



(b) Simulation run for 50, 500 and 5000 maintenance cycles

Fig. 8.3. Threshold (phase-transition) behavior observed by simulations.

hold good anymore in practice. Thus to say, we speculate that the same threshold determines the breaking point of the actual system as well as of the analytical model. However, from the perspective of most applications, the region of interest is indeed before the threshold (so that no object is lost) and the analysis provides the exact behavior of the system in this desirable zone of operation. Moreover its easy to estimate the correctness of the analysis based on the obtained result as already also explained in Section 8.7.3. In fact the condition "$\sum_{i=0}^{M-1} \widetilde{S}_i \leq 0.0001$" used there to make sure that the availability of an individual object is more than 0.9999 is a rather conservative estimate (so far as the validity of the analytical model is concerned), as the simulations show that the threshold is close to but somewhat larger than the churn levels considered there. The fact that the threshold observed in the simulations are indeed close also validate our approach of using $\sum_{i=0}^{M-1} \widetilde{S}_i$ as a metric to judge the breaking point of the system, hence making the analytical model itself useful (without having to simulate always to validate results and verify whether the system is stable or not). Both out of academic interest as well as a more precise estimate, we look forward to extend the present theory in order to hopefully analytically derive the threshold point. Determining analytically the precise threshold may require ideas and tools from percolation theory, but admittedly, that is just a speculation at this juncture.

Simulations also showed that starting from various arbitrary initial conditions (where data objects were still available), and given a churn and maintenance scheme, the system always converged to the corresponding unique dynamic equilibrium state. This also means if churn changes over time, the system will try to move from the dynamic equilibrium state to a new one corresponding to the new churn rate.

8.8 Ongoing and future work

As pointed out in the previous section, it'll be interesting to be able to predict precisely and analytically the breaking point of the system. We speculate the need of percolation theory in order to derive the critical churn corresponding to a maintenance strategy. However, as discussed in Section 8.7.5 the current theory already provides (heuristically using $\sum_{i=0}^{M-1} \widetilde{S}_i$) an approximate estimate, which is already a good indicator for systems design.

We hope that as a direct implication of this work, systems like TotalRecall can benefit by using the randomized sampling based maintenance scheme.

In the meanwhile, we are developing a separate storage system (tentatively called Digit4), where we use our randomized subset sampling based lazy maintenance scheme for large objects. That apart, Digit4 uses Digital Fountain (rateless) erasure codes. This has several practical benefits. First of all, we do not need to keep track of which specific fragments are lost and replace precisely the same one, but only need to replace same number of fragments. Digital Fountain codes ensure that these new blocks introduced will be unique with respect to the previously inserted fragments. This also means that even if we do unnecessary repairs and fragments do come back, we'll only have increased redundancy. If finite rate erasure codes are used there will be duplicates of the same fragment (thus not adding any extra diversity), which in fact makes management and garbage collection tasks more complicated. With the use of Digital Fountain codes as our chosen erasure code and using the randomized sampling based maintenance scheme, we thus aim to realize a storage system with good resilience and maintenance cost as well as lower implementation complexity.

8.9 Conclusion

We proposed a randomized lazy repair strategy, which has much better performance in terms of resilience against churn and correlated failures for comparable (and mostly lower) repair costs in comparison to the existing lazy (deterministic procrastination) strategy.

It is relatively simple to determine the static resilience of a system [24], and was an important first step while choosing design parameters for a system. Static resilience however does not properly capture the dynamics of the system, nor give a clear picture of its resilience and performance under continuous churn and repair operations. In fact, as we observed, for the same average system state, the particulars of the dynamics can still greatly influence the system properties. Since static resilience completely ignores the dynamics, it is not useful for comparing different maintenance schemes.

Only studying the time evolution of the system, particularly measuring the probability mass/distribution function of the possible states gives a precise quantification of the system properties. We use Markov time-evolution analysis based on which we observed that the system arrives at a steady state for a given churn. We employed this analysis methodology to do a case study by comparing the two lazy maintenance schemes and evaluate precisely the performance and costs. We also validated these results with simulation experiments. While in real life churn itself varies over time, so that the system will try to move from one corresponding steady-state to another, the steady-state analysis for a given churn still provides an objective framework to understand the interplay of churn and maintenance as well as qualitatively compare maintenance schemes. The quantitative comparison also holds over periods when the level of churn is relatively static.

In particular, from system's perspective, the immediate implication of our work is that the randomized lazy maintenance scheme we proposed here has significant performance benefits and can be easily integrated to existing systems given its simplicity.

9. A push/pull gossiping primitive for unstructured sub-networks

"It is perfectly monstrous the way people go about nowadays saying things against one, behind one's back, that are absolutely and entirely true." — Oscar Wilde

9.1 Introduction

In most peer-to-peer (P2P) systems data is assumed to be rather static and updates occur very infrequently. For application domains beyond mere sharing of static files, for example, sharing files which can be changed over time, trust management [9] or managing ID-to-IP mappings (Chapter 6) or for managing overlay (e.g., P-Grid's structural) replicas storing pointers to actual stored objects at the directory service realized using an overlay, such assumptions do not hold and updates in fact may occur frequently. Other typical applications where new data items are added, deleted, or updated frequently by multiple users are bulletin-board systems, shared calendars or address books, e-commerce catalogues, and project management information.

To improve fault-tolerance and response time data is heavily replicated in most P2P systems and the system must take into account that peers are autonomous and may be offline frequently and that no global knowledge on the system exists.

Thus, the assumption is that the replicas of a specific (set of) object(s) form an unstructured replica subnetwork. The structural replicas of P-Grid is one special case of such replica subnetwork. But the gossiping primitive we study can be used as the mechanism to communicate updates in any arbitrary group of replicas of a wide range of size and having diverse degree of membership dynamics. To meet the challenges imposed by a high replication factor, lack of global knowledge, and peers being online only with a very low probability, we exploit epidemic algorithms under the assumption that probabilistic guarantees instead of strict consistency is sufficient and such an approach can indeed be used in a decentralized and self-organizing environment.

Our proposed update algorithm is based on rumor spreading. We modify existing message flooding algorithms to achieve lower communication overheads but provide similar probabilistic guarantees and low latency. Since we assume that peers are mostly offline, we propose a hybrid push/pull algorithm so that offline peers can inquire for updates that they had missed when they come online again. In the push phase the algorithm uses a new mechanism, apart from traditional feedback and probabilistic methods to propagate a rumor, to avoid many duplicate messages by propagating a partial list of peers to which a particular message has already been sent. It also employs this list in conjunction with the number of duplicate messages received at a particular node as a local metric to estimate the extent to which a message has spread globally, and thereby provides an opportunity to tune the probabilistic parameters of the generic algorithm locally.

We assume logical connectivity of the replica subnetwork. The algorithm is disentangled from the underlying network/physical connectivity. Consequently, the propagation of messages in the physical network and the implementation of applications is an orthogonal issue.Though this work was initially motivated by the need to maintain structural replicas in P-Grid, the algorithm is generic and can be applied for any group of replicas which are mutually connected according to a random graph. Our modifications to existing message flooding algorithms and other results may as well be applied to other search/update algorithms or broadcast/multicast schemes which employ flooding.

Another significant contribution of this chapter is an analytical model of the gossiping algorithm based on a Markov model for the spread of a gossip. Since our algorithm is generic as argued above and subsumes several previous flooding and gossip schemes (for which only experiment or simulation studies existed), the analytical model is valid for these variants and so are the results of our analysis.

9.2 Motivation and problem statement

We look into the communication mechanism required to propagate updates among a relatively large population of individually unreliable peers.

Various global storage systems have been proposed, for example, Freenet [39], OceanStore [148], Pastry [154], and Farsite [29]. Their main goal is to provide distributed storage that scales to very large numbers of users and data sets. Additionally, they may exhibit certain specializations that stem from their intended application domains. For example, Freenet wants to support free speech and anonymity on the Internet, whereas OceanStore focuses on distributed archival storage, which requires special system support.

From the viewpoint of data management these systems should address two critical areas:

1. Efficient, scalable data access which is provided more or less by all approaches, and
2. Updates to the data stored, especially with respect to replication and low online probabilities.

Many of the access schemes are based on some mechanism that associates peers logically with a partition of the search space by means of a distributed, scalable index structure (P-Grid, OceanStore) and use replication to improve responsiveness and fault-tolerance.

Some of the systems support updates. For example, OceanStore, uses classical schemes for updating replicas and assumes high availability of servers, whereas in the systems we envision the peers are fairly unreliable. Our assumptions are:

- Peers have low online probabilities and quorums cannot be assumed.
- Eventual consistency is sufficient.

- Since we do not target database systems update conflicts are rare and their resolution is not necessary in general.
- Probabilistic success guarantees for search are sufficient.
- Consecutive updates are distributed sparsely.
- The required communication overhead is the critical measure for the performance of the approach.
- The typical number of replicas may be substantially higher than assumed normally for distributed databases but substantially lower than the total network size.
- The connectivity among replicas is high and the connectivity graph is random.

The replicas can comprise of structural replicas of the overlay network, which is relatively small in number. The replicas can also be the peers which store a specific object. Depending on popularity or availability requirements for the object, the replica subnetwork size can vary from tens to thousands. This is typically determined at the application layer or end-users.

Statistics from some of the early music file sharing systems show that on average 200 to 250 replicas of same files are available, not counting the replicas that are not shared [167] (but may still be interested in the updates). This requires devising a scheme which scales at least beyond the hundreds to thousands.

If the replica subnetwork is small, each replica may know all the other replicas. But in order to have a communication primitive that can be used in diverse settings we assume that each replica knows only a fraction of the complete set of replicas uniformly randomly, so that the connectivity among them is a random graph.[1] Even when peers potentially know a large fraction of the complete replica population the use of rumor spreading bears a number of advantages as compared to immediately contacting the complete neighborhoods for updates: distribution of update propagation load, reduced delay due to parallel propagation, improved robustness against changes in the peer network and requirement of only partial knowledge of the neighborhood.

9.3 System model

As observed in [127] we assume a very low rate of conflicts. Indeed, many applications, for example, music file sharing or news dissemination, have such a profile where, if data is altered, it may be treated as distinct and coexists as different versions. Similarly, deletions may use conventional tombstones and death certificates. These issues are relatively orthogonal to the communication mechanism used to convey the updates

[1] If not enough replicas are known to a particular node, they can be efficiently discovered. For overlay replicas a peer can query the overlay (P-Grid) starting at a random peer for any key it itself stores, and thus discover a replica peer. The query may get routed to the peer itself with a small but finite probability depending on the replication factor, but this scenario can be easily dealt with, either preemptively by minor modification in the routing message so that the query is not routed to itself, or based on post-processing, and repeating the query. The actual implementation in P-Grid uses the later. For replicas of stored object, if a directory service with pointers to all the replicas of an object is available, then the set of pointers provides the set of replicas. Additionally replicas have the opportunity to get mutually known through the update mechanism discussed in this chapter.

among the replicas. Further, in a decentralized system, such as P-Grid the "data" may indeed be meta-information about the system (peers), e.g., the ID-to-IP mapping of peers (as studied in Chapter 6). Most of these systems operate with a relatively high degree of imperfect knowledge, which is why probabilistic guarantee of information dissemination in such application scenarios is sufficient.

We assume a decentralized setting, i.e., all peers are equal and no specialized infrastructure, e.g., hierarchy, exists. No peer has a global view of the system but base their behavior on local knowledge, i.e., its routing tables, replica list, etc. The peers can go offline at any time according to a random process that models the behavior when peers are online. Physical connectivity and topology are ignored which can provide opportunities for optimization which we admittedly overlook in the gossiping algorithm we propose for analytical simplicity. In an actual implementation exploiting some of such information heuristically is relatively straightforward, e.g., favoring replicas from the same ISP when making a random choice, or using the same subset of replicas with which the peer had recently successfully communicated. We also assume that if two peers are online a communication channel may be established between them. This assumption does not have any critical impact, since if two peers may not communicate with each other, they will simply perceive each other to be offline. It is primarily the erratic behavior of online availability and the lack of global knowledge, as well as the absence of any centralization, which prompts us to call this environment unreliable. Potentially limited resources, particularly bandwidth (and power in wireless/mobile environments), and the varying degree for tolerance of latency makes the environment even more challenging.

Our update propagation scheme has a push phase and a pull phase which are logically consecutive but may overlap in time. A new update is pushed by the initiator to a subset of replica peers it knows, which in turn propagate it to replica peers they know similar to a constrained flooding scheme. In our analysis of the push phase in the next section we assume a synchronous model which is a standard assumption for analyzing epidemic algorithms [96].

Peers that have been disconnected (offline, disruption of communication) and get connected again, peers that do not receive updates for a long time (locally determined), or peers that receive a pull request, but are not sure to have the latest update, enter the pull phase to synchronize and reconcile. The pull scheme is similar to anti-entropy [54], in the sense that the pulling party tries to synchronize itself with the pulled party. Since the pulled party itself may be out of sync, it is preferable to contact multiple peers and choose the most up to date peer(s) among them.

Push phase of the update algorithm:. When a peer p receives an update request (U, V, R_f, t) from a peer f, where U is the updated data item, V its version,[2] t is a counter which counts the number of push rounds that

[2] This actually is a vector of version identifiers of the form $(VersionId_1, VersionId_2, \ldots, VersionId_n)$. Version identifiers are universally unique identifiers computed locally by applying a cryptographically secure hash function to the concatenated values of the current date and time, the current IP address and a large random number. Also, depending on the type of information being updated, the update message may be somewhat different, e.g., for updating ID-to-IP mappings we used $(E_p, addr_p, TS_p, D_p(E_p, addr_p, TS_p))$ in Chapter 6.

have already been executed for the update, it also receives a partial (flooding) list R_f, to which the same update has been sent (not necessarily received by all peers in R_f). Then p chooses a random set R_p of its known replica peers and forwards the request $(U, V, R_f \cup R_p, t + 1)$ with a probability $PF(t)$ to the set $R_p \backslash R_f$. $PF(t)$ can be any function, and is potentially a self-tuning parameter to be determined locally by p. Another benefit of propagating R_f is that p possibly discovers replicas unknown to her.

