APPLICATION NETWORKING FOR PERVASIVE CONTENT DELIVERY

SU MU
(B.Eng, M.Eng) HUST, P.R.C

**A THESIS SUBMITTED
FOR THE DEGREE OF PHILOSOPHY OF DOCTOR
DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF COMPUTING
NATIONAL UNIVERSITY OF SINGAPORE
2008**

Acknowledgement

Although this thesis presents my individual work, there are many people who contributed to it by their discussion and support. First, I thank Dr. Chi Chi-Hung, my former supervisor, whose guidance, motivation and discussion have been invaluable throughout my studentship in NUS. I also thank Dr. Sung Wing-Kin, my current supervisor, who gave many supports in the last phase of my study.

I thank Henry Novianus Palit, Hongguang Wang, Choonkeng Chua, Junli Yuan for their help and support on my research work.

I also thank to my family for their love and support: Mom for her care, Dad for his advice and motivation, and Brother for his concern. Thanks and life-long memory to my grandma.

In particular, I appreciate my wife, Junxia Zhang. She gave me her largest encouragement and support, especially during the thesis revision phase. Without her, I could not finish my research.

Contents

List of Figures

List of Tables

Summary

In the past few years, the Web and the peer-to-peer (P2P) based content services have witnessed more heterogeneous requests, due to the emergence of diverse user devices and network connections, and other personalized requirements. To provide the best-fit content for clients, researchers proposed many active content delivery systems, which add transformation applications into the content delivery system and allow it to transform the relayed content according to client's requirements. However, these systems carry a common weakness – they treat content transformation and content delivery as two separate processes. It causes the systems to suffer problems like not preserving content's semantics, poor scalability to adopt applications, and low performance due to application placement and content reuse. To address these challenges, we propose a novel framework - Application Networking - that integrates content delivery and content transformation into a unified service delivery process. The research is exploited in three phases: (i) general Application Networking framework, (ii) App.Net system for the web context, and (iii) AN.P2P system for the P2P context.

The Application Networking framework defines our content service structure and content reuse method. The framework organizes content service as an original content object and a workflow that specifies the allowed operation steps to be performed on the content. The content service can be delivered to the network nodes, which will execute the workflow operations to transform the content for the requesting clients. More importantly, the framework can achieve flexible service placement by delivering partial workflow in the network, and this flexibility facilitates the system to achieve efficient content delivery strategies. Meanwhile, the framework defines a metadata based approach to implement the general-purposed and service-oriented content reuse.

The efficacy of our concept is verified by the App.Net system, which applies the general framework in the Web content delivery context. The system extends the legacy Web system by allowing the Web server to deliver the content service to the edge proxy, which will reuse it to serve heterogeneous requests. We also propose intelligent service placement algorithm to minimize the server-to-proxy transmission cost by delivering partial content services in the optimal ways. The performance evaluation results show our App.Net system outperforms the conventional server-based and proxy-based solutions significantly.

This research is further expanded by the AN.P2P system, which applies the general framework to the P2P networks by enabling the network to populate both the content object and its workflow. Firstly, mathematical analysis is used to show the AN.P2P methodology can achieve less search size or query hops than the plain P2P methods that share the final content presentations in the network. Then, a detailed AN.P2P system on the Pastry network is proposed. It uses dedicated solutions: (i) the selective replication method, (ii) the partial replication method and (iii) the pointers cache method, to perform cost-effective service replication and efficient content reuse. The simulation results show our system achieves much better performance than the conventional P2P solution when serving heterogeneous queries.

Keywords: Application Networking, Web, Peer-to-Peer, Pervasive Content Delivery, Content Adaptation

CHAPTER 1

INTRODUCTION

1.1 Background

Nowadays, content service is one of the most important Internet applications, because of people's intensive desire for acquiring and propagating information. Most content delivery systems are built on either Web or peer-to-peer (P2P) technology, to allow the content provider to publish and deliver his contents to the global users.

The main advantage of Web is the simple client-server communication way and the rich media presentation. Many efficient Web content delivery solutions were proposed to support large numbers of clients that would cause considerable access load on the server and significant transmission load on the backbone network. Typical solutions are the Web cache [Rau99, WC97, RS98, FCAB00 and BO00], the Web replication [KRR01, QPV01], and the content distribution networks (CDN) [Ver02, Hul02, RRR98 and RA99]. On the other side, the P2P technology is highly autonomous and scalable, and many content delivery systems are built on it, such as the file sharing systems [Nap, Kaz], the distributed storage systems [RD01, DKK01], and the P2P supported Web delivery systems [PG04, BT].

In recent years, content services witness much more heterogeneous requirements as never before. Besides the traditional personal computers, many new devices, such as PDAs and smart phones, are used as the browsing tools. Meanwhile, with people coming from different parts of the world, Internet is facing a variety of users with different network access speeds, language requirements, multimedia coding preferences, and so on. Under such conditions, the content services are expected to provide the appropriate responses that fit the network, hardware, software and other personalized requirements of the clients.

However, the traditional content delivery systems were designed for contents with a single presentation and they become deficient under the new conditions. To fill this gap, many "*active content delivery*" systems are proposed. In general, these systems add the content transformation applications into the content delivery path, and allow the network nodes to transform content to the appropriate presentations according to the clients' requirements. According to where the transformation applications are performed, existing active systems can be divided into three groups: (i) the origin-site solutions, (ii) the client-side solutions, and (iii) the intermediary solutions.

The origin-site solutions transform content at the Web server or the origin node in the P2P network. When the origin site receives a request, it will transform the content according to the requirements in the request. The adapted content is then sent to the client. The main drawback of this method is the fully adapted content has very poor reusability at the intermediate proxy cache or peer node, since the response is for a specific group of clients and cannot be reused for clients with different requirements.

The client-side solutions transform content at the client's device. Many systems, such as CSI [RXD02] and AJAX [Ajax], use the downloaded applications to transform and manipulate the response content. Although this solution can better understand the capabilities of the client's device and provides good user interaction, the heterogeneity of client devices restricts its wide usage. More importantly, the solution is not flexible enough as transforming content at the client side may not be beneficial at all situations.

Researchers also proposed the active intermediary solutions [Dik04] that let the intermediate node fetch the original content from the origin site and transform it for different client groups. The solutions only need to install applications at the intermediate node but can reuse them for all the connected clients. Thus, it is a one-to-many approach.

Typical active Web intermediary systems are (i) the active proxy [BJA98, BH01, STR02 and BCH04], where the content applications are pre-installed on the edge proxy; (ii) the active cache [CZB98] and the server-directed transcoding (SDT) system [KLM03], which allow the proxy to fetch small applications from the server dynamically; and (iii) the Content Service Network (CSN) [MSB01] and the Application CDN (ACDN) [RXA03] systems, which put application proxies in the network and allow the CDN servers to transform content using these application proxies.

Correspondingly, the active intermediary is also used in the P2P systems, such as Tuxedo [SSM03] for content caching and SpiderNet [GDY04] for data streaming. Such systems assume some network nodes are installed with the transformation applications, and can direct the content to flow through the required application nodes to form the final presentation for the client.

In summary, the history of the “traditional content delivery systems” presents the trend - the content moves from the origin site to the intermediate network nodes. This kind of *content distribution* improves the system’s scalability and efficiency to serve the *homogenous requests* that ask for contents with a single presentation.

Meanwhile, the invention of the “active content delivery systems” presents a new trend - the content applications are added to the intermediate network nodes. This scales up the node’s capability to transform the relayed content. This *application distribution* has exhibited its benefits to improve the system’s scalability and efficiency to serve the *heterogeneous requests* for the customized content presentation.

1.2 Motivations

Although existing solutions can achieve active content delivery to some extent, they are not sufficient to construct a comprehensive framework. There are some major challenges to be conquered, including (i) how to describe the content service as a well-defined structure, (ii) how to reuse the versioned content, (iii) how to perform efficient service placement, and (iv) how to apply a general framework to different content delivery contexts.

1.2.1 Well-defined Content Service Structure

The active content delivery systems allow the intermediate nodes to transform the relayed content objects. A complete content transformation process usually contains multiple operations. For example, to render a Web page on a PDA, the transformation includes document fragmentation, layout customization, and rending language selection. To describe these operations, current systems propose some methods: (i) the active proxy systems [BJA98 and BH01] use the built-in policies to manage the content transformation; (ii) the OPES framework [BCH04] and CSN [MSB01] use the configuration rules to trigger the transformation applications; and (iii) SDT [KLM03] and ACDN [RXA03] send the server-directed instruction to the proxy to notify the required operation and application.

Although these methods facilitate active content manipulation to some extent, they are still insufficient to describe a complete set of content operations precisely. A complete description should include not only the invocation rules, but also the execution sequence of operations, the application needed for each operation, and the input content and parameters for the application. All these factors are necessary to perform precise content transformation. Moreover, the description should also describe different versions of content objects generated from the transformation. This will facilitate the system to reuse content with high flexibility.

Thus, we should provide a well-defined “content service” structure, and allow the content provider to define the transformation steps for his content. The service specification tells the network nodes how to manipulate a piece of content and generate the correct content presentation for the client. Besides these descriptive functions, the service structure should also be loosely coupled to facilitate the system to (i) form a service with the standard and reusable application modules; and (ii) implement flexible service placement to the network nodes.

1.2.2 Content Reuse

Content transformation will generate different versions of content objects. Reusing these versioned contents, especially on the intermediate node, can reduce the network traffic and the user perceived latency. Many systems (e.g. [STR02]) leverage dedicated applications to retrieve the media properties and reuse content according to these properties. Though such type-oriented approach is effective for specific content types, we also need more general approach due to the following reasons.

- Reuse all types of contents: In the open Internet environment, different types of contents will be delivered. To reuse these contents, we need a general judgment mechanism that is independent of particular content types. Thus, the system can determine the reusability of content without interpreting its data format.

- Service oriented reuse: In many situations, the content reusability should be "service-oriented" instead of type-oriented, which means the reusability is decided by the content service's semantics rather than the content type. For example, an advertisement bar can have an image presentation and a text presentation, where the former is used for clients with rich multimedia support and the latter is for thin clients. In this case, the content reusability should be determined by the advertisement service instead of any reuse rule for either image or text.

Thus, we are motivated to provide a general-purposed and service-oriented content reuse mechanism.

1.2.3 Efficient Service Placement

Another important issue in the active system is how to deploy the service applications for high performance content delivery. This issue can be broken into two aspects: (i) where to place an application, and (ii) how to place a content service containing multiple applications.

- Where to place an application?

An application transforms the input content object to one or multiple output objects. Since the input and output objects usually have different size and reusability, placing and executing the application on different network nodes would result in different transmission traffic and user perceived latency. Thus, where to place an application directly affects the system performance.

Many systems install applications either at the fixed nodes or deploy them dynamically to the nodes that are fixed at the design phase. However, placing application at such fixed location is not efficient in all situations. For example, if a large image is intensively accessed by the clients with different preferences, putting the transcoding application on the edge proxy will be beneficial for content reuse. However, if the image is occasionally accessed, a better choice is to transcode it at the server and only send those few transcoded results.

Thus, the efficient application placement should be dynamic, considering factors as the clients' request pattern, the content's size and cacheability, and the application's size and valid period.

- How to place the content service?

A complete content service may contain multiple applications. Thus, the second issue is how many applications should be deployed to the intermediate node. Some solutions, e.g. ACDN [RXA03], treat all applications in a service as an atomic unit and deliver them as a whole. This is a coarse placement granularity, because the possible choices are to deploy either all applications or none of them. The lack of flexibility would restrict the system performance.

Therefore, an efficient system should supply a flexible service placement strategy. It can deploy content service at finer granularity, which means the system can place some selected applications in the service to the intermediate network node according to the relevant performance gain and delivery cost. Such flexibility enables the system to derive more beneficial placement strategy in the dynamic environment.

1.2.4 Wide Framework Adoption

Since Web and P2P are two important content delivery scenarios, it is important to adopt the proposed active content delivery framework to both contexts to explore its general efficacy. In particular, the current active framework studies are mostly based on Web systems. It is necessary to expand the study into the P2P systems, where quite different architectural and performance issues are presented. Meanwhile, many recent studies [PG04 and BT] proposed the hybrid system that can leverage both Web and P2P technologies. Adopting a general framework will enable active content delivery for such hybrid systems.

1.3 Objectives and Contributions

Intrigued by above motivations, this thesis is targeted to propose a novel framework for pervasive content delivery. The thesis focuses on exploring the efficacy of our framework in the Web and P2P contexts, rather than designing particular content services. The objectives and contributions of this thesis are summarized as follow.

1.3.1 Objectives

Our research is accomplished in three stages, each with a particular target:

- A General Framework

To provide a basis for the concrete systems, we first need a general framework for pervasive content delivery. The framework provides a well-defined description for the content services. This description should facilitate flexible service placement, which means (i) the content applications can be deployed dynamically, and (ii) the content services can be delivered in the network at fine granularity.

Meanwhile, the framework should define a general approach to identify and reuse different versions of content objects. Different from the conventional type-oriented method, this approach should enable the general-purposed and service-oriented content reuse.

- Adopt the framework to the Web Context

Our in-depth system study starts from applying the general framework into the Web content delivery context. The proposed system should enable the Web server to deliver not only the response content but also the relevant applications to the edge proxy. The new system must also be compatible with the legacy Web content delivery mechanisms, so that we can deploy it into the actual Web environment. Intelligent service placement strategy should also be provided for efficient content delivery.

- Adopt the framework to the P2P Context

Next, we will adopt the general framework into the P2P context, which extends our study to the network scenario with large numbers of nodes. Firstly, we need the fundamental methodology for the representative P2P network. Performance analysis will be used to show the widespread benefits of the proposed methodology. Then, we will elaborate the methodology into a concrete system with detailed mechanisms for efficient service placement and content reuse.

1.3.2 Contributions

We believe our research efforts on this thesis will enrich the knowledge base of several research areas, particularly on the fields of Web and P2P content delivery. Our key contributions are summarized as follow.

- Application Networking Framework

We propose the Application Networking framework for pervasive content delivery. The framework extends the traditional content delivery frameworks by integrating content delivery and content transformation into a unified service delivery process. It describes the content transformation operations as a well-defined service structure and enables partial service placement which is not provided by the conventional frameworks. It proposes a metadata-based method to achieve the general-purposed and the service-oriented content reuse.

- App.Net System

We propose the App.Net system that applies the Application Networking framework into the Web content delivery context. Our system is proposed in the background that the content transformation is either based on the origin server or the edge proxy in the conventional Web systems. App.Net enables the server to deliver an intermediate response with relevant service applications to the proxy. It not only generalizes the conventional architectures, but also provides more flexible content delivery and transformation mechanisms. Meanwhile, we extend the HTTP 1.1 protocol to cache different versions of responses and make the new system compatible with the conventional Web systems.

More importantly, optimal service placement algorithms are constructed to minimize the transmission cost in the system. We implemented an App.Net prototype on the Jigsaw platform, and performed simulations based on it. The results show the App.Net system achieves higher performance than the conventional server-based and proxy-based solutions.

- AN.P2P System

We also propose the AN.P2P methodology that adopts the Application Networking framework to the P2P network. Different from the traditional P2P method that only replicates response objects, our method enables the replication of content services. A mathematical model is constructed to show the widespread advantage of the proposed methodology. Our model is also the first quantitative analysis model for the Pastry network. A system prototype is implemented on the Free-Pastry platform, and the simulation results show the AN.P2P method outperforms the conventional Pastry system significantly.

Furthermore, we build a detailed AN.P2P system. Our study provides a general purposed framework for active content delivery in the P2P network, whereas most existing solutions are based on specific applications. In particular, we propose (i) the selective replication method for targeted service replication, (ii) the partial replication method for flexible service replication, (iii) and the pointers cache method for distributed content reuse. Simulation results show all these mechanisms enhance the system performance from different aspects.

1.4 Thesis Outline

The thesis is organized as follows. Chapter 2 reviews the traditional and active systems for Web and P2P content delivery. Chapter 3 proposes our general research framework – Application Networking framework. Chapter 4 proposes the App.Net system architecture that applies the general framework into the Web context. Performance models and optimal service placement algorithm are also provided. The simulations are explained in Chapter 5 to verify the effectiveness of this new architecture. In Chapter 6, we propose the AN.P2P methodology that expands our framework to the P2P network. Mathematical model is proposed to estimate the system performance. Then, a detailed AN.P2P system is discussed in Chapter 7. Relevant solutions are provided to achieve efficient service replication and content reuse. Finally, we summarize the research and discuss the future work in Chapter 8.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter reviews the research work related to our thesis. The traditional and active Web content delivery systems are first reviewed. Then, we review the P2P networks, the replication and caching techniques in the P2P systems, and the active P2P solutions. The literature review is organized as in Figure 2.1.

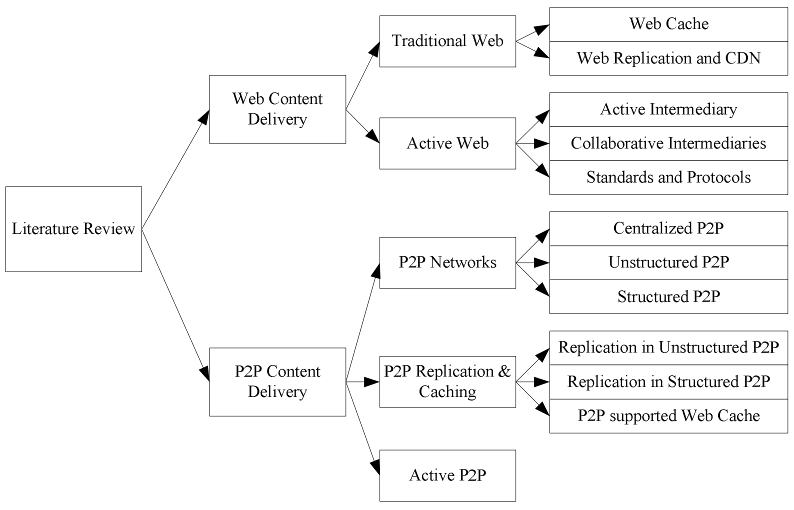


Figure 2.1 Literature Review Organization

2.2 Traditional Web Content Delivery Technologies

In the past decade, the Web has become one of the most popular content service models in the Internet, due to its simple client-server communication and rich media presentations. It has been found that a single Web server cannot manage the load caused by the large numbers of users, due to its limited processing capability, network bandwidth, and the connection availability. Thus, many solutions were proposed for efficient content delivery by distributing the load to multiple nodes. Two well-known solutions are the Web caching and the Web replication.