Algorithm 7 Push phase at replica p upon receiving $Push(U, V, R_f, t)$

```
if $ProcessedUpdate(U, V, R_f, t) == FALSE$ then
    Select a random subset $Rp$ of replicas with $|Rp| = R * f_r$;
    With probability $PF(t)$: $Push(U, V, RfunionRp, t + 1)$ to $Rp$ $Rf$;{PF(t): deterministic or self-tuning function}
    $ProcessedUpdate(U, V, R_f, t) = TRUE$;
end if
```

Since any replica pushes the update at most once (to multiple replicas[3]), the local termination decision is trivial. The number of push rounds gives the latency of propagating the update to all online replicas.

Pull phase of the update algorithm:. When a peer gets connected again because it was offline or suffered from a communication disruption, received no update for some time, or receives a pull request but is not sure whether it is in sync, then it enters the pull phase and inquires for missed updates, e.g., based on version vectors.

Note that peers use logical identifiers to communicate with each other. Over sessions, peers may change their physical address. To be able to communicate with such peers it is essential to locate their latest physical address. While the push phase is relatively insensitive to communication failure, the pull phase is more sensitive. In any case, over a period of time, such information need to be refreshed at peers. We assume in this work that peers refresh the physical address of other replica peers in the gossip network using an extrinsic background process, which may be realized by querying the self-referential directory introduced earlier in Chapter 6.

9.4 Analysis

9.4.1 Setup and notation for the analysis

The goal of our update algorithm is not to achieve complete consistency but rather to know what is the probability of a correct answer given certain model parameters for the gossiping scheme. We assume that every peer knows a subset of all replicas that replicate the same data. We consider the replica network to be a

[3] Such a one time push model also makes sense for geographically constrained systems like sensor networks etc. where the neighbors stay the same any way, but the connectivity is not random graph in such network, so the results are not directly applicable.

small P2P network itself but with no specific internal structuring. It handles updates/requests for a partition of the key-space.

In the analysis we start from a completely consistent state, analyse a single update request, and evaluate the number of messages and time (rounds of message exchange) required to reach a consistent state again. Since most of the replicas are offline most of the time, our notion of consistent state is more related to the online population R_{on}^{τ} at a given time τ rather than the whole set of replicas $\Re$. Though our analysis is generic, we evaluate the algorithm for realistic scenarios: availability of the peers to be a random process with expected value of being online between 10% to 30%. The replication factor is assumed to be between 100 to 1000 and though scalability might not be a major issue for such moderately small numbers, larger replication factors too have been investigated. Table 9.1 shows the notation used in our analysis.

Notation	Explanation
R	Cardinality of the set of replicas $\Re$
t	Number of the push round for a particular update
U	The update message or its size (notation depends on the context)
$ML(t)$	Size of messages in round t
$L(t)$	Normalised size of the partial list of replicas which have the update in round t. This is equal to the number of entries in the list divided by R.
$R_{on}(t)$	Number of replicas online in round t
σ	Probability that a peer stays online in the next push round
f_r	Fraction of replicas to which peers initially decide to forward the update message
$newreplicas(t)$	Number of new replicas receiving update in round t
$msg(t)$	Number of messages in round t, including messages to offline replicas
$f_{\triangle aware}(t)$	Increment in fraction of online replicas which are aware of the update after round t
$f_{aware}(t)$	Total fraction of online replicas which are aware of the update at the beginning of round t.
$P_F(t)$	Probability that a peer pushes an update in round t if it received it in round $t-1$.
B	Size of data required to describe one replica (e.g., 10 bytes).

Table 9.1. Notation used in the analysis

When an update U is initiated for a set $\Re$ of replicas with cardinality R, in general the online population in push round t will be $R_{on}(t) = R_{on}(t-1)*\sigma+[R-R_{on}(t-1)]*\epsilon_2$ where $\sigma = 1-\epsilon_1$ and $R_{on}(0) = R_{on}^{\tau}$ if the update starts at time τ. ϵ_1 is the probability of an online peer going offline in one push round and ϵ_2 is the probability of an offline peer coming online in a push round. These values are typically small and may vary in different push rounds. For the sake of simplification, we will initially ignore the effect of replicas coming online, and will further assume a constant σ, hence we have $R_{on}(t) = R_{on}(t-1)*\sigma$. Neglecting the effect of positive ϵ_2 is justified because peers coming online need to execute pull any way, and thus do not contribute to the push phase. Even if some peers come online during a push phase and receive update through push, they will not contribute to the push phase and thus not make any difference to the whole system's behavior or the analysis of the same. The assumption of a very small ϵ_1 is justified because a single push round will take a very small time (network delay for a single message), and unless there is any kind of catastrophic failure, a very small number of peers will suddenly decide to go offline. Having said that, we still evaluate the performance of the push phase for rather wide range of values of σ to study the robustness of the gossip algorithm. Further, we choose a discrete time model for the rumor spreading algorithm, just like most other rumor algorithms. This in itself does not mean that we need synchronous rounds. It is indeed possible that because of variation in network latency, messages of different push rounds live in the network at the same instant of time. Instead of treating t strictly as time, it needs to be interpreted as the round number, and the replicas which get infected by that round are effectively replicas that eventually gets updates from this round. Thus t does not in itself define an ordering of receiving updates among all peers in the system.

Typically the parameters, such as f_r, σ, R, $R_{on}(0)$, may vary over time. But for the purpose of analysis we may assume that they remain constant throughout a single update push phase. In Section 9.6 we will give some indication of how the parameters can adapt over time to the varying network conditions.

The choice of two parameters P_F (probability of forwarding an update) and f_r (fraction of total replicas to which peers initially decide to forward an update) rather than defining only one parameter which couples both of them together is because we wanted to study the effects of both these factors in limited flooding algorithms. For example, a protocol like Gnutella [40] uses flooding with a fixed fanout, but uses no notion of P_F. Actually its use of time-to-live (TTL) for messages effectively means that P_F is 1 for TTL rounds, and 0 after that. Some other systems, for example one using gossip for ad-hoc routing [79], on the other hand uses probability of forwarding rumors as a design parameter. In order for our analysis to be general enough, such that all these variations of limited flooding can be reduced to special cases of our model, we included the notion of both fanout and probability of forwarding.

9.4.2 Analysis of the push phase

The expected number of replica peers which receive an update in a given round depends on the number of newly infected replicas in the previous round, as does the total coverage. Thus the spread of the gossip in the push round can be modeled to have the Markov property. The absorbing state corresponds to peers who have already received and gossiped the rumor, and the absorbing state is when all (online) peers have been reached by the gossip.

Round 0.

The replica initiating the update propagation sends U to f_r fraction of replicas. Thus we obtain a total number of messages, $msg(0) = R * f_r$ (including messages to offline replicas). The number of new replicas which receive the update is $newreplicas(0) = R_{on}(0)f_r$. The number of online replicas without update is $R_{on}(0)(1 - f_r)$. The message length in this round is $ML(0) = U + R * B * f_r$

Round 1.

Assuming message flooding, where every replica which received an update message decides with probability $P_F(1)$ to forward it to $R * f_r$ replicas, we have:

$$msg(1) = R_{on}(0)\sigma P_F(1) R f_r^2 (1 - f_r) \tag{9.1}$$

The expression may be explained as follows. $R_{on}(0)f_r$ of the online population received the update in the previous round, a fraction σ of these replicas continue to stay online in the present round, a $P_F(1)$ fraction of these replicas decide to forward the message. Each of the $R_{on}(0)f_r\sigma P_F(1)$ peers decide to push the update, forwarding it to $R(f_r - f_r^2)$ replicas, since it knows that the update has already been sent to f_r^2 of the f_r fraction of randomly chosen replicas. Actually, in case a replica receives update information from more than one replica, it can use the list of 'updated replicas' in each of those messages, and hence the number of messages can be further trimmed, at an additional computational cost.

$$\begin{aligned} newreplicas(1) &= R_{on}(0)\sigma(1 - f_r) * \\ &\quad [1 - (1 - f_r)^{R_{on}(0) f_r \sigma P_F(1)}] \end{aligned} \tag{9.2}$$

The expression may be explained as follows: Of the $R_{on}(0)\sigma(1 - f_r)$ uninformed online peers, a fraction $(1 - f_r)^{R_{on}(0) f_r \sigma P_F(1)}$ peers continue to stay uninformed when each of the $R_{on}(0)f_r\sigma P_F(1)$ informed peers forward (push) to f_r fraction of random peers. The others receive the update after this round. For the message length we have:

$$\begin{aligned} ML(1) &= U + R * B * (f_r + f_r(1 - f_r)) \\ &= U + R * B * (1 - (1 - f_r)^2) \end{aligned} \tag{9.3}$$

Round $t \geq 2$.

The results may be generalized as follows:

$$\begin{aligned} newreplicas(t) &= R_{on}(t-1)(1 - f_{aware}(t-1))\sigma * \\ &\quad (1 - (1 - f_r)^{R_{on}(t-1) f_{\triangle aware}(t-1) \sigma P_F(t)}) \end{aligned} \tag{9.4}$$

Thus we obtain the fraction $f_{\triangle aware}(t)$ and $f_{aware}(t)$ as:

$$\begin{aligned} f_{\triangle aware}(t) &= (1 - f_{aware}(t)) * \\ &\quad (1 - (1 - f_r)^{R_{on}(t-1) f_{\triangle aware}(t-1) \sigma P_F(t)}) \end{aligned} \tag{9.5}$$

Then,

$$\begin{aligned} f_{aware}(t) &= f_{aware}(t-1) + f_{\triangle aware}(t-1) \\ &= 1 - (1 - f_{aware}(t-1)) * \\ &\quad (1 - f_r)^{R_{on}(t-2) f_{\triangle aware}(t-2) \sigma P_F(t-1)} \end{aligned} \tag{9.6}$$

Note that this is a recursive relationship and f_{aware} rapidly grows to 1. The expression for f_{aware} may exceed the value of 1, but that will have no physical relevance, and thus the function needs to be determined using a ceiling function and $f_{\triangle aware}$ too needs to be reevaluated accordingly in the final push round. Also note that $P_F(t)$ can be any arbitrary function of t, which individual nodes can define in an ad-hoc manner, and we will see that this parameter can be tuned for the push phase in order to significantly reduce duplicate messages.

It is subtle to determine the number of messages and length of these messages. If the partial list of replicas, to which the update has already been transmitted along with the update information U, is ignored, we have

$$msg(t) = R_{on}(t-1) f_{\triangle aware}(t-1) \sigma P_F(t) R f_r \tag{9.7}$$

since each of $R_{on}(t-1) f_{\triangle aware}(t-1) \sigma P_F(t)$ replicas (these replicas received the update in the previous round, and continued to stay online, and decided to forward the same) forward the update to Rf_r replicas. If the partial list of replicas is accounted for, then the number of messages decrease to

$$msg(t) = R_{on}(t-1) f_{\triangle aware}(t-1) \sigma P_F(t) * R f_r (1 - f_r)^t \tag{9.8}$$

and the length of each message in round t is given as

$$ML(t) = U + R * B * (1 - (1 - f_r)^{t+1}) \tag{9.9}$$

We now prove the two equations above by induction. Let the normalized length of the partial list of replicas in a message be denoted by $L(t)$. The normalized length of the partial list is the fraction of the total replicas that the partial list contains. Then $ML(t) = U + R * B * L(t)$.

Induction hypothesis: $L(t) = 1 - (1 - f_r)^{t+1}$

Now $L(t+1) = f_r + L(t) - f_r L(t)$, since the Rf_r replicas chosen randomly are independent of the replicas in the partial list. Now, if our hypothesis is true then,

$$\begin{aligned} L(t+1) &= f_r + 1 - (1-f_r)^{t+1} - f_r(1-(1-f_r)^{t+1}) \\ &= 1-(1-f_r)^{t+1} + f_r(1-f_r)^{t+1} \\ &= 1-(1-f_r)^{t+2} \end{aligned} \tag{9.10}$$

Thus the hypothesis is consistent. Since the hypothesis is true for $t = 0, 1$, using induction, we conclude that $L(t) = 1 - (1 - f_r)^{t+1}$ where $ML(t) = U + R * B * L(t)$. Thus,

$$\begin{aligned} msg(t) &= R_{on}(t-1)f_{\triangle aware}(t-1)\sigma P_F(t)R * \\ & \quad f_r(1-L(t-1)) \\ &= R_{on}(t-1)f_{\triangle aware}(t-1)\sigma P_F(t)Rf_r(1-f_r)^t \end{aligned} \tag{9.11}$$

As may be observed, $L(t)$ increases with round number t and a legitimate question to ask is its effect on the resource (Memory/CPU/Bandwidth/Power) available at each of the replicas. A way to deal with increasing $L(t)$ may be to chose a normalized threshold length $L_{max}(t)$ such that $L(t) = min(L_{max}(t), L(t)^*)$ where $L(t)^* = L(t-1) + f_r - L(t-1)f_r$. This can be achieved by discarding either random entries or the head or tail of the partial list. In this case, $msg(t+1) = R_{on}(t)f_{\triangle aware}(t)\sigma P_F(t+1)Rf_r(1 - L_{max}(t))$. $f_{\triangle aware}$ and f_{aware} stay unchanged, since the extra messages generated by reducing the $L(t)$ are all duplicate messages. Thus the nodes which push the update in the later rounds pay the penalty of forwarding extra messages without enhancing the coverage.

Note that the case where $L_{max}(t)$ is zero for all replicas corresponds to the case where no list is propagated, and will enhance the number of duplicate messages, without any improvement in coverage of unreached replicas.

9.4.3 Analysis of the pull phase

If a replica p comes online at a random time (after the push phase is over), then it will (very likely) find the update information from any of its online replicas. The underlying assumption for such an optimism is that any replica that came online in the meantime must have pulled the update information by the time the concerned peer p came online. This justifies the eagerness of the Update Pull algorithm.

What is more interesting is what happens if p comes online while a push of an update is underway. If f_{aware} fraction of the replicas R_{on} are already aware of the update, the probability of a replica p getting the update in a attempts is

$$1 - [1 - (R_{on}f_{aware}/R)]^a \tag{9.12}$$

which implies that a constant number of pull attempts should give the update information with high probability. Since updates are propagating by push as well, the above term gives a worst case estimate.