2.2.1 Web Caching

The Web caching method [BO00] allows the intermediate proxy to cache the Web objects so that the future users' requests for these cached objects will be served from the proxy directly. [BCF99] claims the Web users' access pattern to Web contents presents the temporal and spatial locality characteristics, where the minority of objects is queried most frequently while most others are not queried so frequently. Therefore, caching those high utility objects at the proxy can reduce the user perceived latency and the traffic on the Internet backbone. Many cooperative caching structures [Rau99] were also proposed to explore the collaborative storage of multiple caches. Typical structures are the hierarchical cache proxies and the distributed cache proxies.

The hierarchical caching systems [WC97, RS98, and FCAB00] arrange cache proxies as a tree structure. A proxy may have its parent proxy and the sibling proxies. When receiving a request, the proxy will first look up its local cache. If the cache does not store the object, the proxy will query the sibling proxies, any of which with the object will reply the query. If none sibling replies, the proxy will forward the request to its parent proxy. The hierarchical caching structure enables content reuse between different proxies and it can reduce the network bandwidth demand by reducing redundant object retrieval at different levels of the network. However, the hierarchical caching causes significant disk storage requirement at the high-level proxy, and the cache-hit ratios at the high-level proxies are less than those at the low levels due to the "trickle-down effect" [DCG01].

To overcome the disadvantages of the hierarchical caches, [PH97] proposed the distributed cache proxies structure, where only the leaf proxies are responsible for caching objects while the higher level proxies only index the objects cached by its descent nodes. The inter-proxy query is resolved in the hierarchy, but object is transmitted between leaf proxies. Thus, the system avoids the requirement for the high-level nodes to maintain large storage space, and exploits the hierarchy for quickly locating the cached objects.

To achieve efficient caching information exchange, [RS98] and [FCA00] proposed the “cache digests” and “cache summary” respectively to index the cached objects at a proxy. They use a Bloom filter to summary the keys of the cached objects. A proxy will propagate its digest to other proxies and keep it updated periodically. When a proxy tries to query other proxies, it will check their digests and select one proxy that most likely has the object. In addition, CARP [Carp97] provides another distributed caching structure, where the proxies are arranged as an array and each proxy is responsible for a range of objects. A hash function maps the object’s key to a responsible cache proxy. Thus, CARP can provide a deterministic request resolution to locate an object in the array of proxies.

2.2.2 Web Replication and Content Distribution Networks

The Web replication technology allows the origin server to put replicas of its content objects to other servers. The client’s request can be served by either the origin server or one of the replica servers. [Rab98] defines two major issues for content replication: (i) “request distribution” that redirects a request for a logical object to one of the physical object replicas; (ii) “replica placement” that determines how many object replicas are to be placed onto which set of replica servers.

Replication is intensively used in the Content Distribution Network (CDN) [Ver02, Hul02], such as Akamai [Aka] and Digital Island [Dig]. CDN defines a well-organized network infrastructure with a large number of proxy servers distributed in diverse geographic areas. Dedicated request distribution algorithms are used to redirect the requests, and content placement algorithms are used to replicate content objects strategically.

Typical request distribution methods are DNS (Domain Name Service) indirection, HTTP redirection, and IP multiplexing. The DNS indirection [RRR98] allows the DNS servers to map a host domain name to a set of IP addresses and choose one of them for every request. The HTTP redirection [Http] allows the Web server to redirect a request using the HTTP redirection message. This method can distribute requests at fine granularity, down to individual pages, whereas it is heavy weight, needing extra pair of HTTP messages. The IP multiplexing [Ver02] uses a multiplexing router to forward the requests to the back-end servers. Both the DNS indirection and the HTTP redirection are used for global request distribution, while the IP multiplexing is for local request distribution.

[Rab98] claims there are two replica placement ways: (i) the static placement and (ii) the dynamic placement. The static placement assumes rather coarse-granularity decisions, based on the observed request characteristics. It cannot adapt to the changing environment and is infeasible for a large-scale system. The dynamic placement requires the system to constantly monitor the requests to various resources and continuously adjust replica sets for all objects. The dynamic placement algorithms normally consider the load and proximity factors to provide acceptable responsiveness adapting to the changing request patterns. [QPV01] argues finding the optimal replica placement in a CDN is a NP-hard problem. Hence, heuristic approaches [KRR01, QPV01] were used in the actual networks.

To have a general view about the CDN infrastructure, we review the RaDaR system [RRR98, RA99], proposed by AT&T. The RaDaR (Replicator and Distributor and Redirector) architecture is illustrated in Figure 2.2, where the CDN divides the network into multiple management domains. Each domain consists of a Replication Service, a Multiplexing Service and multiple hosting servers. The multiplexing service distributes requests the hosting servers. The request distribution algorithm considers the client- server proximity and the hosting server's load. A hosting server that is nearest to the requesting client and the server's load is below a watermark will be chosen to serve the request. RaDaR allows each hosting server periodically runs the replica placement algorithm. The dynamic replica placement algorithm is designed to improve the client-server proximity and the load balancing on the hosting servers. The RaDaR system can effectively reduce bandwidth consumption and latency, remove hot spots from the network and smooth out bursts in bandwidth demand, while imposing little network traffic overhead.

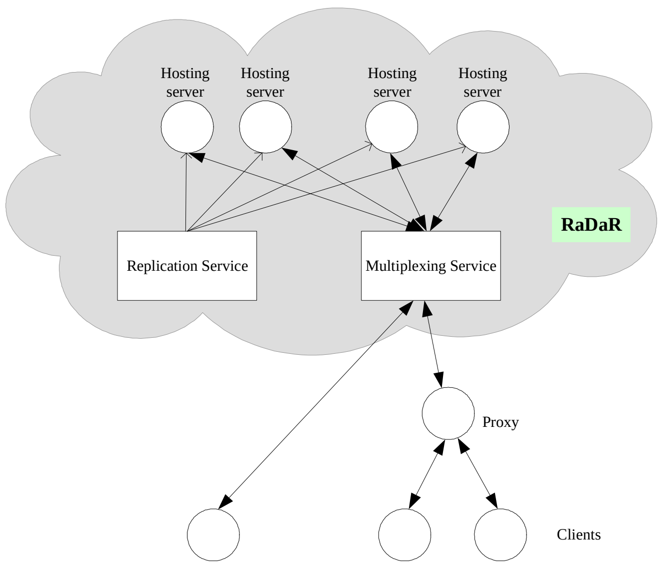


Figure 2.2 Architecture of RaDaR

2.3 Active Web Intermediaries

In recent years, the Internet has witnessed the emergence of different user devices, such as PCs, PDAs, and smart phones. These devices have wide variations among their computation power, display capability, and the associated network bandwidth availability. It is a challenge to the content provider with one source to offer the best-fit presentations to the global users with different requirements. To address these heterogeneous requirements, value-added operations are put into the content delivery process to perform content adaptation or service customization. The adaptation operations can be performed on the server, the client or the proxy.

If we perform content adaptation at the origin server, the adapted content will be generated at the origin server and sent out from the server. The intermediate proxy that relays the response to the client may cache the adapted objects. The server-side adaptation lets the content provider customize his own contents to achieve good quality for the delivered results. However, the main drawback of this method is the considerable low content reuse at the intermediate cache proxy, because the fully adapted response for one group of clients cannot be reused for other clients with different requirements.

Content adaptation can also be performed at the client, using the client-side scripts (e.g. JavaScript) or applets (e.g. Java applet). Example solutions are CSI [RXD02] and AJAX [Ajax]. [RXD02] proposes a client-side inclusion (CSI) method that uses JavaScript to compose dynamic Web pages at the client's browser. It reduces the last-mile transmission cost, since the client can reuse the page template so that it only needs to retrieve the page fragments for every request. Another technique is AJAX [Ajax]. It enables web pages to be more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire web page does not have to be reloaded. These client-side methods provide good user interaction and flexible presentation transformation, whereas they have some limitations. Firstly, the methods cannot deal with the scenario where the operations need to be executed before the content is sent to the client. For example, operations like cropping a large image to smaller size to shorten the downloading time should be performed in the network rather than on the client's device. Secondly, the methods require the client device to interpret the delivered scripts or applets. Nevertheless, it is not easy to find a ubiquitous script accepted by a community of the heterogeneous client devices.

To overcome the drawbacks in the server-side and client-side methods, many researchers propose to perform content adaptation on the intermediate proxies. These intermediary-based methods change the traditional network to be the active network. Unlike the traditional networks that are just the passive carrier of data, the active networks have the capability to inject customized applications into themselves. The active network nodes can execute these customized applications to manipulate the data flowing through the network. These solutions are called as the "active Web intermediary" methods [Dik04].

The active Web intermediary methods have several advantages. As the contents are transformed on the intermediate proxies, these proxies can reuse the retrieved original contents to serve the clients with different requirements. This solves the poor content reuse problem for the server-side methods. On the other hand, the active intermediary methods execute the installed or downloaded applications at the proxies, which can reuse these applications for a community of clients with heterogeneous devices. Thus, the intermediary methods are the one-to-many solution and more scalable than the client-side methods.

Existing active Web intermediary systems into three categories, according to their respective emphases: (i) the “active proxy” that focus on the active mechanism of a single proxy; (ii) the “collaborative intermediaries” that emphasize the cooperation between multiple proxies; and (iii) the “active CDN” systems that adopt the active intermediaries into the CDN network.

2.3.1 Active Proxy

An early active proxy system is MOWSER [BJA98]. It can perform active transcoding for the relayed data to present Web information for the mobile client. On receiving a request from the client, the proxy fetches the preferences set by the client and transforms the original object to the most suitable format. MOWSER proxy can insert appropriate ACCEPT headers into the request to specify the client’s capability and preference. The Web server would generate an appropriate version of contents according to the ACCEPT header. Meanwhile, the proxy can modify image and video objects received from the server before transmitting them to the clients if necessary. To support clients with very limited resources and hardware capabilities like PDAs, the proxy even parses the HTML stream to remove the active content and tags that cannot be handled by the client’s device. With adaptation being done at both request and response steps, MOWSER can match the preferences of the clients, while using the wired bandwidth in efficient manner.

An improved active proxy is the Service Enabled Caching Proxy (SECP) [BH01]. It provides an open and flexible structure for the active proxy, as shown in Figure 2.3. The SECP has three significant features: (i) multiple processing points, (ii) the invocation rules, and (iii) the proxylet with the callout server.

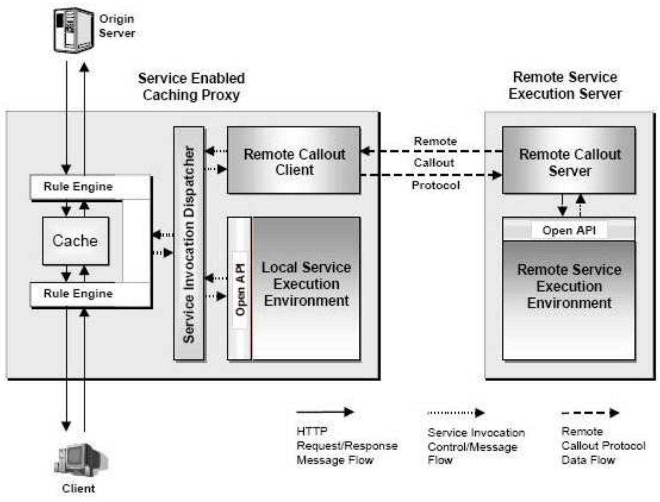

Origin Server
Service Enabled Caching Proxy
Remote Service Execution Server
Rule Engine
Cache
Rule Engine
Service Invocation Dispatcher
Remote Callout Client
Open API
Local Service Execution Environment
Remote Callout Protocol
Remote Callout Server
Open API
Remote Service Execution Environment
HTTP Request/Response Message Flow
Service Invocation Control/Message Flow
Remote Callout Protocol Data Flow
Client

Figure 2.3 Structure of Service Enabled Caching Proxy

Firstly, the SECP proxy can perform the add-in operations at four "processing points", including the client request point, the proxy request point, the origin server response point, and the proxy response point. The usage of multiple processing points allows the proxy to perform a wide range of services, such as language translation, virus scanning, Web access control and other personalization services.

Secondly, SECP makes use of the rules to trigger the service modules. The rules can be configured by the content providers or the clients. The proxy will trigger suitable services to manipulate the request/response on behalf of the content providers or the clients. The usage of rules improves the system's flexibility in that the proxy can maintain large numbers of services and execute them correctly according to the requirements of different parties.

Thirdly, SECP allows the adaptation services to be implemented as either the local applications or the remote services. The system implements the lightweight service as a "proxylet", which is installed and executed on the proxy locally. However, the system places the heavyweight service, such as video transcoding, onto a separate application server. The proxy can access the application server through ICAP (Internet Content Adaptation Protocol). As the resource consuming computations are offloaded to the application server, the proxy's service capacity will scale with the number of requests rather than the content transformation load.

In summary, the SECP system can support many kinds of services, perform content adaptation for different parties, and offload heavyweight operations to the application servers. This structure evolves to be the standard OPES framework.

In above systems, the service modules are pre-installed on the active proxy. This requirement limits the scalability of system in the open Internet environment, where a large number of services exist and new services are created continually. Therefore, researchers proposed many "dynamic active proxy" systems that allow the service modules to be deployed to the proxy dynamically. Typical systems include the Application Level Active Network (ALAN) [FG98], the Active Cache [CZB98], and the Server-Directed Transcoding (SDT) [KLM03].

SDT provides a representative structure for the dynamic active proxy. Its general operations are shown in Figure 2.4. The system lets the origin server to send content transcoding directives to the active proxy along with the response content object. The directives specify the properties of the response object and a location from which an "applet" can be downloaded. When the proxy receives such response, it will download the applet first. Then the proxy executes the applet to transcode the original response object according to the client's preference. Finally, the transcoded content presentation is sent to the client. Meanwhile, the proxy caches the response object with the applet, and reuses them for subsequent requests.

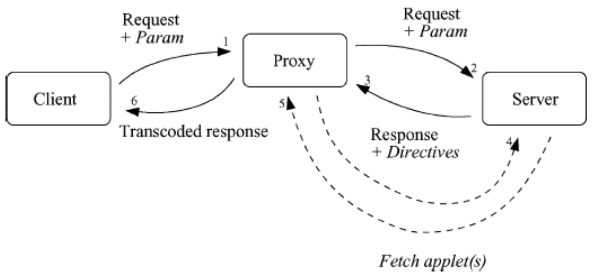


Figure 2.4 Structure of the Server-Directed Transcoding System

[CZB98] indicated the dynamic active proxy can be used for many applications, including multimedia transcoding, user access authorization, advertising banner rotation, client-specific information distribution, and web page’s delta encoding. Compared to the basic active proxy, the dynamic active proxy is more flexible and scalable, because new applets can be downloaded to the proxy dynamically. However, such dynamic application deployment also causes some security problems. [FG98] and [CZB98] mentioned several concerns about the system security: (i) the applet should be retrieved from dedicated applet reservoir or the trustable Web servers; (ii) the applet should have the controlled interface to access the system resources; and (iii) the applet’s resource consumption should be monitored by the system.

2.3.2 Collaborative Intermediaries

Many collaborative intermediary systems were also proposed to overcome the limited storage and processing capability of a single proxy and enable cooperation between multiple proxies. Typical systems have Ninja, CANS, and the collaborative hierarchical proxies.

- Ninja

Ninja [GWB00] composes an active network with distributed data services and active proxies. The data services provide the original contents for clients to request, and the active proxies transform the content sent from the service to the client. To address the heterogeneous requirements from diverse user devices, Ninja proposes to establish a data path dynamically that passes through multiple active proxies to compose a complex adaptation service. Within a path, the adaptation is decomposed into several steps that are performed sequentially by the active proxies along the path. Moreover, [BS01] proposes to place cache along the adaptation path. The objects stored by a cache are actually the adapted result from the previous proxy and the input object to the next proxy. This enables the network to fulfill subsequent requests while requiring the minimal adaptation costs. However, this paper does not provide quantitative study on the effectiveness of this caching method.

- CANS

[FSK01, FSA01 and IHA02] propose the Composable Adaptive Network Services (CANS) architecture. Different from Ninja that is based on the static services, CANS enables service composition and decomposition on multiple active intermediaries using mobile agents called "drivers". In the CANS network, a service is composed by the data source, the data path, and the drivers. The drivers can form data path from the data source to the client by moving data from their input ports to the output ports. After a data flow is sent out from the data source, it passes multiple drivers distributed along the data path and finally reaches the client. CANS uses the type-based composition method, and it requires that the output data type from the previous driver should be compatible to the input data type of the next driver. The author provided a mapping algorithm that can automatically generate a data path and allocate drivers to proxies while yielding maximum throughput.

- Collaborative Hierarchical Proxies

[CYH00] proposes a collaborative Web content transcoding system based on the hierarchical proxies. From bottom to top, the proxies inside the hierarchy are named as edge proxies, internal proxies, and the root proxy respectively. When an edge proxy receives a request, it will check the local cache for a reusable object. If none is found, the proxy will forward the request to the higher-level proxies. The query forwarding will stop at an internal proxy that has a reusable object in the cache, or the root proxy that will retrieve the object from the server. After obtaining the object, the proxy will check the client's preference and transcode the object according to the client's preference. The system uses three policies to balance the load on hierarchical proxies.

(i) NOINFO policy: it lets the current proxy perform the transcoding without considering any load balancing.

(ii) Threshold policy: it considers only local load. The proxy decides to delegate to a lower-level proxy if the present proxy's utilization exceeds a given threshold.

(iii) Least Loaded policy: it lets the proxy perform object transcoding only if its load is the lowest among all lower-level proxies in the request path.

The simulation results showed that the least loaded policy achieves the least response latency. This indicates that global information is useful for balancing load between the active proxies. However, this study only considers a single transcoding task, whereas the content adaptation may include multiple tasks. How to assign multiple transcoding tasks in the hierarchy is a topic to be considered.

2.3.3 Active CDN

The active Web intermediary is also adopted into the CDN networks. Typical systems are the Content Service Network (CSN) and the Application CDN system (ACDN).

- Content Service Network (CSN)

[MSB01] proposes the Content Service Network (CSN) system, where some application servers, which are installed with service applications, are added to the boundary of the CDN network. The CDN servers can trigger the service applications on the application servers to do the transcoding work. CSN proposes the "Internet Service Delivery Protocol" (ISDP), which allows the system to perform the added services on behalf of either the content providers or the clients.