Indeed if $f_{\triangle aware}$ (refer to push phase analysis) fraction of online replicas received updates in the previous push round ($t-1$), then (if they continue pushing) the probability of getting a push is

$$1-(1-f_r(1-L(t)))^{R_{on}(t-1)f_{\triangle aware}(t-1)\sigma P_F(t)} \tag{9.13}$$

9.4.4 Query (request)

Servicing requests under (possibly relatively frequent) updates is similar to the Pull phase of updates. For simple servicing of requests, we may indeed use the same analysis as in the Pull section. Since requests are more sensitive (updates can be lazy, and strong consistency is not our goal, however we intend to return correct and most recent result for any query) we may define some majority logic, or use a version scheme for identifying latest updates, or a hybrid of the two. The specifics can depend on the application requirement - for example, for ID-to-IP mapping information, only the owner can make a change, and can strictly order these updates (using a local timestamp) and this timestamp can be used as the basis to choose the latest information.

9.5 Analytical results

Based on the analytical model developed in the previous section we investigated for various environmental parameters the performance of the push phase of the propagation of a single update. For the evaluation of the recursive analytical functions a C-program had been developed.

Our performance criterion for this analysis is primarily the number of messages that are generated as part of a single update, compared to the extent to which the update propagates among the online population. As a simplifying (and for IP networks, realistic) assumption we ignore message size, as a single or very few messages can accommodate the messages of maximal size that can occur in our setting, alternatively we can terminate the piggy-backed meta-information based on the IP packet-size constraints.

In the following result plots (e.g., Fig. 9.1) we will show on the y-axis the number of messages generated per member of the initial online population. As assumed in the previous section peers coming online subsequently are not participating in the propagation. Ignoring the fact that peers may go offline throughout the push phase makes this estimate more pessimistic. On the other hand since other approaches do not account for peers going offline, we chose the simple metric of comparing to the initial population size, in order to enable comparisons to related approaches. On the x-axis we will give the percentage of the online peers that have become aware of the update. Since the analysis is made in rounds the plot is discrete, and the marks (points) on the curves indicate the discrete steps. From the number of points on the curves it can be seen how fast the rumor spreads (latency), but our main interest is the communication cost involved in updating all online peers.

9.5.1 Impact of the initial online population size

In this analysis we studied the impact of varying the initial online population for the plain flooding scheme. If the initial population size is too small as compared to the total population, the probability that a peer to which a message is sent is available is too low, and the rumor will not spread. Varying initial online replicas $R_{on}(0)$ between 1 to 100% it is observed in Fig. 9.1(a) that without a significant initial online population($< 5\%$), it is difficult to make all online peers aware of an update. In case there is a significant initial online population, the message overhead is relatively independent of the online population, as seen in the Fig. 9.1(b) for a variation from 5-100% of total population. However, message overhead is very high for this plain flooding scheme, around 80 messages per online peer.

9.5.2 Impact of varying fanout (f_r)

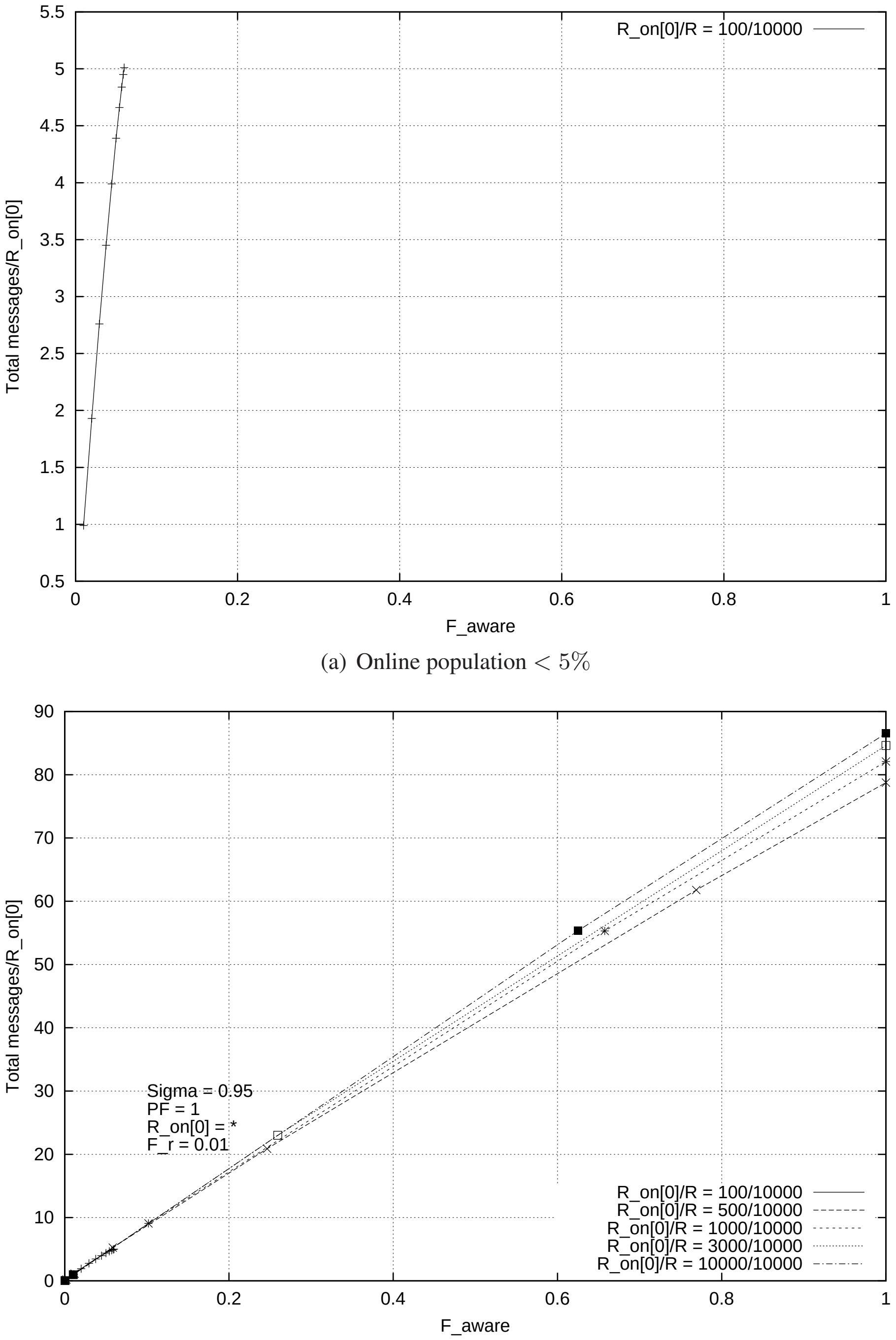
Since flooding is exponential in nature, a limited fanout is sufficient to spread the update to a complete population. A large fanout will cause unnecessary duplicate messages. Varying f_r it is inferred in Fig. 9.2 that the intuitive expectations are true, and it is not necessary to push to too many replicas, since it does not significantly enhance the update propagation, however creates eight to ten times more duplicate messages. Thus it is sufficient, and indeed desirable to have a small fanout.

9.5.3 Impact of departing peers (σ)

Even if the environmental parameter σ (probability of online peers staying online in consecutive push rounds) varies, and is quite low, Fig. 9.3 demonstrates that the algorithm is quite robust to replicas going offline (without forwarding the update) after receiving the update. Indeed, typically σ will be larger than 0.95. We investigated lower values of σ, because curiously the message overhead decreases significantly if several replicas 'fail' to forward the update. This was an additional reason that prompted us to introduce $P_F(t)$ in our analysis, and is discussed next.

9.5.4 Impact of probability of forwarding ($P_F(t)$)

With the progress of the push rounds, a large population will become aware of the update (exponential growth initially), and a very small population will be left unaware. Consecutively, if all newly aware peers decide to continue gossiping, a large number of messages are generated, for a small target audience. Thus even if a small fraction of the newly aware peers gossip, it is sufficient to reach out all uninformed peers, and using a substantially lower number of messages. Fig. 9.4 indicates that the best strategy is to reduce the probability of forwarding updates with the increase in number of push rounds, which eliminates many unnecessary messages. On the downside, it is essential to properly tune $P_F(t)$, lest the update is not propagated to the whole population. We will briefly describe tuning of $P_F(t)$ in Section 9.6 for optimizations and self-tuning of parameters in a decentralized manner using only local information.



(a) Online population $< 5\%$

(b) Significant online population ($> 5\%$)

Fig. 9.1. Results with varying initial online replicas $R_{on}(0)$ between 1 to 100%

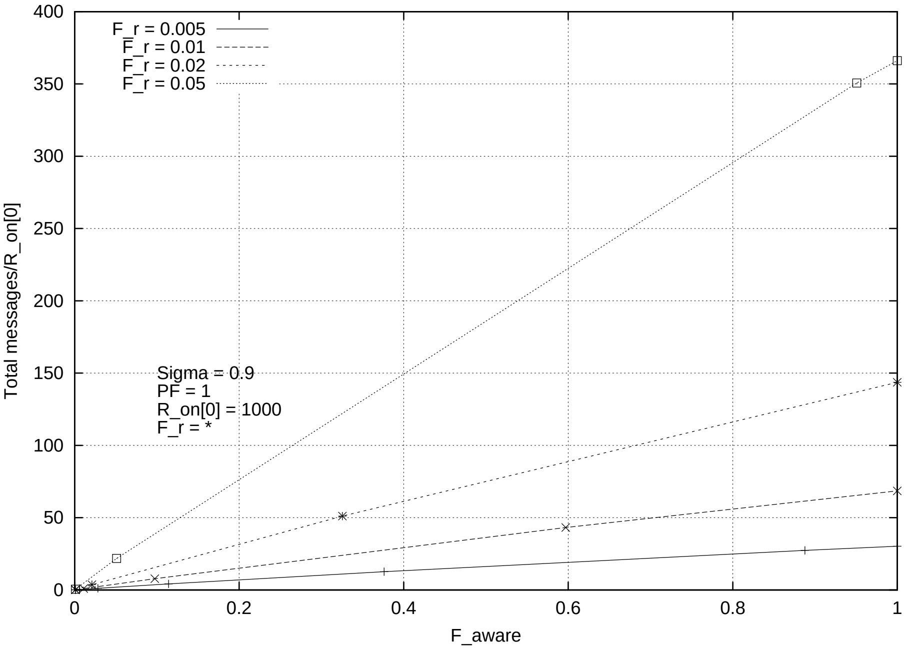


Fig. 9.2. Impact of varying fanout (f_r)

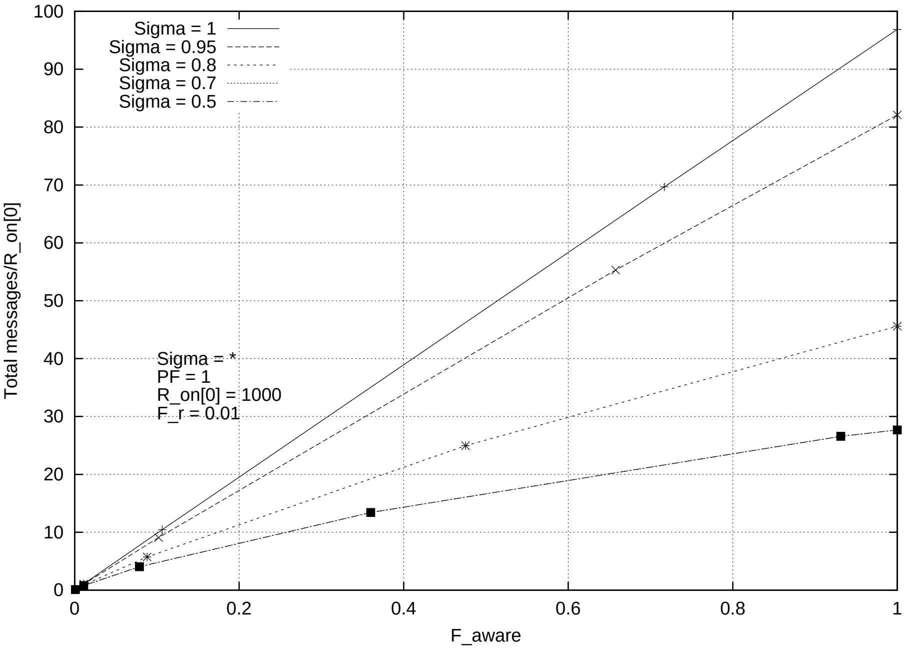


Fig. 9.3. Impact of departing peers (σ)

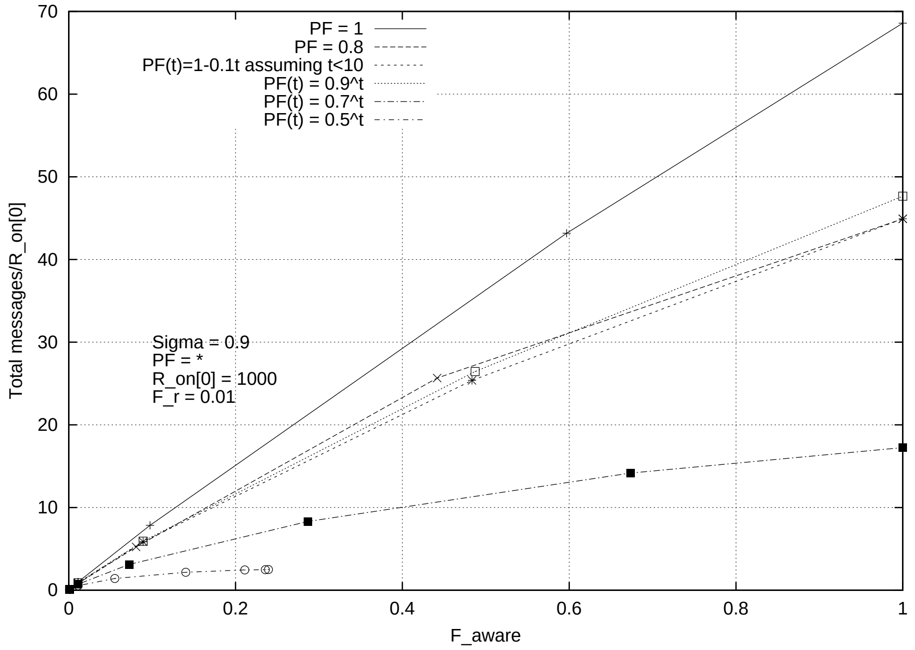


Fig. 9.4. Impact of probability of forwarding ($P_F(t)$)

9.5.5 Scalability

As stated previously, scalability has not been our principle concern with replication factor between 100-1000, but our push scheme also scales well, as observed for a total population varied between 10^4 to 10^8 with $R_{on}/R = 0.1, \sigma = 1, P_F(t) = 0.8 * 0.7^t + 0.2$ and f_r chosen such that a message is sent to ten online peers, i.e., $R_{on} * f_r = 10$. The results are shown in Fig. 9.5. As may be observed the total number of messages per initially online peer has a decently low value. With the increase in total population, the number of messages per online peer is decreasing, for all other parameters kept fixed. For a small population, we do not need a fanout of ten online peers, and choosing a smaller fanout increases the number of push rounds but decreases the message overhead as shown in Section 9.5.2.

Thus we conclude that for a very large range of total population, the message overhead can be, with proper choice of fanout and probability of continuing the push, limited to around 20 messages per initial online peer. Given the fact that this is so when there is no knowledge as to which replicas are actually online, and thus the best that can be done is to use on an average ten messages (for $R_{on}/R = 0.1$), we conclude that our simple (look and implementation wise) push algorithm is quite robust, as well as scalable.

9.5.6 Comparison with simple flooding (like in Gnutella) and variants

Since our Push phase algorithm uses Gnutella-like limited message broadcast (flooding with some specific fan out), which is known to have scalability problems [151, 162], it is imperative to point out the im-

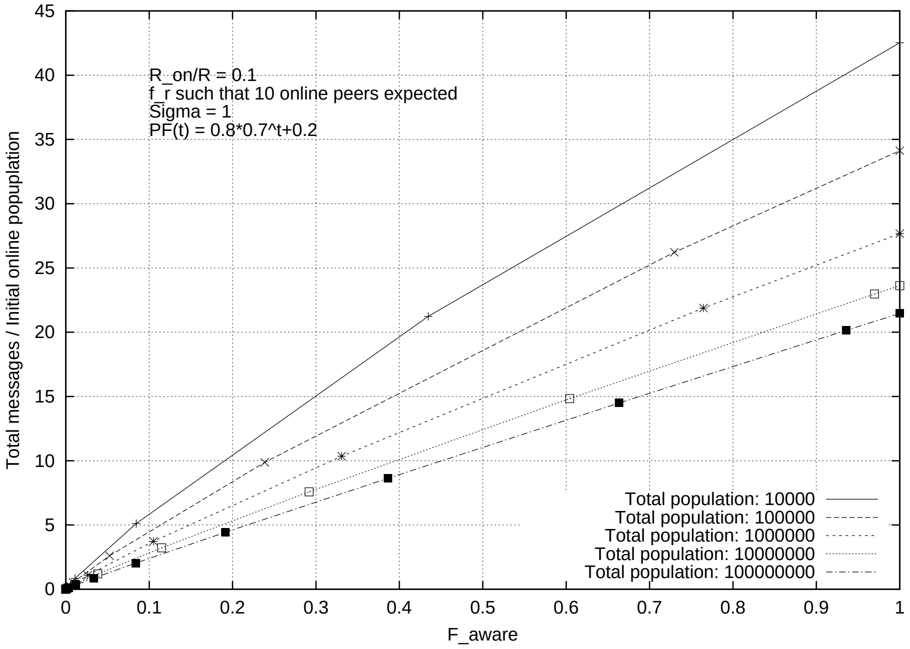


Fig. 9.5. Scalability of the push algorithm

provement achieved by very minor changes, since similar modifications may be made even in the Gnutella message flooding schemes to make it more efficient.

Though flooding (Gnutella) has been amply analysed by many researchers and music file sharing enthusiasts, the ping and pong messages required to establish such a connectivity/neighbourhood are mostly ignored, which makes Gnutella worse. Assume a random distribution for the replicas to stay online, with a probability p_{on}. We have $avg(R_{on}) = p_{on}R$. Then the expected number of peers that are reached in A attempts when actually K replicas are online is $(K * A)/R$. Thus the expected number of attempts to reach S online replicas $E_S(A) = \sum_S^R \frac{R*S*P(K)}{K}$. Assuming peers stay online according to a Poisson process, i.e. $P(K) = \frac{e^{-Rp_{on}}(Rp_{on})^K}{K!}$ we have

$E_S(A) \approx S/p_{on}[1 - exp(-R * p_{on}) \sum_0^S (R * p_{on})^K / K!]$

We then use $f_r = E_S(A)/R$. Then the expected messages required in pure flooding (without duplicate avoidance) may be obtained from the geometric sum $1 + (R * f_r) + (R * f_r)^2 + ... + (R * f_r)^{RequiredPushRounds-1}$. In the case of Gnutella like duplicate avoidance, the total number of messages created per update will be exactly the average fanout multiplied by number of peers online, that is to say, there will be on an average f_r messages per online peer and the propagation of update will incur the same latency as in the case if flooding without duplicate avoidance, since duplicate avoidance only reduces the number of redundant messages without any effect on the spread of the update itself.

A variant of pure flooding has been proposed by Haas et.al. [79] called $G(p,t)$ for the "Ad-hoc On Demand Distance Vector (AODV)" routing algorithm. There, for the first t rounds it follows a pure flooding, while in the next rounds $t' > t$, each node decides to continue flooding with a probability p. Simulation results have shown that such an approach reduces message overhead by a quarter to a third, as compared to pure flooding. Since this scheme is strictly a special case of our algorithm, it is obvious that with proper parameter choices out algorithm will perform at least as well. In Table 9.2 we summarize the comparison in terms of total messages per initially online peer and latency (number of rounds). Our analytical result agrees with the simulation result of [79], as it may be seen in Table 9.2 that using $G(0.8, 2)$ eliminates substantial unnecessary messages as compared to duplicate avoidance like in Gnutella or even with partial list. However improvements with our scheme are dramatic with appropriate parameter choices, either when the whole population is online or when only 10% of them are online, and is significantly better even than $G(p, t)$[79], with a marginal drawback of an additional push round (latency) in each case.

Scheme	$\frac{Msgs}{R_{on}(0)}$	Push rounds
Gnutella	4	7
Using Partial List	3.92	7
Haas et.al.'s G(0.8,2) [79]	3.136	7
Our Scheme, $P_F(t) = 0.9^t$	2.215	8

$R_{on}/R = 10^3/10^3; \sigma = 1; F_r = 0.004(fanout = 4)$

Scheme	$\frac{Msgs}{R_{on}(0)}$	Push rounds
Gnutella	40	5
Using Partial List	35.22	5
Haas et.al.'s G(0.8,2) [79]	28.49	5
Our Scheme, $P_F(t) = 0.8^t$	16.35	6

$R_{on}/R = 10^2/10^3; \sigma = 1; F_r = 0.04$
(Expected Effective $fanout = 4$)

Table 9.2. Comparison of flooding and gossip mechanisms with the push phase of our algorithm

In conclusion, what may be argued is that once neighbours are located in Gnutella, there is no need to repeat this exercise. However since this scheme is meant for propagating updates, which are relatively infrequent, and using efficient indexing schemes such that message flooding is not required for searching, it is incorrect to assume that established online replicas continue to stay online. It is primarily this kind of unreliable environment, which had prompted us for our push/pull scheme.

9.6 Potential optimizations and self-tuning

Apart from using certain optimization techniques like directional gossiping [107], we may use certain heuristics to reduce the total bandwidth usage.

Foremost we can use an acknowledgement (ack) that replica p sends back to replica f if p receives an update from f. Here p may adopt a policy to reply back only to the first or first k random replica f_1, from which it receives the update. Consequently, f_1 will have better chances to find online replicas in future updates. Since most of the messages are wasted in locating online replicas, this strategy may help. Furthermore, if there are other replicas f_i which had forwarded an update to p, they will assume (from the lack of an ack) that p is offline, and hence may decide not to send future updates, thereby reducing the number of duplicate messages sent to p. This strategy will only be effective for short time intervals, since over a period of time, p is expected to be online according to a random distribution for all the replicas. Moreover it is desirable that f_i again forwards updates to p in remote future since it is possible (quite likely) that f_1 is no more online.

The number of duplicate messages received by a replica p also provides an essential, locally available metric that p may utilize to tune parameters $P_F(t)$ and f_r. In the case that replicas adopt a policy of sending multiple acks then the number of acks may be used similarly. Though we have used deterministic $P_F(t)$, it can be assigned an ad-hoc value as well without affecting the general inferences drawn from such simplistic functions. Most importantly, $P_F(t)$ should be reduced significantly with increment of t, specially since there are fewer unaware replicas as t increases, and hence the need to propagate message is lesser. Another information available to the replicas is the message length $L(t)$ which provides an estimate of the extent of propagation of update message, and hence to tune f_r and P_F.

Similarly, we may significantly decrease the number of Pull messages. It is not necessary for a replica p coming online to instantaneously pull updates. It can wait till it receives push based update from some replica f and pull updates from f. This saves the unnecessary messages which are otherwise wasted to find an up to date online replica. However this lazy and optimistic approach has a performance tradeoff during queries. This is because if there is a query Q that p recieves, then it will not be able to answer the query (since it is not aware whether it has an up to date information), but instead will itself have to initiate a pull.

9.7 Related work

Updates in the presence of replication is a widely studied topic. This section positions our approach with respect to the research done in the areas of database systems, group communication, and P2P systems. Most of the related work has been done in the context of database systems. Recently, group communication techniques (lazy epidemic algorithms) have been investigated for this purpose as well. Only little work on replication and updates is available from the P2P domain.

9.7.1 Replication and updates in databases

Several recent approaches exist that attempt to address some of the problems given in Section 9.2, but cannot meet all requirements. For example, iAnywhere Solutions [68, 89] propose a central server-based scheme for mobile data management with wireless and offline data access. Clearly, such centralized schemes do not suit a peer-to-peer environment. [97] describes hierarchy-less distribution of data, but the approach is confined to highly available sites. [78, 141] propose optimistic replica management schemes in a peer-to-peer way (or using hybrid schemes), and primarily address mobility through reconciliation techniques which may be considered as variants of anti-entropy. They use a pull-based reconciliation scheme which thus exhibits limited consistency guarantees. In Xerox PARC's Bayou project [166], a weakly connected replicated storage system, update conflict management has been addressed through tentative and committed writes to provide best effort consistency, along with anti-entropy based conflict resolution. It is similar to the approach presented in [74]. However, it assumes significantly less replicas, less updates (and hence conflicts), and while the system supports frequent temporary network partitions, it assumes that disconnections are rather short.

Data replication in Mariposa [158] uses economic measures to determine when to replicate data and uses unidirectional periodic reconciliation techniques and rule-based conflict resolution. Other economic paradigms to maintain distributed data replicas include [87, 88, 126] where a primary copy model is used to provide one-copy serializability. These approaches optimize resource usage but inherently assume the availability of the resources and replicas in general. The Ficus [127] replicated Internet-file system tries to scale to large numbers of users and files. It uses optimistic P2P-based file replication based on the assumption that in file sharing systems, conflicts are rare, and can often be resolved.

9.7.2 Group communication and lazy epidemic schemes

Many conventional database replication schemes and file sharing schemes often use either group communication methods or rumor concepts to propagate updates [35], assuming that such primitives are in themselves robust enough. Group communication primitives typically can tolerate a specific number of faults but are not applicable in such highly unreliable environments that we assume. Even gossip-based approaches, for example, probabilistic broadcast [27], are insufficient, and a hybrid push/pull scheme is required. The novel approach of our work is the use of push/pull in the context of replicas being offline long and frequently, and in the significant reduction of message overhead in the push phase. Our approach may be considered as a generic version of [79].

Randomized rumor spreading algorithms may be categorized [54] by the gossip termination decision criteria used by peers. The first category is defined by whether nodes use feedback from other nodes (for example, whether they already know the rumor or not) and thus decide on their future course, or not (generally called "blind" then). The second category of algorithms uses either probabilistic (coin flipping) or deterministic (counter) measures to determine when to stop. Many rumor spreading algorithms are hybrids of these two categories and results indicate that feedback and counters improve the latency of rumor spreading.

By using the partial random list of replicas to which a rumor has been sent, we are also sending information about replicas hitherto unknown to certain nodes, thus gradually propagating global information, and the idea is similar to work done in the context of resource discovery, called the name dropper scheme [82].

The directional gossiping approach [107] exploits knowledge of the logical connectivity/topology of the system to minimise the number of messages required for update distribution. Unfortunately, this approach cannot be applied in the scenarios we address because replicas go online/offline frequently which changes the topology considerably so that topological knowledge cannot be exploited.

In another analysis of randomized rumor spreading [96], it has been shown that a hybrid push/pull algorithm has performance benefits since push grows fast (quadratically in the beginning, and then exponentially) when there are very few nodes with the rumor, and a very large target audience, while pull is efficient when most nodes already have a rumor, and very few still need it. Their algorithm, besides being very complex assumes continuous availability of all peers, and can tolerate a limited number of permanent failures, but does not deal with fluctuating (online/offline) behavior. However, their hybrid push/pull scheme motivated us to employ a similar strategy. In our work we exploit the advantage of push/pull, though there is a subtle difference of objectives. We exploit the exponential nature of push to achieve a rapid spread of updates among online nodes, so that any node coming online later may easily pull the same.

9.7.3 Peer-to-peer systems

Generally state-of-the-art P2P systems consider the data they offer to be very static or even read-only. Unsurprisingly, most of them thus do not address updates. Typically, centralized (or hierarchical) P2P systems, such as was Napster, can maintain a centralized index of data items available at online peers. If an update of a data item occurs this means that the peer that holds the item changes it. Subsequent requests would get the new version. However, updates are not propagated to other peers which replicate the item. As a result multiple versions under the same identifier (filename) may co-exist and it depends on the peer that a user contacts whether the latest version is accessed. The same holds true for most decentralized systems such as Gnutella [40].

The Freenet [39] P2P system uses a heuristic strategy to route updates to replicas which is uncertain to guarantee eventual consistency. Searches replicate data along query paths ("upstream"). In the case of an update (which can only be done by the data's owner) the update is routed "downstream" based on a key-closeness relation. Since the routing is heuristic, the network may change, and no precautions are taken to notify peers that come online after an update has occurred, consistency guarantees are limited.

In OceanStore [148] every update creates a new version of the data object (versioning). Consistency is achieved by a two-tiered architecture: A client sends an update to the object's "inner ring" (some replicas who are the primary storage of the object and perform a Byzantine agreement protocol to achieve fault-tolerance and consistency) and some secondary replicas that are mere data caches in parallel. The inner ring commits the update and in parallel an epidemic algorithm distributes the tentative update among the secondary replicas. Once the update is committed, the inner ring multicasts the result of the update down the dissemination tree. There is no specific analysis or evaluation of the latency, overheads and consistency guarantees for this update scheme.