When the content provider subscribes a service on an application server, the server will issue him a certificate. The certificate includes a rule to trigger the subscribed service and the ICAP command to access the service. Upon receiving a request, the content provider's Web server will reply the response object together with the certificate to the CDN server, which will parse the certificate's rule and determine whether the content adaptation is needed. If the rule is satisfied, the CDN server will forward the object to an application server using the ICAP command in the certificate. The application server will process the object and return the transformed object to the CDN server, which will send the result to the client.

On the other hand, when a client subscribes to an application server, he will also receive a certificate. The certificate should be attached to the request issued by this client. After the CDN server retrieves the response object, it may send the object to an application server according to the certificate's rule.

Although CSN uses a "redirector" that lets the content provider or the client subscribe to a nearby and lightly loaded application server, it does not provide solution to handle the changing environment. If an application server becomes overloaded or encounters network failure, how will the system adjust the established service subscription, and how should the system reallocate the services on the application servers under such dynamic conditions?

- Application CDN

[RXA03] proposes the Application CDN system (ACDN), based on the RaDaR architecture. ACDN extends the conventional CDN by replicating not only content objects but also Web applications between network servers. The system proposes a placement algorithm to replicate applications automatically. An ACDN server replicates applications to other servers for two purposes: (i) increasing the access proximity or (ii) offloading the present server. For the first purpose, each ACDN server periodically measures the access traffic for the vicinity servers. If the traffic is larger than the application's size plus the application replica's update traffic, the application will be replicated to that server. Meanwhile, if an application's utility is too low, the server may remove it from the local storage. For the second purpose, if an ACDN server detects it is overloaded, it will notify the central replication server to choose a lightly loaded server to offload its applications.

ACDN improves the request distribution algorithm of RaDaR by considering the application replication. The algorithm can redirect the requests to different application replicas for better access proximity and load balancing.

However, a key limitation of ACDN is that it organizes the application as an atomic unit that is replicated as a whole. This is a coarse granularity of replication, which will restrict the system from implementing more efficient application deployment.

2.4 Adaptive Content Delivery Standards

The industry has defined a set of standards for the active intermediary framework. This facilitates the active intermediary components developed by different parties to interoperate with each other. This section reviews the OPES framework and several callout protocols.

2.4.1 Open Pluggable Edge Service

A standard architecture for the Web intermediary system is the Open Pluggable Edge Service (OPES) framework [Opes, BBC02, and BCH04], as illustrated in Figure 2.5. This architecture can be described in three interrelated concepts: (i) the OPES intermediaries, the process nodes operating in the network; (ii) the OPES flows, the data flows that are cooperatively realized by the OPES intermediaries; and (iii) the OPES rules, which specify when and how to execute the OPES service applications.

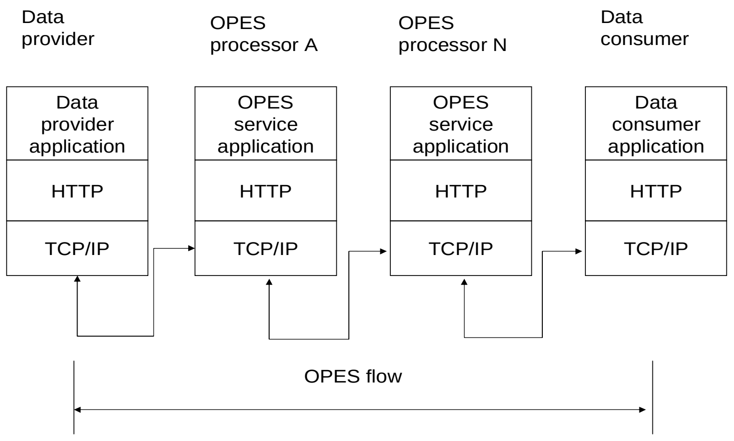


Figure 2.5 OPES Architecture

An OPES intermediary operates on a data flow between the data provider application and the data consumer application. Each OPES intermediary can be divided into two parts: (i) the OPES service application, which analyzes and possibly transforms message exchanged between the data provider and the data consumer; (ii) the Data dispatcher, which invokes the appropriate service application based on an OPES rule set. The OPES rules are specified with the Intermediary Rule Markup Language (IRML) [BH01], and they reflect the intents of the content owners or the content consumers. Thus, a data flow is cooperatively transmitted and manipulated by a data provider, a data consumer, and zero or more OPES intermediaries.

The OPES defines two ways to trigger the service applications. The first way is to invoke the remote services using the OPES Callout Protocol (OCP) [Rou05]. Another way is to use the proxylet [Wal01], an API for the local service applications. OPES framework adopts the “pre-deployment” policy [YH01], where the rules and proxylets are deployed by an administration server to the intermediary servers. The rule owner and the proxylet vendor can be different parties. This means the system can get rules from the content provider or the clients, while getting proxylets from the third-party application vendors. The active intermediaries will be responsible for executing the applications according to the rules. In this way, the OPES framework regulates the roles for each party within the system.

However, the OPES framework does not define the dynamic service deployment procedure and the data reuse policy on the intermediaries.

2.4.2 Callout Protocols

As some content transformations are resource demanding, many systems execute them on the separate application servers. The callout protocols are defined to enable the interaction between the Web proxy/server and the application servers.

A well-known protocol is the Internet Content Adaptation Protocol (ICAP) [Icap]. ICAP is a lightweight remote procedure call protocol based on HTTP. It allows the proxy (i.e. ICAP clients) to pass HTTP messages to the application server (i.e. ICAP servers) for adaptation. The application server processes the transaction through performing specific tasks, like image transcoding, page customization, and virus scanning. When transaction is finished, the processed HTTP message will be sent back to the proxy.

Some researchers also proposed to use SOAP (Simple Object Access Protocol) as a callout protocol. SOAP [Soap] is another lightweight protocol to invoke the remote services. It becomes to be a standard for XML-based messages exchange over the computer network, with the prevalence of XML Web Services. [MDS03] implements an active proxy system, called SEE, which adopts both ICAP and SOAP as the callout protocols. The simulation results show little performance variance between ICAP and SOAP. However, the authors argue that compared to ICAP, SOAP is more extensible and it facilitates the integration of the content adaptation services with the legacy systems.

OPES proposes the OPES Callout Protocol (OCP) [Ocp, Rou05], which marshals application messages from an OPES intermediary to a callout server. Compared to ICAP, OCP provides a wider range of messages to describe components like connection, application, data, transaction, and service group. These rich functioned messages allow the intermediary to manipulate content adaptation operations more precisely. However, OCP is still in the development phase. Its wide acceptance by the industry is not clear yet.

2.5 Peer-to-Peer (P2P) Networks

Recent years, the P2P technology has attracted more and more attention from both the research and the industry due to its inherent scalability. In a P2P network, each node, called a “servent”, acts as both a client and a server, and the data is transmitted directly between the requesting node and the serving node. P2P networks can be divided into three categories: (i) the centralized networks, (ii) the distributed unstructured networks, and (iii) the distributed structured networks.

2.5.1 Centralized P2P Networks

The centralized P2P networks rely on one or a cluster of directory servers to index the objects (e.g. data files) shared by each node. Any node searching for an object needs to query the directory server. The server will reply the requesting node the address of nodes those probably have the queried object. Then, the requesting node will ping those nodes to check the object’s availability, and choose one valid node to download the object. However, due to the usage of directory server, such centralized network scales poorly and has the single points of failures. A successful example of the centralized network is the Napster [Nap], except its legal troubles.

This section reviews a well-known P2P network – Bit Torrent [BT]. According to its working mechanism, the legacy BT system belongs to the centralized network. The BT is designed to distribute large amounts of data widely without incurring the corresponding consumption in costly server and bandwidth resources. To achieve higher data availability and faster data exchange speed, BT partitions one large file into many fragments and allows them to be downloaded from many nodes in parallel.

To share a file, the client first creates a small “torrent” file, which contains an “announce” section and an “info” section. The “announce” tells the address of a “tracker”, the host that coordinates the file distribution. The “info” contains metadata about the shared file, such as the file name and the file length. In addition, the “info” section also specifies the size of file fragments, and the SHA-1 hash code for each fragment, which lets clients verify the integrity of the received data.

The torrent file is published on a website, and it can be accessed by the Internet users. According to the torrent, the client connects to the tracker, which should notify the client a “swarm” of other clients those are currently downloading the same file. Initially, there may be no other peers in the swarm, the client will directly connect to and request file fragments from the “initial seeder”, the node with the initial copy of the file. As more peers enter the swarm, they begin to trade fragments with others, instead of downloading directly from the initial seeder. BT peers download fragments in a random order, so that the opportunity to exchange fragments between peers can be increased.

Since the original BT system relies on a tracker node to implement peer lookup, we categorize it as the centralized P2P network. However, the centralized tracker limits the system’s performance and usually becomes the single point of failures. Therefore, many tracker-less BT systems were proposed, such as µTorrent [UTor], BitComet [BitC] and KTorrent [KTor]. Such tracker-less systems treat every peer in the swarm as a tracker, so that the systems no long need an individual tracker server. The tracker-less improves the system scalability, and it makes the new BT systems to be the distributed networks.

2.5.2 Unstructured P2P Networks

In a distributed and unstructured P2P network, there is neither the centralized directory server nor the precise control on the network topology or file placement. A typical example is the Gnutella [Gnu]. The network is formed by nodes joining the network following some loose rules. To startup, the client must bootstrap and find at least one other node (neighbor). This can be achieved using the pre-existing address list of possibly working nodes shipped with the software, or using the updated web caches of known nodes (called GWebCaches) [GWeb]. After joining the network, any user can share files on its own node. To find a file, a node floods the query to all its connected neighbors, which will further flood to their own neighbors. The query is soon propagated to all neighboring nodes within a certain radius.

However, such flooding mechanism is not scalable as it generates large loads on the peer nodes. The mechanism is also unreliable in the changing network topology. The P2P network is usually composed by computers that keep toggling between connection and disconnection, so that the network topology is never completely stable. Meanwhile, the bandwidth cost of flooding grows exponentially with the number of connected nodes, so that the saturating connections often render the nodes with poor bandwidth useless. As the result, the search requests would often be dropped, and most queries fall in a very small percentage of nodes in the network.

To solve above problems, [SGL04] proposes a tiered Gnutella system. The system treats nodes with limited bandwidth and intermittent connection as “leaf” nodes, which are not responsible for any routing. Meanwhile, the system promotes nodes capable of routing messages to be the “ultra-peers”, which will accept leaf queries and route them into the network. The tiered structure is helpful to stabilize the routing topology and allows propagating searches further through the network.

[LCC02] proposes two methods to reduce the query overhead in the Gnutella network. The first method is called “expanding ring”, which lets the query node increase the flooding radius continuously until the object is found. The expanding ring achieves the bandwidth savings at the expanse of slight increasing the delay. The second method is the “random walking”. Instead of flooding to all neighbors, the method forwards the query to a set of randomly chosen neighbors at each step until the object is found. “State” information is attached to the query message to record the nodes that forwarded this message. This state information can detect the potential circled walking or the overlapped walkers, and thus eliminate many duplicated messages. By choosing appropriate number of walkers, the random walking method can cut down the message overhead significantly by roughly an order of magnitude less than the expanding ring method. However, this paper does not supply a deterministic way to decide the optimal number of walkers.

2.5.3 Structured P2P Networks

The acknowledgement to the drawbacks of the unstructured networks inspires the development of the distributed but structured P2P networks. The “structured” means (i) the network topology is well controlled and (ii) the files are placed at specific locations where the subsequent queries can be easier to satisfy. Most structured networks are based on the Distributed Hashing Table (DHT), which partitions the ownership of a set of keys among the network nodes and can efficiently routes messages to the unique owner of any given key. Typical DHT networks are

Chord [SMK01], Pastry [RD01], CAN [RFH01], and DKS(*N*,*k*,*f*) [AAB03]. This section reviews the DKS network first, and then briefly goes through the Chord and Pastry networks.

- DKS Network

DKS uses a uniform hashing function that can map a node to a particular identifier. The size of identifier space is assumed to be *N*, which means the network can contain at most *N* nodes. The nodes are arranged as a circle in the identifier space (or id ring). Data objects are also mapped onto this circle, sharing the same identifier space as the nodes. The node with the closest matching identifier, which is less than or equal to the data's identifier, is responsible for that piece of data. Figure 2.6 illustrates a network, where the responsible range for node 1 should be [1, 2, 3, 4].

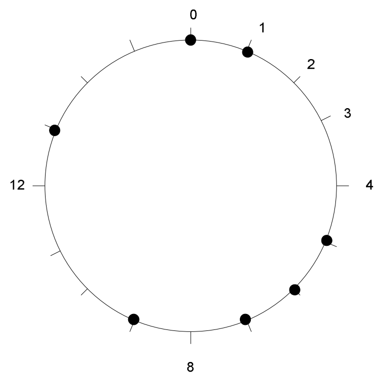


Figure 2.6 An Example DKS network (dots represent network nodes, *N*=16)

To find a piece of data, the requesting node sends into network a query message, which will be routed to a destination node that is responsible for the queried data's identifier. DKS explains this routing process as the "*k*-ary routing" mechanism. Each node maintains a routing table that is divided into $\lceil \log_k N \rceil$ levels (i.e. rows) and each level contains *k* intervals (i.e. columns). The entries for the first level are chosen by dividing the identifier space into *k* intervals and keeping pointer to the first node in each interval. The succeeding levels are created by repeatedly dividing the first interval from the previous level until it cannot be divided anymore. In this way, we can say a routing table arranges its pointers to $\lceil \log_k N \rceil$ sets of nodes at exponentially increasing distance in the identifier space. For example, Table 2.1 sketches the routing table for node 0 (Figure 2.6).

Table 2.1 Routing Table of Node 0, (*N*=16, *k*=2)

Level	Intervals	Responsible Range	Next Hop
1	1	[0,8)	0
	2	[8,16)	7
2	1	[0,4)	0
	2	[4,8)	1
3	1	[0,2)	0
	2	[2,4)	1
4	1	[0,1)	0
	2	[1,2)	1

When a node wants to look up an identifier in the network it checks its routing table and firstly finds the interval in the first level the identifier belongs to. If the responsible node for that interval is the present node itself, it continues to search in the second level and so on until it finds an interval for which the node itself is not responsible. Then, the message is routed to that node.

When a node receives a lookup message, it first checks whether it is responsible for that identifier. If not, it starts searching in its routing table and forwards the message to the next-hop node. This new node will repeat the same process. However, when the new node searches its routing table, it will start searching from the routing table level that equals to the routing table level used by the previous node plus one. In this hop-by-hop manner, the message is routed to the responsible node. For the example in Figure 2.6, if node 0 queries identifier 11, the route should be $0 \rightarrow 7 \rightarrow 9$.

- Chord Network

Chord [SMK01] is one of the first DHT networks. It arranges nodes and data identifiers in a circle as the DKS system. However, Chord's routing protocol can be viewed as a specific case of DKS's *k*-ary routing mechanism, by fixing the value of *k* to 2. It means Chord always divides the identifier space in half, and searches across different routing levels like a binary search.

A significant difference between Chord and DKS is how they handle the node failure and maintain the correctness of routing tables. Chord uses a “stabilization protocol” to detect the failures periodically. Each node keeps track of a number of its immediate successors. If a failure is detected, the detecting node will remove the failed node from the circle. All nodes perform the failure detection periodically to make sure that the routing tables are correct and up to date. However, the stabilization protocol causes considerable bandwidth consumption, and the out-of-date routing information cannot be corrected until the next stabilization period. DKS addresses these problems using an alternative approach [AAB03]. It makes use of the level and internal information embedded in the lookup message to correct erroneous routing entry once it is discovered. When detecting a message is routed incorrectly, the receiving node will find a better candidate node from its own routing table and then tell the sender about this new candidate. DKS can correct the error routing on the fly, and incur much less maintenance traffic.

- Pastry Network

Pastry [RD01b] is another DHT network, but it is different from the previous two networks at the node’s responsible range. The responsible range for a DKS or Chord node starts from the node’s identifier and ends at the successor node’s identifier. However, a Pastry node always resides in the middle of its responsible range. For instance, in Figure 2.6, the responsible range for node 13 in the Pastry network is [11, 12, 13, 14], while it is [13, 14, 15] in the DKS and Chord network. Due to this difference, the Pastry routes message to the node whose identifier is numerically closest to the query key, but the DKS and Chord networks route message to the closest node that is equal to or less than the query key.

In addition, Pastry implements the proximity-aware routing, where the system always routes messages to nearby nodes. This is achieved by maintaining a "neighborhood set" on each node to record a number of physically nearest nodes to the current node. Pastry nodes will exchange the neighborhood information when the network topology changes, so that each node can use the nearby nodes to fill its routing table and implements the proximity-aware routing.

2.6 Replication and Caching in the P2P Networks

Content replication and caching technologies bring important benefits to the P2P networks. Firstly, the P2P networks are usually composed of transient nodes, which constantly join and leave the network, so that replicating content onto multiple nodes can improve the data availability. Secondly, replicating popular objects to more nodes can balance the load on network nodes and avoid hot spots in the network. Thirdly, other benefits like reducing the search size or improving the retrieval proximity can also be achieved using replication and caching. This section reviews these studies as three sections: (i) replication in the unstructured networks, (ii) replication and caching in the structured networks, and (iii) the P2P-suppored Web content caching.

2.6.1 Replication in the unstructured P2P Networks

[LCC02] introduces several replication methods in the Gnutella-like networks, including the uniform replication, the owner replication, the path replication, and the random replication.

The "uniform replication" creates fixed number of replicas for all objects. Though this method can improve the data availability to certain extend, it presents poor performance in terms of average search size and load balancing.

A better method is the "owner replication", which replicates object to the node that requests this object. It results in a proportional distribution of object replicas. E. Cohen and S. Shenker [CS02] proved this distribution could achieve the best load balancing as the number of replicas is proportional to the popularity of the object.

The "path replication", used by Freenet [Free], is an active replication method. When a search succeeds, the object is replicated at all nodes along the path from the requesting node to the serving node. Although it can generate considerable replication cost, E. Cohen and S. Shenker [CS02] indicated the path replication would result in a square-root object distribution and this could reduce the average search size compared to the uniform replication and the owner replication methods.

[LCC02] pointes out that the normal path replication method tends to replicate objects to nodes that are topologically along the same path. It causes the resulting numbers of replicas do not match the precise square-root distribution. The authors proposed a "random replication" method that puts objects to the random nodes but the number of replicas is still proportional to the number of nodes probed. Simulation results show this method results in more precise square-root distribution and it gets smaller search size than the normal path replication method.

2.6.2 Replication and Caching in the Structured P2P Networks

This section reviews the PAST [RD01], replica enumeration [WHB03] and LAR [GSB04] systems, those use replication and caching technologies in the structured networks.