In many overlays, particularly those which use the overlay partitioning to determine object location and store the actual content (rather than pointers) replicate the content in a globally determined small fixed number of replicas at peers located deterministically, e.g., consecutive peers on the ring in DHASH [46] and hence can support read/write operations [123]. Updates of these few deterministically located replicas is algorithmically trivial, however, such an update mechanism is not more generally applicable. Our approach provides the flexibility to deal with diverse replication mechanisms and arbitrary sizes of replica networks.

9.8 Future work

Tuning the push phase may not only be done through feedback mechanisms (to determine when to stop pushing), but also by a speculative (feed-forward) mechanism. In this chapter, we have used heuristics to find proper parameters, but we plan to explore the possibility of both feed-back and feed-forward to evolve a proper mechanism of parameter tuning using local knowledge. It may be interesting to look into bimodal behavior [27, 79] in the specific low online probability environment we considered. Bimodal behavior denotes a reliability model which corresponds to a family of bimodal probability distributions, i.e., the traditional "all or nothing" guarantee becomes "almost all or almost none." Also the effect of non-uniform online probability of peers needs to be explored. In such a scenario a relatively reliable network backbone would exist and thus would make possible further performance improvements. Apart focussing on improving the communication primitive, dealing with updates in peer-to-peer systems with multiple writers for same object, conflict resolution, etcetera still remain open problems. We speculate that instead of relying on a generic underlying mechanism, these issues will be better handled at application layer tailored to meet application requirements.

9.9 Conclusions

This chapter described an efficient, generic push/pull gossiping algorithm for highly unreliable, replicated environments. It provides an analytical model to demonstrate the significant reduction of message overhead using certain optimizing techniques (partial lists) and proper tuning of the gossiping (push) phase which in consequence improves the scalability of the algorithm. The analytical model for the gossiping algorithm is a significant contribution in contrast to most of the literature in this area which relies on simulation results. Our algorithm subsumes several existing gossip schemes, hence the analytical model and results are valid for those variants of flooding algorithms. We have demonstrated that our algorithm is robust and applicable in unreliable environments such as current peer-to-peer systems, and it has been integrated as an integral part of the P-Grid software. Another major advantage of the algorithm is that it is totally decentralized and uses no global knowledge but exploits local knowledge instead. This makes it suitable for state-of-the-art systems in the P2P domain as well as potentially for mobile and ad-hoc networking environments.

Part IV

Conclusion

10. Conclusion

"The appearance of the wheels and their work was like unto the colour of a beryl; and they four had one likeness; and their appearance and their work was as it were a wheel within a wheel." — Ezekiel 1:16 (from the Old Testament)

10.1 Interplay of peer-to-peer systems

Complex peer-to-peer systems are more than just how a set of peers interact and interdepend but also how different such systems comprising numerous peers (may be the same physical peers participate in different such P2P systems) interact and interdepend to perform and provide wide range of functionalities.

The main focus of this dissertation was to look at four fundamental functionalities which are required by P2P storage systems, apart many other systems and applications which too may use some of these - (i) discovery of resources; (ii) address independent communication; (iii) maintaining availability of stored content; and (iv) a gossip based robust communication primitive for replica updates. In designing these, from a system designer's perspective we come across intriguing interdependences of these systems. In addition, we explore other ideas to enable such intertwined systems as well as to use such systems to create derivative applications.

These fundamental building blocks (substrates) can be, and in fact are used in numerous other peer-to-peer systems and applications. We summarize such dependencies and inter-dependencies in Table 10.1, particularly referring to existing systems and applications which make use of our systems.

Some of these derivative applications like a decentralized public key infrastructure [50] using a self-referential directory as an alternative to web-of-trust models (like in PGP [66]) are our direct contributions. Many other endeavors including that of managing social relationships and meta-information about peers (trust/reputation [9]), semantic overlays [3], peer-to-peer information retrieval [134] among others have been pursued separately but using the substrates designed in this dissertation as building blocks. One may expect many more interesting applications that can be derived based on some of these.

Both the substrates themselves, as well as other applications using them exploit some of the salient features like the good multi-faceted load-balancing under dynamic workload as well as their low-cost fault-tolerance and adaptivity to network dynamics.

Self-organizing Substrates at large		
System/Application		
Sys/App Name	Functionality	Substrate(s) used
Contributions of the author		
P-Grid [8] (based on original proposal of the data-structures [1])	Load-balanced structured overlay (internet-scale distributed index)	Self-referential directory (for overlay maintenance) Updates (for structural replication maintenance)
Complex queries	Range queries [52] Similarity queries [95] (by others)	P-Grid's order preserved indexing
Self-referential directory [5]	The overlay used to maintain information about the overlay itself.	P-Grid (for discovery) Updates (for propagating latest information)
Updates [51]	Keep arbitrary set of replicas synchronized	P-Grid/Self-referential directory (to re-locate replica peers)
Persistent storage system [47]	To provide highly available and persistent storage	P-Grid (to access stored content) Self-referential directory (to exploit content already stored at returning peers) Updates (potentially, for new objects or new versions)
Decentralized PKI [50] (preliminary work)	As an alternative to web-of-trust to store and retrieve peers' public keys securely.	P-Grid/Self-referential directory
Used by others		
Managing peers' social relationships	Manage meta-information about peers e.g., Trust/reputation management [9] Semantic overlay network (GridVine [3])	P-Grid/Self-referential directory (to locate peers across sessions)
Information systems	P2P information retrieval [134] XPath query processing [159]	P-Grid's indexing
Content distribution network	Caching [45]	P-Grid's overlay routing structure
Back-up system (potential use)	To preserve data of individual users from disk or other failures (like virus attack)	Persistent storage

Table 10.1. Use of self-organizing substrates in various systems and applications, including mutual dependencies of some of these generic substrates (fundamental building blocks) - namely - indexing (P-Grid [8]), peer meta-information management using self-referential directory [5], persistent storage [47] and update [51] propagation mechanism within unstructured replica sub-networks. Though not too many usages of highly persistent and available storage have been explicitly shown here, any application dealing with content will preferably need such a reliable storage primitive. Here we indicate direct usages of the substrates, but because of the interdependencies, indirectly almost all the P2P applications use all these fundamental building blocks. Furthermore, many of these applications rely on the substrates' good multi-faceted load-balancing and fault-tolerance properties for good performance.

During the design and evaluation of these substrates which in their own right are full-blown P2P systems, we have not shied away from occasional use of heuristics, however we have made a conscientious effort to better understand the fundamental behavior and dynamics of such large-scale systems. In that pursuit, we have used the general practice from cybernetics of modeling such systems as Markov processes, and studied the time-evolution of such systems to look into the absorbing or equilibrium states in order to predict the long-run behavior of such systems and accordingly design (parameters) to achieve certain desirable properties - ranging from load-balancing to fault-tolerance.

In the fast-paced (often heuristic) research area of peer-to-peer systems, some significantly radical results have come about based on such an objective pursuit. Moreover the analysis has revealed potential ways to design better algorithms. That is to say, the algorithm design and analysis have strong mutual influence as well.

For example, buoyed by the dynamic equilibrium analysis of overlay networks and design of cost-efficient maintenance mechanisms we wanted to understand the dynamics of lazily maintained storage systems. Apart from the successful reuse of the basic dynamic equilibrium analysis[1] for such systems, we could propose a simple randomized variant of existing lazy repair algorithm which has substantial performance gains in comparison to the existing mechanism in terms of the storage system's resilience for same communication overhead.

The significance of reduction of maintenance and administrative cost for such large-scale systems while providing provable (probabilistic metrics) resilience guarantees under wide range of dynamic environment can not be overstated.

Similarly, we are arguably the first to come up with a mechanism for fast construction of overlay, and only one while taking into account the possibility of arbitrary skew in load-distributions. Multi-faceted load-balancing as offered by our overlay, along with the salient features of the possibility to rapidly deploy it, make it suitable for data centric as well as a wide range of network and communication applications.

Such better algorithmic (re-)designs resulting in better system performance is a direct benefit of using the analysis methodology that we have pursued in this dissertation. We briefly explore this idea next.

10.2 We build upon our tools

We can discern a significant contrast in the systems design style we have pursued in this dissertation in comparison to the traditional practice.

Most (P2P) systems designs and corresponding algorithms have concentrated on achieving some invariants deterministically. For example, in the design of the Chord system, the self-stabilization algorithm focuses on maintaining the global ring invariant. In the design of the TotalRecall storage system, the invariant

[1] While the philosophy is the same, both the involved dynamics and the details of the dynamic equilibrium analyses are of course different for the two different systems.

to maintain was a threshold redundancy. In the design of OceanStore, quorums based on classical distributed algorithms are used to maintain replica consistency, explicitly assuming a rather reliable environment with predetermined bound of limited faults in the inner ring.

The performance evaluation of systems designed to meet invariants deterministically came a posteriori. In other words, somehow the systems determine the analysis. While using such an approach to design systems, failure to strictly meet the system invariants are expected, even as a design objective, to render it useless, at least until the invariants are reestablished.

In contrast, we often looked first at the impact of local behavior of peers on the convergence of the system as a whole to some global properties. The "legal zones" of operation for the systems we design are then those where the global properties are such that the system invariants (e.g., connectivity of the overlay network; coverage of replicas or availability of stored content) are also met, but often only probabilistically. The system algorithms are designed such that they adapt the local behaviors so that the system as a whole is "safely" within such a legal zone of operation for diverse environments. This provides the opportunity for a more graceful performance degradation in an adversarial situation, as well as better exploration of performance cost trade-offs in general.

It is possible that some of the systems and algorithms we designed could have been invented by chance. Nevertheless, causally, the analysis methodology we employed strongly influenced in the design of these systems. Based on this confessedly limited experience, we speculate a strong influence of the employed tools (more specifically, analysis techniques) on the salient properties of the resulting systems.

Throughout the history of human civilization, we have built upon the tools we have (and employed). This brings us to the somewhat philosophical, nevertheless very practical question relevant to systems designers, "Do the analytical tools we employ determine some fundamental characteristics of the systems we design?"

10.3 Analyzing self-organization

We summarize the concepts from probability theory which we used within the context of our general analysis methodology of using Markov models for analyzing self-organization. In particular, we discuss how the analysis can be generally classified as (i) transient versus steady-state analysis; and (ii) mean value versus density (distribution) function based analysis.

10.3.1 Transient versus steady-state analysis

We mentioned at the beginning of this dissertation that the emergent behavior of the self-organizing systems correspond to the equilibrium or absorbing states of the Markov chains with which the system are described. We worked under this premise to analyze and find the "attractors" of self-organization for the various systems studied here.

A dynamic equilibrium of a system is studied by looking into a possible stationary distribution of the corresponding Markov process. This was the case for our studies of overlay networks as well as storage systems (redundancy) maintenance, under continuous churn and maintenance.

A transient analysis is employed in order to study if the system converges to some absorbing states. This was the model chosen to analyze a single step of the key-space partitioning process, where absorbing states corresponded to peers deciding on one of the two possible partitions. Similarly, in the push phase of the gossiping primitive, the absorbing state corresponded to peers already receiving the gossip, i.e., in epidemic algorithm terminology, infected.

The ensemble system state does not change once the system arrives at either a steady-state (If one of the external factors, like rate of churn is changed, the system will be perturbed out of the steady-state.) or absorbing states, but there is a fundamental difference between these two kinds of “attractors” of self-organization.

Once the system reaches an absorbing state, there is no more dynamics in the system. The interaction process is over. In contrast, a steady-state is a continuous process, where individual participants continue to change their (microscopic) local properties, even while the system’s ensemble (macroscopic) properties stay the same, unless one of the external forces is changed, in which case the system is perturbed out of the steady-state again.

Next we look into the two class of probabilistic analysis that can be generally be found in the literature.

10.3.2 Mean value versus density (distribution) function

We observe that in general two different analysis techniques can be found in the literature for studying the stochastic behavior of probabilistic systems (which is what self-organizing systems inherently are): (i) The first approach can be broadly termed as a *mean value analysis (MVA)*, where the evolution of the mean state is studied assuming that the system always actually resides in the mean state. (ii) The second approach looks at the probability distribution function of the system states at any time instance, thus studying the *evolution of the (probability mass) distribution function (EoDF)*.

The former approach (MVA) simplifies the analysis and is a convenient tool for system designers to understand the expected behavior of the system, but at the cost of losing potentially important information like the deviation, the stability and the convergence of the system. This is because MVA collapses the problem into two dimensions - time and design parameters, with the system state assumed to have a singular value (corresponding to the mean) at any point of time. That is to say, the system state is averaged first, and then the time evolution of such an average-state approximation of the system is studied. Because of the reduction of the whole system states into a single (averaged) value, the resulting analysis is simpler, and for a desired resulting system state, the design parameters can be determined analytically.

EoDF on the other hand results in a more complicated model since it looks into the time evolution of the probability distribution of the system states for any choice of parameters. Moreover, solving the resulting equations is computationally more intensive. The reward is that it yields detailed information of the deviation of the system behavior from the mean behavior, among other properties. EoDF is also accurate, unlike MVA which may not hold for non-linear systems.

The single step of key-space partitioning process was studied earlier in Section 4.3.2 using MVA. We provide an EoDF analysis of the process to observe the deviation from the mean in the Appendix A.1. Such an EoDF analysis also helps ascertaining the correctness of the MVA which, as mentioned above, may not always (particularly for non-linear systems) hold.

For the analysis of the overlay networks as well as storage systems (redundancy) under churn and maintenance, we actually used a hybrid analysis. We looked into the probability density of the different system states; e.g. how many routing entries are usable, or how many redundantly stored blocks are available. However, the probability corresponding to any specific state itself is bound to vary over time. We (implicitly) considered instead that this value resides at the mean value. In fact, as observed from simulations for each of these systems, the value does fluctuate from the mean value, but only slightly, so that studying how the mean value of the probability distribution function (MVA of the EoDF) evolves over time was sufficient for our purpose. It may however be, that looking at the probability distribution in stead the mean value of how the system's states evolve over time can reveal more information about the stability of the steady-states, which we could validate only using simulations so far.

For analyzing the gossip scheme we employed MVA, since we were primarily interested in looking into the communication overhead. It has been shown that gossip mechanisms generally have bimodal behavior [27], i.e., it either spreads to the whole network or dies without infecting almost any member in the gossip group. Such bimodal behavior is typically studied using EoDF analysis. We assumed that the parameters chosen were aggressive enough so that the gossip is expected to spread, and calculated the expected latency and the expected number of messages for the gossip to spread.

10.4 Future directions

The first half of this first decade of the century saw the rebirth and rapid growth of the peer-to-peer computing paradigm. Some of the fundamental barriers to develop dependable scalable systems and applications are more or less crossed. These included development of internet-scale index structures, storage systems and communication mechanisms which perform reliably even though the individual participants act unreliably (though collaboratively). Together these provide the basic infrastructure on which to build other peer-to-peer systems.

Research to refine the existing ideas and systems will continue, e.g., our initiative to build more flexible structured overlays for heterogeneous environment and workloads [69, 70] using Kleinberg [98] style small-world topology; query-adaptive partial indexing [99]; or our recent work studying how distinct overlays can be merged gracefully [48].

Loose ends in either refining existing systems or developing the understanding of the dynamics of such systems (or both) also remain. Details of such impending future works were already indicated in place wherever we studied various systems in this dissertation. While there is still room for improvements, working solutions providing peer-to-peer infrastructure with decent performance and reliability guarantees are already in place. The stage is set for the next steps; e.g., to invent other interesting peer-to-peer applications and services and proper business models to be built on top of these basic building blocks.

As more and more such P2P applications and services proliferate our daily lives and economy, they will also increasingly become susceptible to selfish or malicious users. Various aspects of security as well as mechanisms to provide incentives or other means to enforce collaboration, as well as conflicting goals of accountability and privacy, among others will become increasingly important. While there has been some initial work in these general directions, unlike the infrastructural issues, the solutions to meet security concerns for such P2P infrastructures are far from practicable yet.

Even as the infrastructure to store and access resources get mature, making the diverse resources usable by end users poses another serious challenge. To take a simple example, different set of users may represent information in different formats. Even when using the same technology, say XML, these may use different schemas, or may not even follow a well defined schema. Dealing with such heterogeneous and “dirty” resources to make them globally usable is another busily researched area which will continue to both challenge and enthrall researchers. Composing services, enabling distributed work-flows, etcetera are many other such avenues which will need further developments to enable usage in a P2P fashion.

In this thesis we had a first glimpse of how different (but still rather similar) large-scale systems intermingle together. As more and more large-scale systems from diverse domains are developed and deployed, a natural next step will also be to integrate these diverse large systems into an amalgamated system.

Such an amalgam may comprise of sensor networks gathering information and feeding it to peer-to-peer overlays for storage, dissemination and (stream) processing, and access by end users through different physical channels - wired or wireless, as well as new information triggering automated response like further processing of some information.

Thus to say, in the mid-term future, there is a plethora of possibilities for large scale distributed systems, and these are some of the general directions we can envisage the P2P research community to pursue.

10.5 A blend of systems and theory for peer-to-peer research

This dissertation looked into the design of some of the basic P2P systems, employing tools from cybernetics, particularly employing Markov analysis to understand the self-organizational processes and exploit this understanding to refine the systems themselves. The contributions of this dissertation to P2P systems design and development is thus two pronged. Firstly the dependable scalable systems with interesting and useful, and sometimes unique (with respect to the contemporary systems) properties and functionalities themselves, which can in turn be used by end users as well as developers of other systems. Secondly, this thesis provides significant insight into the dynamics of such large-scale systems and adapts some standard tools from other well established research areas like cybernetics to suit the study of peer-to-peer systems. We also see how often the system's design and the analysis have had strong mutual influence.

In the long run, we expect such a methodology to also be used more widely in distributed systems design, analysis and development in general and in particular for large-scale dynamic systems like P2P systems.

References

[1] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *Proceedings of the Sixth International Conference on Cooperative Information Systems (CoopIS)*, 2001.

[2] K. Aberer. Efficient search in unbalanced, randomized peer-to-peer search trees. Technical report, 2002.

[3] K. Aberer, P. Cudre-Mauroux, M. Hauswirth, and T. van Pelt. Gridvine: Building internet-scale semantic overlay networks. 2004.

[4] K. Aberer, A. Datta, and M. Hauswirth. The Quest for Balancing Peer Load in Structured Peer-to-Peer Systems. Technical report, 2003. EPFL Technical Report IC/2003/32.

[5] K. Aberer, A. Datta, and M. Hauswirth. Efficient, self-contained handling of identity in peer-to-peer systems. *IEEE Transactions on Knowledge and Data Engineering*, 16(7), 2004.

[6] K. Aberer, A. Datta, and M. Hauswirth. Multifaceted Simultaneous Load Balancing in DHT-based P2P systems: A new game with old balls and bins. *Self-* Properties in Complex Information Systems, "Hot Topics" series, LNCS*, 2005.

[7] K. Aberer, A. Datta, and M. Hauswirth. P-Grid: Dynamics of self-organizing processes in structured P2P systems. *(Book chapter) Peer-to-Peer Systems and Applications, State-of-the-art Survey series, LNCS 3485*, 2005.

[8] K. Aberer, A. Datta, M. Hauswirth, and R. Schmidt. Indexing data-oriented overlay networks. *31st International Conference on Very Large Databases (VLDB)*, 2005.

[9] K. Aberer and Z. Despotovic. Managing Trust in a Peer-2-Peer Information System. In *Proceedings of the 10th International Conference on Information and Knowledge Management (ACM CIKM)*. ACM Press, 2001.

[10] S. Abiteboul, I. Manolescu, and N. Preda. Constructing and Querying Peer-to-Peer Warehouses of XML Resources. In *SWDB*, 2004.

[11] I. Abraham, B. Awerbuch, Y. Azar, Y. Bartal, D. Malkhi, and E. Pavlov. A generic scheme for building overlay networks in adversarial scenarios. In *In Proceedings of IDPDS03*, 2003.

[12] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communication environments. In *SIGMOD Conference*, 1995.

[13] E. Adar and B. Huberman. Free Riding on Gnutella. *First Monday*, 5(10), 2000. http://firstmonday.org/issues/issue5_10/adar/index.html.

[14] S. Ajmani, D. E. Clarke, C.-H. Moh, and S. Richman. ConChord: Cooperative SDSI Certificate Storage and Name Resolution. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, number 2429 in LNCS. Springer, 2002.

[15] Akamai. Akamai Content Delivery Network. http://www.akamai.com.

[16] L. O. Alima, S. El-Ansary, P. Brand, and S. Haridi. DKS(N,k,f): A Family of Low Communication, Scalable and Fault-Tolerant Infrastructures for P2P Applications. In *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, 2003.

[17] L. O. Alima, A. Ghodsi, and S. Haridi. A Framework for Structured Peer-to-Peer Overlay Networks. In *Post-proceedings of the Global Computing Conference*, LNCS. Springer Verlag, 2004.

[18] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in publicresource computing. *Commun. ACM*, 11(45), 2002. http://setiathome.berkeley.edu/.

[19] D. Angluin, J. Aspnes, J. Chen, Y. Wu, and Y. Yin. Fast construction of overlay networks. In *SPAA*, 2005.

[20] J. Aspnes and G. Shah. Skip graphs. In *Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, Baltimore, MD, USA, 2003.

[21] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. Pruhs, and O. Waarts. On-line load balancing of temporary tasks. *Journal of Algorithms*, 22:93–110, 1997.

[22] A.-L. Barabási. *Linked: The New Science of Networks*. Perseus Publishing, 2002.

[23] R. Bhagwan, S. Savage, and G. Voelker. Understanding availability. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[24] R. Bhagwan, S. Savage, and G. M. Voelker. Replication Strategies for Highly Available Peer-to-Peer Storage Systems. Technical Report CS2002-0726, UCSD, 2002.

[25] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker. Total recall: System support for automated availability management. In *Proceedings of Symposium on Networked Systems Design and Implementation (NSDI)*, Mar. 2004.

[26] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. *SIGCOMM Comput. Commun. Rev.*, 34(4):353–366, 2004.

[27] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *TOCS*, 17(2), 1999.

[28] C. Blake and R. Rodrigues. High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two. In *Ninth Workshop on Hot Topics in Operating Systems (HotOS-IX)*, 2003.

[29] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *SIGMETRICS*, 2000.

[30] W. Buntine and M. Taylor. ALVIS: Superpeer Semantic Search Engine. In *European Workshop on the Integration of Knowledge, Semantic and Digital Media Technologies*, 2004. http://www.alvis.info/alvis/.

[31] C. Buragohain, D. Agrawal, and S. Suri. A game theoretic framework for incentives in p2p systems. In *Proc. 3rd Intl. Conf. on Peer-to-Peer Computing*, 2003.

[32] J. Byers, J. Considine, and M. Mitzenmacher. Simple load balancing for distributed hash tables. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[33] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Security for structured peer-to-peer overlay networks. In *In Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02)*, 2002.

[34] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC)*, 20(8), 2002.

[35] S.-W. Chen and C. Pu. A structural classification of integrated replica control mechanisms. Technical Report CUCS-006-92, Dept. of CS, Columbia Univ., 1992.

[36] V. Cholvi, P. Felber, and E. W. Biersack. Efficient search in unstructured peer-to-peer networks. In *Proceedings of the Sixteenth Annual ACM Symposium on Parallel Algorithms (SPAA)*, 2004.

[37] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3), July 2003.

[38] I. Clarke, T. W. Hong, S. G. Miller, O. Sandberg, and B. Wiley. Protecting Free Expression Online with Freenet. *IEEE Internet Computing*, 6(1), 2002.

[39] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, number 2009 in LNCS, 2001.

[40] Clip2. The Gnutella Protocol Specification v0.4 (Document Revision 1.2), Jun. 2001. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[41] B. F. Cooper. An optimal overlay topology for routing peer-to-peer searches. In *ACM/IFIP/USENIX 6th International Middleware Conference*, 2005.

[42] L. Cox, , C. D. Murray, and B. Noble. Pastiche: Making backup cheap and easy. In *In Proceedings of Fifth USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[43] R. Cox, A. Muthitacharoen, and R. Morris. Serving DNS using a Peer-to-Peer Lookup Service. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, number 2429 in LNCS. Springer, 2002.

[44] A. Crespo and H. Garcia-Molina. Semantic overlay networks for p2p systems, 2002.

[45] P. Cudré-Mauroux and K. Aberer. A Decentralized Architecture for Adaptive Media Dissemination. In *IEEE International Conference on Multimedia and Expo (ICME 2002)*, 2002.

[46] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, 2001.

[47] A. Datta and K. Aberer. Internet-scale storage systems under churn - A steady state analysis. In *The 6th IEEE International Conference on Peer-to-Peer Computing (P2P)*, 2006.

[48] A. Datta and K. Aberer. The challenges of merging two similar structured overlays: A tale of two networks. In *International Workshop on Self-Organizing Systems (IWSOS)*, 2006.

[49] A. Datta, S. Girdzijauskas, and K. Aberer. On de Bruijn routing in distributed hash tables: There and back again. In *The 4th IEEE International Conference on Peer-to-Peer Computing (P2P)*, 2004.

[50] A. Datta, M. Hauswirth, and K. Aberer. Beyond "web of trust": Enabling P2P E-commerce. In *IEEE Conference on Electronic Commerce (CEC'03)*, 2003.

[51] A. Datta, M. Hauswirth, and K. Aberer. Updates in Highly Unreliable, Replicated Peer-to-Peer Systems. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, 2003.

[52] A. Datta, M. Hauswirth, R. John, R. Schmidt, and K. Aberer. Range queries in trie-structured overlays. In *The Fifth IEEE International Conference on Peer-to-Peer Computing (P2P)*, 2005.

[53] A. Datta, W. Nejdl, and K. Aberer. Optimal caching for first-order query load-balancing in decentralized index structures. In *Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, 2006.

[54] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *PODC*, 1987.

[55] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[56] J. Douceur. The sybil attack. In *Proc. of the IPTPS02 Workshop*, 2002.

[57] A. O. (ed.). *Peer-to-Peer: Harnessing the Power of Disruptive Technologies.* O'Reilly & Associates, 2001.

[58] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi. Efficient broadcast in structured p2p networks. In *IPTPS*, 2003.

[59] S. El-Ansary, E. Aurell, and S. Haridi. A physics-inspired performace evaluation of a structured peer-to-peer overlay network. In *IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN)*, 2005.

[60] P. Felber, K. W. Ross, E. W. Biersack, L. Garcés-Erice, and G. Urvoy-Keller. Structured Peer-to-Peer Networks: Faster, Closer, Smarter. *IEEE Data Engineering Bulletin*, 28(1), 2005.

[61] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and S. Diot. Packet-level traffic measurements from the Sprint IP backbone. *IEEE Network*, 16(6), 2003.

[62] M. J. Freedman, E. Freudenthal, and D. Mazires. Democratizing Content Publication with Coral. In *1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[63] P. Ganesan, M. Bawa, and H. Garcia-Molina. Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems. In *VLDB*, 2004.

[64] L. Garcés-Erice, E. W. Biersack, K. W. Ross, P. Felber, and G. Urvoy-Keller. Hierarchical Peer-To-Peer Systems. *Parallel Processing Letters*, 13(4), 2003.

[65] L. Garcés-Erice, P. A. Felber, E. W. Biersack, G. Urvoy-Keller, and K. W. Ross. Data Indexing in Peer-to-Peer DHT Networks.

[66] S. Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly & Associates, 1994.

[67] A. Ghodsi, L. O. Alima, and S. Haridi. Symmetric replication for structured peer-to-peer systems. In *DBISP2P*, 2004.

[68] E. Giguère. Mobile data management: Challenges of wireless and offline data access. In *ICDE*, 2001.

[69] S. Girdzijauskas, A. Datta, and K. Aberer. On Small-World Graphs in Non-uniformly Distributed Key Spaces. In *1st IEEE International Workshop on Networking Meets Databases (NetDB)*, 2005.

[70] S. Girdzijauskas, A. Datta, and K. Aberer. Oscar: Small-World Overlay For Realistic Key Distributions. In *Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, 2006.

[71] Gnutella. Gnutella file sharing network. http://en.wikipedia.org/wiki/Gnutella.