- PAST

PAST [RD01] is a distributed storage system based on Pastry. It uses three sets of data replication and caching methods to achieve high data availability and load balancing between nodes. Firstly, PAST lets a home node (i.e. the responsible node for an object) to replicate its object to *k* numerically closest nodes. These replicas are called the “primary replicas”. If the home node is offline, the query messages are still likely to be routed to one of these primary replicas. Thus, the system can achieve high data availability. Meanwhile, the primary replicas are much likely to be dispersed geographically, so that they can improve the data retrieval proximity in the network. Secondly, PAST solves the storage load imbalance problem, which may be caused by the statistical variation in identifier assignment, the size variance of the inserted files, or the storage capacity variance of individual node. A “replica diversion” technique is used to balance the remaining free storage space among nodes. If a node cannot accommodate a file locally, it considers diverting the file to another node, which is in this node’s leaf set but does not contain a primary replica of the file. The origin node will keep a pointer to the replica diversion. When it receives a query for that file, the node will find the replica diversion through the pointer. Finally, PAST sets a cache on each node to minimize the client access latencies, maximize the query throughput, and balance the query load in the network. A file that is routed through a node as part of a lookup is inserted into the local cache. Due to usage of these techniques, PAST can achieve efficient object storage and delivery.

- Replica Enumeration

[WHB03] proposes a “replica enumeration” system to achieve the controlled object replication in the DHT networks. The system replicates objects to deterministic nodes through a hashing function. This facilitates dynamic updates to these replicas by enumerating and addressing individual replicas using the same hashing process. Meanwhile, the replica enumeration can also improve the overall retrieval proximity. To achieve this, the client first enumerates a possible range of replicas using the hashing function. The distance between the client and these replicas are measured.

Then, the nearest replica will be chosen. Simulation results show the enumeration process causes negligible overhead but considerably improves the user latency.

- LAR

[GSB04] proposes a lightweight and adaptive replication system, called LAR. The paper indicates creating replicas to all nodes on a source-destination path is inefficient and can generate considerable overhead. To balance the load, LAR relies on node load measurement to choose the replication points precisely. The nodes with the least load will be replicated physically, while the system puts onto other nodes the "cache pointers" that records the location of the replica. When receiving a query, the cache pointer can shortcut the original routing and direct the query to the object replica. The simulation results show LAR can achieve reasonable performance while incurring negligible replication overhead.

2.6.3 P2P-based Web Content Caching

Researchers also proposed the P2P networks, formed by a community of computers, to cache the Web objects. Two typical systems are Squirrel [IRD02] and the Top-K replication [KRT02].

- Squirrel

Squirrel [IRC02] implements a distributed Web cache based on the Pastry network, which is composed of a community of personal computers. When a client attempts to access a Web object, the object's URL is hashed to a Pastry identifier. The request is subsequently routed by the Pastry substrate to the "home node" responsible for caching this object. If this request is hit in the home node's cache, it will reply the object to the requesting node. Otherwise, the home node will retrieve the object from the remote server. The retrieved object is returned to the requesting node, and cached by the home node. Compared to a centralized Web cache, Squirrel has several advantages. Firstly, the disk throughput may limit the performance of the centralized cache to the disk bound, but distributing the objects across many nodes allows many disks to fetch data in parallel, which improves the overall system throughput. Secondly, Squirrel pools main memory from many nodes, and this allows more objects are served from the main memory rather than from the disk.

Squirrel is also able to handle node join and leave situations. When a new node joins the network, it notifies the leaf set nodes. The two numerically neighboring nodes will migrate the relevant objects to the newly joined node. When a node leaves the network, it will migrate the cached objects to the left and right neighboring nodes. However, Squirrel cannot gracefully handle the node failure. In this situation, the cached objects on the failed node will be lost, and subsequent requests to these objects need to fetch them from the remote server.

- Top-K Replication

[KRT02] implements another distributed Web cache based on the DHT substrate. However, it uses a Top-K replication mechanism, which can handle the node failure and load balance more gracefully. The system hashes an object's URL to *k* winning nodes, any of which is possible to cache the object. The requesting node will query the winning nodes sequentially. A winning node may serve the object from its local cache or by retrieving it from the remote server. If the current winning node is detected not active, the next winning node will be probed. The simulation results show that by replicating an object to *k* winning nodes the community's external bandwidth usage can be reduced significantly. In addition, the system also uses a rejection mechanism to balance loads between winning nodes. If a winning node is overloaded, it can reject the request and the requesting node will query the next winning node. However, the Top-K replication can cause much query overhead due to the maximum *k* round probing. Although the overhead becomes negligible when retrieving large objects, it can be perceivable when retrieving small objects (e.g. an HTML page). A solution could be choosing the dynamic value of *k* according to the sizes of objects, so that the overall probing overhead can be diminished.

2.7 Active P2P Solutions

Recent years, the demand for content adaptation and service customization becomes increasingly intensive in the P2P systems, due to the increasing requirements from clients with heterogeneous devices and network connections. As a new topic, the amount of active P2P systems is incomparable to that of the active Web systems. Nevertheless, there are still some pioneer studies, such as (i) the active P2P systems for adaptive media streaming, and (ii) the active P2P caching system.

- Adaptive P2P Streaming Systems

Many adaptive content delivery studies in the P2P network focus on media streaming applications. [CN03] proposes a layered P2P streaming solution. It assumes the media content can be divided into several layers. With all layers, the stream provides the original quality of content; with some layers, the stream provides media content with the reduced quality. The peer nodes can selectively relay partial media layers according to their respective bandwidth capability. The receiver can retrieve media layers from multiple peers to satisfy its quality requirement. In this way, the system provides adaptive media streaming subjecting to peer node's bandwidth capability and client's quality requirement.

[GDY04] and [CRK05] propose two similar systems for adaptive media streaming in the P2P network. We use SpiderNet [GDY04] as an example to introduce their working mechanism. SpiderNet supposes that the transcoding services are allocated on some dedicated network nodes, called transcoders. A bounded composition probing protocol (BCP) is proposed to allow the client to compose the required transcoders to form a distributed media streaming service. The service formation process can be described as a series of probing steps, which start from the client and stop at the media source node. Suppose the media streaming service requires function-1 and function-2 in sequence. The client first sends a probe message to find transcoder nodes providing the last transcoding function. When a transcoder node that is qualified to perform function-2 receives the probe message, it will join the service and continue to probe another transcoder node that provides the previous function in the service. When the transcoder node that is qualified to perform function-1 receives the probe message, it will join the service and send a message to the data source node. During the probing process, multiple services (paths) can be formed, and the data source node will choose the optimal path to stream the media.

The advantages of BCP are (i) it can choose network nodes according to the streaming service's Qos requirements, (ii) it can compose the optimal service with acceptable probing overhead, and (iii) the probing process can find multiple streaming paths, so that the system can switch to the backup path in case of sudden node failure in the used streaming path.

- Active P2P Caching System

W-S Shi proposed the Tuxedo [SSM03], which implements a P2P caching system. Tuxedo is different from the Squirrel and Top-K systems, because it not only caches the content objects, but also can transcode these objects and cache the transcoded results. The system is formed by a set of hierarchical proxies. Every proxy maintains a "neighborhood table" for each cached object. Each row in the table tells which another proxy is caching the same object and what the latency and bandwidth for that proxy is. When the proxies retrieve content objects between each other, they also exchange the neighborhood tables and merge the received tables into their own tables. As the result, all proxies form a P2P network, which allows the proxy to retrieve content objects from the optimal node with low latency and large bandwidth.

Meanwhile, Tuxedo allows the proxy to install some transcoding applications. The proxy can execute these applications to transform the content objects with respect to the client's requirements. The system uses a "cache digest" to let a proxy summarize its installed applications and the stored transcoded objects. The digest will be inserted into the neighborhood table for each (original) content object and be propagated with the neighborhood table to other proxies. The proxy can perform local transcoding or retrieve the transcoded object from another proxy. The decision is made by comparing the local transcoding overhead with the remote proxy's latency and bandwidth. As the result, the system can serve client with the diminished latency.

- Observation

The above systems can perform active content delivery in the P2P networks. However, a key problem is they install the transcoding applications on the network nodes statically. Such static application deployment would promise the overall system performance in the dynamic environment, where both the clients' request pattern and the network conditions change continuously. A more suitable way should be letting the system place the applications dynamically according to the content reuse and application utility on the network nodes.

Meanwhile, these systems are mostly proposed for purpose-specific content services, such as media streaming and Web content cache. However, we also need a general framework for active content delivery in the P2P networks. This framework should provide the generic methods to deliver contents and applications. Specific mechanisms can be added to the framework to customize it for specific P2P protocols.

2.8 Summary

In this chapter, we review the work related to our research study. The literature review spans from the traditional Web and P2P systems to the active systems. From the review we see content delivery systems are changing from propagating content data into the network to adding applications into the network, in order to provide pervasive content services to the heterogeneous client requests. This trend motivates our research to provide a general framework for pervasive content delivery, which can be applied to both Web and P2P scenarios.

CHAPTER 3

APPLICATION NETWORKING FRAMEWORK

3.1 Motivations

In the traditional content delivery system, as illustrated in Figure 3.1, the content provider publishes the original content on the "origin site", such as the Web server or a peer node that shares the original copy of content. The clients can request the content and retrieve it using HTTP or specific P2P protocols. The requests and the response are transmitted through a content delivery channel, such as the Web proxies, the CDN network or the P2P network. We call the nodes in the delivery channel as the "intermediate nodes".