[72] P. B. Godfrey and I. Stoica. Heterogeneity and load balance in distributed hash tables. In *INFOCOM*, 2005.

[73] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, and P. Keleher. Adaptive replication in peer-to-peer systems. In *The 24th International Conference on Distributed Computing Systems*, March 2004.

[74] J. Gray, P. Helland, P. E. O'Neil, and D. Shasha. The dangers of replication and a solution. In *SIGMOD*, 1996.

[75] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 381–394, New York, NY, USA, 2003. ACM Press.

[76] A. Gupta, B. Liskov, and R. Rodrigues. One hop lookups for peer-to-peer overlays. In *Ninth Workshop on Hot Topics in Operating Systems (HotOS-IX)*, pages 7–12, Lihue, Hawaii, May 2003.

[77] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[78] R. Guy, P. Reiher, D. Ratner, M. Gunter, W. Ma, and G. Popek. Rumor: Mobile data access through optimistic peer-to-peer replication. In *Workshop on Mobile Data Access*, 1998.

[79] Z. Haas, J. Y. Halpern, and L. Li. Gossip-based ad hoc routing. In *INFOCOM*, 2002.

[80] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI'05)*, May 2005.

[81] A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The Piazza Peer Data Management System. *TKDE*, 16(7), 2004.

[82] M. Harchol-Balter, T. Leighton, and D. Lewin. Resource discovery in distributed networks. In *PODC*, 1999.

[83] N. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *In proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS '03)*, Seattle, WA, March 2003.

[84] F. Heylighen. The science of self-organization and adaptivity. *The Encyclopedia of Life Support Systems*, 2002.

[85] F. Heylighen and C. Joslyn. Cybernetics and second order cybernetics. *R.A. Meyers (ed.), Encyclopedia of Physical Science & Technology (3rd ed.), (Academic Press, New York)*, 4, 2001.

[86] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed Object Location in a Dynamic Network. In *Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2002.

[87] Y. Huang and O. Wolfson. A competitive dynamic data replication algorithm. In *ICDE*, 1993.

[88] Y. Huang and O. Wolfson. Object allocation in distributed databases and mobile computers. In *ICDE*, pages 20–29, 1994.

[89] Y. Huang, O. Wolfson, and A. P. Sistla. Data replication for mobile computers. In *SIGMOD*, 1994.

[90] P. Huck, M. Butler, A. Gupta, and M. Feng. A Self-Configuring and Self-Administering Name System with Dynamic Address Assignment. *ACM Transactions on Internet Technology*, 2(1), 2002.

[91] IETF-RFC:3174. Secure Hash Algorithm 1 (SHA1), 2001. http://www.ietf.org/rfc/rfc3174.txt.

[92] M. Jelasity and O. Babaoglu. T-Man: Gossip-based overlay topology management. In *Engineering Self-Organising Applications (ESOA'05)*, 2005.

[93] M. F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[94] D. R. Karger and M. Ruhl. New Algorithms for Load Balancing in Peer-to-Peer Systems, 2003. IRIS Student Workshop (ISW).

[95] M. Karnstedt, K.-U. Sattler, M. Hauswirth, and R. Schmidt. Similarity Queries on Structured Data in Structured Overlays. In *2nd IEEE International Workshop on Networking Meets Databases (NetDB)*, 2006.

[96] R. M. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized rumor spreading. In *FOCS*, 2000.

[97] B. Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-r, a new way to implement database replication. In *VLDB*, 2000.

[98] J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.

[99] F. Klemm, A. Datta, and K. Aberer. A Query-Adaptive Partial Distributed Hash Table for Peer-to-Peer Systems. In *International Workshop on Peer-to-Peer Computing & DataBases (P2P&DB 2004)*, 2004.

[100] G. Koloniari and E. Pitoura. Content-Based Routing of Path Queries in Peer-to-Peer Systems. In *EDBT*, 2004.

[101] S. Krishnamurthy, S. El-Ansary, E. Aurell, and S. Haridi. An analytical study of a structured overlay in the presence of dynamic membership. *Joint IEEE/ACM Transactions on Networking*, 2005.

[102] S. Krishnamurthy, S. El-Ansary, E. Aurell, and S. Haridi. A statistical theory of chord under churn. Ithaca, NY, USA, 2005.

[103] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[104] A. Kumar, J. Xu, and E. W. Zegura. Efficient and Scalable Query Routing for Unstructured Peer-to-Peer Networks. In *Proceedings of the IEEE Infocom*, 2005.

[105] J. Li, J. Stribling, R. Morris, and M. F. Kaashoek. Bandwidth-efficient management of dht routing tables. In *Proceedings of the 2nd Symposium on Networked System Design and Implementation (NSDI'05)*, 2005.

[106] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the Evolution of Peer-to-Peer Systems. In *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.

[107] M. J. Lin and K. Marzullo. Directional gossip: Gossip in a wide-area network. In *European Dependable Computing Conference, LNCS*, 1999.

[108] W. Litwin, M. Neimat, and D. A. Schneider. RP*: A Family of Order Preserving Scalable Distributed Data Structures. In *VLDB*, pages 342–353, 1994.

[109] W. Litwin, M. Neimat, and D. A. Schneider. LH* – A Scalable, Distributed Data Structure. *ACM Transactions on Database Systems*, 21(4):480–525, 1996.

[110] M. Luby. LT Codes. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002.

[111] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *International Conference on Supercomputing*, 2002.

[112] R. Mahajan, M. Castro, and A. Rowstron. Controlling the Cost of Reliability in Peer-to-Peer Overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, LNCS. Springer, 2003.

[113] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.

[114] P. Maniatis, M. Roussopoulos, T. Giuli, D. S. H. Rosenthal, M. Baker, and Y. Muliadi. Lockss: A peer-to-peer digital preservation system. *ACM Transactions on Computer Systems (TOCS)*.

[115] G. S. Manku. Routing networks for distributed hash tables. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing (PODC)*, pages 133–142. ACM Press, 2003.

[116] G. S. Manku. Balanced binary trees for id management and load balance in distributed hash tables. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing (PODC)*, 2004.

[117] G. S. Manku. Balanced binary trees for ID management and load balance in distributed hash tables. In *ACM PODC*, 2004.

[118] G. S. Manku. Know thy neighbor's neighbor: the power of lookahead in randomized p2p networks. In *Proceedings of the 36th ACM Symposium on Theory of Computing (STOC)*, 2004.

[119] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed Hashing in a Small World. In *USITS*, 2003.

[120] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In *IPTPS*, 2002.

[121] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distrib. Syst.*, 12(10):1094–1104, 2001.

[122] A. Mizrak, Y. Cheng, V. Kumar, and S. Savage. Structured superpeers: Leveraging heterogeneity to provide constant-time lookup. In *IEEE Workshop on Internet Applications*, 2003.

[123] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen. Ivy: A read/write peer-to-peer file system. In *Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[124] M. Naor and j. . T. y. . . U. Wieder, title = Novel Architectures for P2P Applications: the Continuous-Discrete Approach.

[125] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Löser. Super-peer-based routing strategies for RDF-based peer-to-peer networks. *J. Web Sem.*, 1(2), 2004.

[126] E. Pacitti, P. Minet, and E. Simon. Fast algorithms for maintaining replica consistency in lazy master replicated databases. In *VLDB*, 1999.

[127] T. W. Page, R. G. Guy, J. S. Heidemann, D. Ratner, P. Reiher, A. Goel, G. H. Kuenning, and G. J. Popek. Perspectives on optimistically replicated peer-to-peer filing. *Software—Practice and Experience*, 28(2), 1998.

[128] A. Parker. P2P in 2005 (Cache Logic study), 2005. http://www.cachelogic.com/research/p2p2005.php.

[129] D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of ACM Sigmod*, 1988.

[130] C. E. Perkins, B. Woolf, and S. R. Alpert. *Mobile IP Design Principles and Practices.* Prentice Hall PTR, 1998.

[131] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, 2003.

[132] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 1997.

[133] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. *Theory of Computing Systems*, 32, 1999.

[134] I. Podnar, T. Luu, M. Rajman, F. Klemm, and K. Aberer. A Peer-to-Peer Architecture for Information Retrieval Across Digital Library Collections. Technical report, 2006. TechnicalreportLSIR-REPORT-2006-005.

[135] M. Raab and A. Steger. "balls into bins" - a simple and tight analysis. In *RANDOM '98: Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 159–170, London, UK, 1998. Springer-Verlag.

[136] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Brief Announcement: Prefix Hash Tree. In *ACM PODC*, 2004.

[137] V. Ramasubramanian and E. Sirer. Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays. In *Usenix Symposium on Networked System Design and Implementation (NSDI)*, 2004.

[138] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load Balancing in Structured P2P Systems. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, LNCS. Springer, 2003.

[139] D. Ratajczak and J. Hellerstein. Deconstructing DHTs. In *IBM-TR-03-042*, 2003.

[140] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of the ACM SIGCOMM*, 2001.

[141] D. Ratner, G. J. Popek, and P. Reiher. Roam: A scalable replication system for mobile computing. In *Mobility in Databases and Distributed Systems*, 1999.

[142] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *J. SIAM*, 8(2):300–304, June 1960.

[143] M. K. Reiter and S. G. Stubblebine. Resilient authentication using path independence. *IEEE Transactions on Computers*, 47(12), 1998.

[144] M. K. Reiter and S. G. Stubblebine. Authentication metric analysis and design. *ACM Transactions on Information and System Security*, 2(2), 1999.

[145] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: The Oceanstore prototype. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, 2003.

[146] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. In *Proceedings of the USENIX Annual Technical Conference*, 2004.

[147] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A Public DHT Service and Its Uses. In *PProceedings of ACM SIGCOMM*, 2005.

[148] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-free global data storage. *IEEE Internet Computing*, 5(5), 2001.

[149] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal*, 6(1), 2002.

[150] J. Risson and T. Moors. Survey of Research towards Robust Peer-to-Peer Networks: Search Methods. Technical Report UNSW-EE-P2P-1-1, University of New South Wales, Sydney, Australia, Sep. 2004. http://www.ee.unsw.edu.au/~timm/pubs/robust_p2p/submitted.pdf.

[151] J. Ritter. Why gnutella can't scale. no, really. http://www.darkridge.com/~jpr5/doc/gnutella.html, 2001.

[152] R. Rodrigues and B. Liskov. High Availability in DHTs: Erasure Coding vs. Replication. In *IPTPS*, 2005.

[153] A. Rowstron and P. Druschel. Past: A large-scale, persistent peer-to-peer storage utility. In *HOTOS '01: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, page 75, Washington, DC, USA, 2001. IEEE Computer Society.

[154] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany*, 2001.

[155] P. H. Salus, editor. *Big Book of IPv6 Addressing RFCs (Big Book).* Morgan Kaufmann, 2000.

[156] N. Sarshar, P. O. Boykin, and V. Roychowdhury. Percolation search algorithm, making unstructured p2p networks scalable. In *IEEE P2P*, 2003.

[157] A. Shokrollahi. Raptor codes, 2002.

[158] J. Sidell, P. M. Aoki, A. Sah, C. Staelin, M. Stonebraker, and A. Yu. Data replication in mariposa. In *ICDE*, 1996.

[159] G. Skobeltsyn, M. Hauswirth, and K. Aberer. Efficient processing of xpath queries with structured overlay networks. In *ODBASE*, 2005.

[160] Skype.com. P2P Telephony Explained - For Geeks Only, as of April 2006. http://www.skype.com/products/explained.html.

[161] N. Spring, L. Peterson, A. Bavier, and V. Pai. Using PlanetLab for Network Research: Myths, Realities, and Best Practices. Number Second Workshop on Real, Large Distributed Systems.

[162] K. Sripanidkulchai. The popularity of gnutella queries and its implications on scalability. http://www-2.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html, 2001.

[163] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM*, 2001.

[164] D. Stutzbach, R. Rejaie, and S. Sen. Characterizing unstructured overlay topologies in modern p2p file-sharing systems. In *USENIX Internet Measurement Conference (IMC)*, 2005.

[165] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 175–186, New York, NY, USA, 2003. ACM Press.

[166] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *SOSP*, 1995.

[167] K. Truelove. Opennap use crashes, Nov. 2001. http://www.openp2p.com/pub/a/p2p/2001/05/11/opennap.html.

[168] H. Weatehrspoon, B.-G. Chun, C. So, and J. D. Kubiatowicz. Long-term data maintenance: A quantitative approach. Technical Report UCB/CSD-05-1404, UC Berkeley, 2005.

[169] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, 2001.

[170] Wikipedia. Napster. http://en.wikipedia.org/wiki/Napster.

[171] Wikipedia.org. Wikipedia. http://wikipedia.org/.

[172] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. In H. Jin, T. Cortes, and R. Buyya, editors, *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, pages 90–106. IEEE Computer Society Press and Wiley, New York, NY, 2001.

[173] J. Xu, A. Kumar, and X. Yu. On the Fundamental Tradeoffs between Routing Table Size and Network Diameter in Peer-to-Peer Networks. *IEEE Journal on Selected Areas in Communications*, 22(1), 2004.

[174] B. Zhao, L. Huang, J. Stribling, A. Joseph, and J. Kubiatowicz. Exploiting routing redundancy via structured peer-to-peer overlays. In *ICNP*, 2003.

[175] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-are location and routing. In *UC Berkeley Technical Report UCB/CSD-01-1141*, 2001.

Appendix

A. Decentralized partitioning (further analysis).

A.1 Evolution of (probability) distribution function

The properties of the decentralized (adaptive eager) partitioning algorithm AEP (Section 4.3.1) can be studied by modeling it as a Markov process. As observed in Section 10.3.2, there are two different analysis techniques that can be found in the literature for studying the stochastic behavior of such a randomized system/algorithms: (i) The first approach can be broadly termed as a *mean value analysis (MVA)*, where the evolution of the mean state is studied assuming that the system always actually resides in the mean state. (ii) The second approach looks at the probability distribution function of the system states at any time instance, thus studying the *evolution of the (probability mass) distribution function (EoDF).*

Earlier in Section 4.3.1 the MVA analysis was used to determine the algorithm parameters for Adaptive Eager Partitioning (AEP) decentralized partitioning algorithm. Here we perform an EoDF analysis to ascertain the appropriateness of the results obtained there. The time evolution of the probability mass function - the distribution of p^0 and p^1 can be given as:

$$\begin{aligned} P\left[\begin{pmatrix} p^0 \\ p^1 \end{pmatrix}, t+1\right] &= P\left[\begin{pmatrix} p^0-1 \\ p^1 \end{pmatrix}, t\right]\frac{p^1}{n}\beta \\ &+P\left[\begin{pmatrix} p^0 \\ p^1-1 \end{pmatrix}, t\right]\frac{p^0+(p^1-1)(1-\beta)}{n} \\ &+P\left[\begin{pmatrix} p^0-1 \\ p^1-1 \end{pmatrix}, t\right]\frac{1}{n}\alpha \\ &+P\left[\begin{pmatrix} p^0 \\ p^1 \end{pmatrix}, t\right]\frac{1}{n}(1-\alpha) \end{aligned} \tag{A.1}$$

for $p^0+p^1 \leq n$ with the last term replaced by $P\left[\begin{pmatrix} p^0 \\ p^1 \end{pmatrix}, t\right]$ for $p^0+p^1 = n+1$.

In the expression above, we assume that $P\left[\begin{pmatrix} x \\ y \end{pmatrix}\right] = 0$ if either $x < 0$ or $y < 0$.

This equation, along with the initial condition $P\left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, 0\right] = 1$ and $P\left[\begin{pmatrix} p^0 \\ p^1 \end{pmatrix}, 0\right] = 0$ for all other cases, allows to determine the probability distribution function at any time t for any choice of n, α, β. We are of-course primarily interested in the probability mass function of the states at the end of the partitioning process. It turns out that for $n \to \infty$ the expectation value for p^0 and p^1 follow the equations 4.1 and 4.2 until t_f and remain constant thereafter. Thus to say the expectation values of the EoDF and the MVA analyses match, which is why we could use the approximate MVA analysis in Section 4.3.1.

Note that if the system were to lead to non-linear equations (as may be the case for some some other systems) then the MVA analysis might have led to a different result than the EoDF analysis. However solving the MVA equations numerically is computationally much cheaper than solving the equations obtained from EoDF analysis. This is because for the MVA analysis, there is only one variable per system state, representing its mean value. In contrast, for EoDF we need to deal with the whole distribution of each of the states, which makes solving the EoDF based equations computationally expensive. However EoDF allows in addition to obtain the higher order moments of the system states p^0 and p^1 as a function of the design parameters α and β, apart from the probability distribution function itself, which is calculated from equation A.1. Figure A.1 shows the probability distribution function obtained for various desired values of p for a total peer population $n + 1 = 300$ where the design parameters α and β are chosen by solving the MVA based equations.

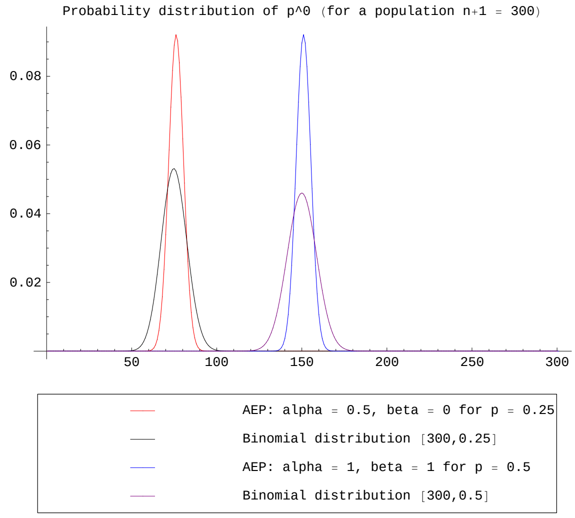


Fig. A.1. Probability distribution function obtained using EoDF analysis of adaptive eager partitioning **(AEP)** algorithm for α and β chosen for desired $p = \frac{p^0}{n+1}$ values is compared with the (binomial) distribution as obtained using autonomous partitioning **(AUT)**.

We also plot the probability distribution (binomial) for the corresponding p values as is obtained by autonomous partitioning (AUT). We observe that the spread (variance) obtained using the AEP decentralized partitioning algorithm is smaller than what is observed in a binomial distribution. Thus to say, apart the fact that we needed AEP to reduce the communication cost of the partitioning process when subjected to the referential integrity constraint, adaptive eager partitioning (AEP) performs better than autonomous partitioning (AUT) in terms of satisfying the proportional replication constraint for a globally known p.

A.2 Error Analysis

In the mean value analysis of the adaptive eager partitioning (AEP) algorithm in Section 4.3.1 as well as the evolution of the density function analysis above, we assumed that the value of p is known to all peers. In practice however peers will derive an estimate for p by sampling. Therefore, in the following we analyze the effect of errors introduced by only approximate knowledge of p. Other potential sources of errors, such as taking the limit case $n \to \infty$ and using mean value analysis turned out to have a negligible influence.

Assume peers obtain s samples from their locally stored data keys. The samples correspond to Bernoulli variables $X_1, \ldots, X_s$ with probability p. The peers estimate p by computing the mean value $X = \frac{1}{s}\sum_{j=1}^{s} X_j$ which is binomially distributed. We would like to determine the effect of an error in estimating p on the values of $\alpha(p)$ and $\beta(p)$ and the resulting effect on the partitioning process when using approximate values of α and β. In the following we will use α and β instead of $\alpha(p)$ and $\beta(p)$ as long as the meaning is clear.

We provide an exemplary error analysis for the evolution of p_i^1 for the case where $p \geq 1 - \log(2)$.

We assume that in step i the estimation value $p_i^* = p + \gamma_i$ is used to determine an estimation value $\beta_i = \beta + \epsilon_i$. The error γ_i is the sampling error obtained by the peer initiating step i. Let us denote by $\tilde{p}_i^1 = p_i^1 + \delta_i^1$ the error introduced into the result of the partitioning process due to sampling errors. We can derive the following closed-form expression for δ_i^1 from analyzing the Markov model of the process.

$$\delta_i^1 = \left(1 - \frac{\beta}{n}\right)^i \sum_{j=0}^{i-1} - \frac{\left(1 - \left(1 - \frac{\beta}{n}\right)^{-j}\right) n\, \epsilon_j}{\beta\ (\beta - n)} \tag{A.2}$$

Since the sampling errors are presumably small we use a Taylor series expansion to approximate $\beta(p)$. In fact, for reasons that will become clear later, we need to make a second order approximation to perform a proper error analysis. For a given value p, we have

$$\epsilon_i \approx \beta'(p)\gamma_i + \frac{1}{2}\beta''(p)\gamma_i^2 \tag{A.3}$$

for small γ_i. We now determine the expectation value and standard deviation for δ_t^1 (to simplify the presentation we will write t instead of t_β in the following). Since $E[\gamma_i] = 0$ and $E[\gamma_i^2] = var[\gamma_i] = \frac{1}{s}p(1-p)$ we obtain for the expectation value

$$E[\delta_t^1] \quad = \quad \frac{1}{2}\beta''(p)\,\frac{n}{s}p(1-p)\,g(\beta,n,t) \tag{A.4}$$

where $g(\beta,n,t) = \delta_i^t/n|_{\epsilon_j=1}$ using (A.2). Asymptotically, for $n \to \infty$, we find numerically that $g(\beta,n,t)$ is bounded between -0.25 and -0.15. This shows that sampling introduces a systematic shift of the balance between the resulting partitions.

Since $var[\sum_{i=1}^{t}\gamma_i] = t\,var[\gamma_i] = \frac{t}{s}p(1-p)$ we obtain for the standard deviation by a similar analysis

$$\sigma[\delta_t^1] \quad = \quad \beta'(p)\,n\sqrt{\frac{t}{s}\,p(1-p)}\,f(\beta,n,t) \tag{A.5}$$

where for the valid range of p we determine numerically that $0.007 \le f(\beta,n,t) \le 0.0105$.

The impact of the errors depends in particular also on the behavior of the functions $\beta'(p)$ and $\beta''(p)$. Using numerical differentiation we observed that the functions are well-behaved in the relevant region.

Performing an analogous analysis for $p < 1 - \log(2)$ the behavior of the functions $\alpha'(p)$ and $\alpha''(p)$ will be relevant for the error behavior. We have included a plot of $\alpha''(p)$ in order to point out an important observation (Figure A.2): For very small values of p the second derivative grows extremely fast, and consequently the error will be large as well.

The error analysis shows that in the presence of sampling errors, we have to include correction terms in the probabilities $\alpha(p)$ and $\beta(p)$ used in AEP.

$$\alpha_{corr}(p) \quad = \quad \alpha(p) - \frac{1}{2}\alpha''(p)\frac{1}{s}p(1-p) \tag{A.6}$$

$$\beta_{corr}(p) \quad = \quad \beta(p) - \frac{1}{2}\beta''(p)\frac{1}{s}p(1-p) \tag{A.7}$$

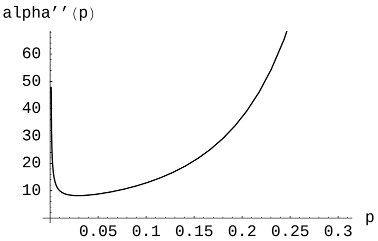


Fig. A.2. Numerical Solution for $\alpha''(p)$.

A.2.1 Numerical Simulation of the Markov Model

To validate the correctness of our analytical models we performed numerical simulation experiments. We simulated five models:

1. MVA: simulation of the mean value model for AEP with known p
2. SAM: simulation of the mean value model for AEP; the value of p is estimated from s samples
3. AEP: discrete simulation of AEP with peers taking discrete decisions based on $\alpha(p)$ and $\beta(p)$ instead of adding mean value contributions as in the mean value model
4. COR: discrete simulation of AEP with corrected probabilities $\alpha_{corr}(p)$ and $\beta_{corr}(p)$ (as obtained in Section A.2).
5. AUT (Discrete Autonomous Partitioning): Discrete simulation of autonomous decision making where p is estimated from s samples

We present the results for $n = 1000$ and $s = 50$. Each experiment has been repeated 100 times.

Figure A.3 shows the deviation of the mean value of p_0^t from the expected value $p\,n$ averaged over all experiments. As expected, using sampling for estimating p leads to a systematic deviation of the resulting distribution (SAM, AEP). The error correction strategy (COR) eliminates the deviation almost completely. Clearly, autonomous partitioning (AUT) on average achieves the desired distribution.

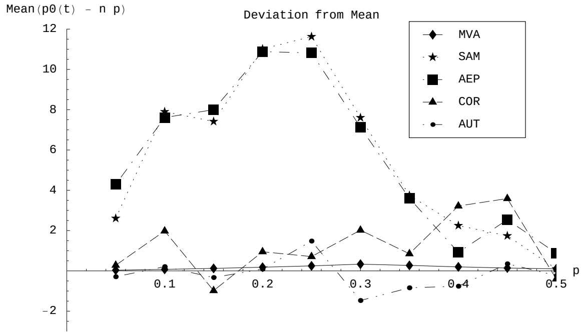


Fig. A.3. Mean of p_t^0 over 100 experiments, the expected value $p\,n$ is subtracted to highlight the deviation.

Figure A.4 shows the cost of each algorithm measured in number of interactions. As theoretically predicted, we observe that adaptive eager partitioning performs better than AUT, except for small values of p (approx. $p < 0.15$) independent of which version is considered (MVA, SAM, COR).

Further experiments with different sample sizes showed that the sample size has practically no influence. Even very small samples (1 or 2 samples) lead to the same results as larger sample sizes. Experiments also showed that adaptive eager partitioning has a further advantage over autonomous partitioning as it reduces the standard deviation of the error in partitioning by approximately a factor of 2. Thus our AEP approach optimizes both performance in terms of number of required interactions and error control in terms of matching the partitioning ratio p.

Superficially, AEP appears to be a more complex algorithm than AUT while not considerably outperforming AUT. However, the complexity is in the analysis required to determine the correct decision proba-

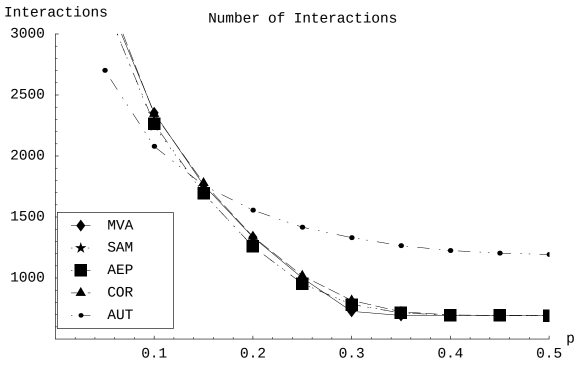


Fig. A.4. Mean total number of interactions over 100 experiments.

bilities, whereas for practical implementation AEP has several advantages since it leads to a lower variance than AUT as seen in the previous section A.1 and it provides an invariant: When taking a decision for a partition, the availability of a reference is guaranteed.

Index

Anwitaman Datta

SHORT BIOGRAPHY

Anwitaman Datta did his PhD at the School of Computer and Communication Sciences (I&C) at EPFL. He has published more that twenty peer-reviewed papers during his PhD, and participated in the Swiss National Center of Competence in Research on Mobile Information and Communication Systems (NCCR-MICS), the Evergrow EU project as well as the internal project for developing the P-Grid peer-to-peer system.

RESEARCH INTERESTS

- Self-organization and dynamics in large scale systems.
- Distributed and Peer-to-Peer (P2P) systems and algorithms.
- Epidemic and gossip algorithms.

EDUCATION AND RESEARCH EXPERIENCE

Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

Oct'2001-Aug2006: PhD. Candidate, School of Computer and Communication Sciences (graduation: August 2006)
Advisor: Prof. Karl Aberer

2000-2001: Graduate School in Communication Systems Department, EPFL.

University of California Berkeley, California, USA.

2005 (April-July): Visiting Scholar at Department of Electrical Engineering & Computer Sciences (EECS)
Hosted by Prof. Ion Stoica and also collaborated with Prof. Michael Franklin.

Indian Institute of Technology, Kanpur, India

1996-2000: B. Tech. in Electrical Engineering (major), Computer Science (minor).

ACADEMIC HONORS

National Talent Search Examination (NTSE 1994) Awardee (awarded by the Government of India).

EMPLOYMENTS & INTERNSHIPS

2001 - 2006: Research Assistant at EPFL, Lausanne, Switzerland.
2000: Associate Member of Technical Staff at Transwitch India, New Delhi, India.
1999: Summer internship at Center for Advanced Technology (CAT), Indore, India.