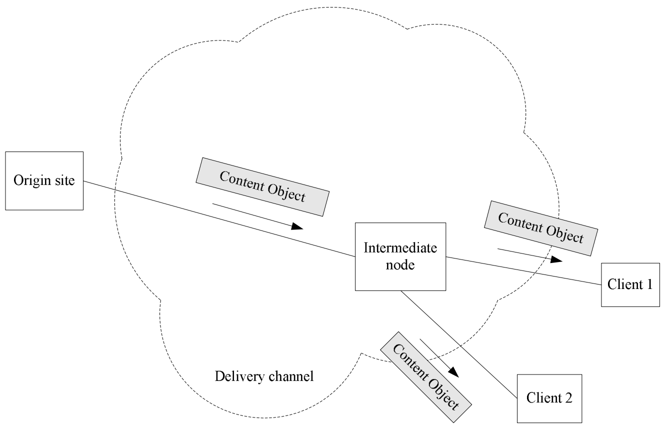


Figure 3.1 A Traditional Content Delivery Framework

The origin site in the traditional content delivery frameworks serves contents with a single presentation. However, nowadays, the origin site receives more and more requests with heterogeneous requirements, due to the emergence of various user devices, different network connections and the other personalized demands. To satisfy such requests, many adaptive solutions were proposed. They can transform the original content to appropriate presentations according to the clients' requirements. Example applications include media transcoding, Web page customization, language translation, document watermarking, and so on. Though these solutions can achieve adaptive content delivery to some extent, there have several key limitations, regarding to the content semantics, the service scalability and the system efficiency.

(i) Existing solutions usually perform content transformation by adding the transformation applications into the delivery channel. However, the added operations are usually orthogonal to the content provider's intentions for the content's semantics. As the result, they may manipulate contents inappropriately and generate results that do not preserve the content's semantics.

(ii) Many solutions [Opes and MSB01] require the transformation applications to be pre-installed on the network nodes. Though the pre-installation approach can be applied to those popular applications, it has poor scalability in the open Internet context, where huge numbers of applications exist and new applications are continuously adopted.

(iii) Many solutions execute the transformation applications at the fixed network nodes. However, this static application placement is normally the result of human judgment base on experiential analysis, and it is not beneficial in all situations. The placement cannot adapt to the changing request pattern and network condition to provide efficient content delivery constantly. Though some systems, such as ACDN [RXA03], can reallocate applications dynamically, they organize the required applications as an atomic unit and deploy them as a whole. Such coarse placement granularity lacks of flexibility and it also compromises the system efficiency.

We believe *a key reason causing the limitations above is that the existing solutions treat the content delivery and the content adaptation as two independent processes*. This makes: (i) the execution of the transformation applications cannot preserve the content's semantics, (ii) the intermediate nodes cannot smoothly scale up their processing capabilities along with the content delivery process, and (iii) the placement of applications does not fully consider the resulting system performance. It motivates us to propose the Application Networking framework, which provides an integrated and fine-grained delivery mechanism for the content objects as well as the relevant applications, to achieve pervasive, scalable, and efficient content delivery.

3.2 Framework Overview

The Application Networking framework is illustrated in Figure 3.2. Different from the traditional frameworks that only delivers the content object, our framework allows the origin site to deliver both the content object and the relevant content service logic to the intermediate network node. The service logic defines the operations to be performed on the content, the required application modules, and the conditions to transform the content. According to the service logic, the intermediate node can download the relevant applications and transform the content for the heterogeneous requests.

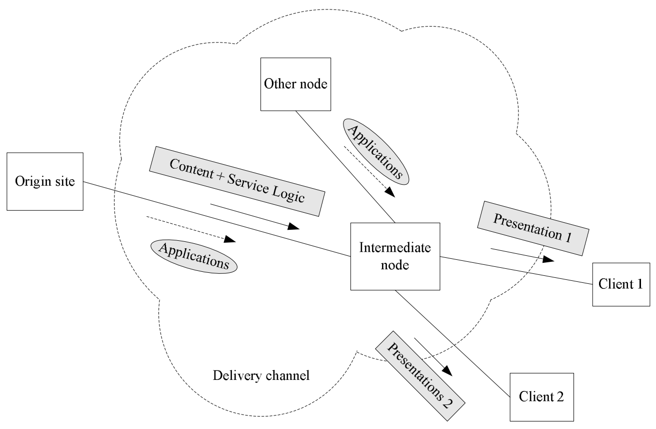


Figure 3.2 Application Networking Framework

Through delivering the content service logic, our framework integrates the content transformation process with the content delivery process. This change enables the framework to address the content's semantics, the system scalability and efficiency issues by means of (i) deliverable content service, (ii) compositional service formation, and (iii) flexible service deployment.

(1) Deliverable Content Service

Our framework deems a content object is correctly transformed if the semantics of the transformed content accords with the content provider's original intention, because we believe the content provider should determine how to present his content. To achieve this, we let the content provider specify the allowed content processing logic as a service and publish it with the original content object on the origin site. The origin site can deliver the content object and its service logic to the intermediate network node, which will use them to serve the client requests. Since the content service gives the operations authorized by the content provider, the node will generate the correct presentations that accord with the content provider's intention.

Though our framework proposes to use the deliverable service, it does not require all modules of the content service to be deliverable, because some modules need to be executed on the origin site due to the specific execution constrain or security concern. In actual systems, we can divide the entire service into two tiers: the static service tier and the deliverable service tier, as illustrated in Figure 3.3. The static tier performs operations like fetching content from disk, querying the database, and performing the backend business logic. The deliverable tier accepts the output content from the static tier and generates the content presentations for the clients. The static tier should be executed on the origin site, and the deliverable tier can be sent to the intermediate network nodes. In the following discussion, we only focus on the deliverable service tier and call it as the “content service”, and the content object got from the static tier is called as the “original content object”.

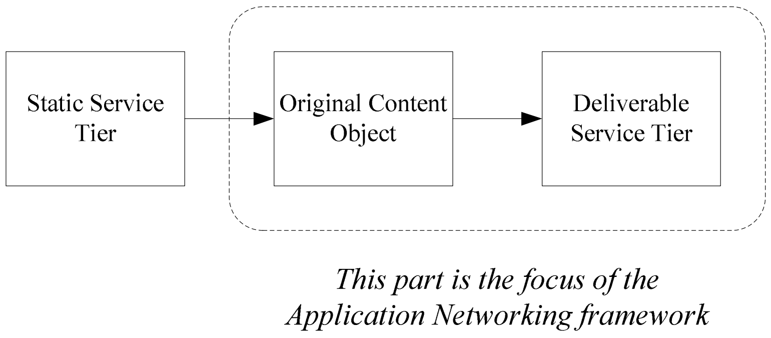


Figure 3.3 Static and Deliverable Service Tiers

(2) Compositional Service Formation

Our framework also supports compositional service formation. Firstly, we organize the content service as a “workflow” that decomposes the entire service into multiple sub-operations. This enables us to construct the service using the standard and reusable applications, each of which implements a sub-operation in the service. Secondly, the service workflow specifies addresses, which provides the required application modules. After the service is delivered, the intermediate node will be able to instantiate the service by downloading the needed applications from the specified addresses.

Combining above two aspects, we say our service is formed in the compositional way, which improves the system scalability because the intermediate nodes can scale up their capability for content processing along with the content delivery process.

(3) Flexible Service Deployment

Moreover, our framework provides a flexible service deployment approach, where the origin site can executes part of the service and delivers the generated intermediate response content with the unexecuted partial service to the intermediate node. This extends the traditional content service methodologies, which either execute all operations on the origin site or move all operations to the intermediate node. More importantly, this flexibility facilitates the Application Networking system to derive efficient delivery strategies by considering the utility and corresponding delivery overhead of different partial services.

The rest of this chapter will explain the workflow structure and a metadata based content reuse method, which form two basic components of the Application Networking framework and will be used in the concrete Web and P2P systems.

3.3 Workflow

3.3.1 Workflow Structure

The Application Networking framework uses the "workflow" to describe the logic of a content service. The workflow is composed of one or more segments (SEG) that are linked sequentially. Figure 3.4 presents an example workflow for the image provisioning service, where the first segment transforms the image according to the display preference of the client's device, and the second segment inserts a watermark into the image to identify its recipient.

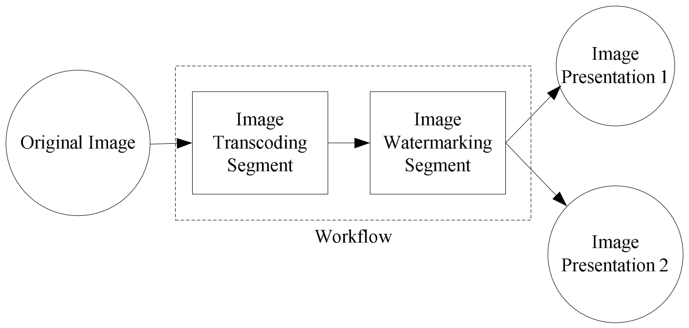


Figure 3.4 An Example Workflow

Each segment consists of several components, as given in Definition 3.1. The configure rule (*CR*) specifies the conditions to trigger the segment task. If the rule conditions are fulfilled, the segment task (*TSK*) should be performed and it will transform the input object (OBJ_{in}) to be the output object (OBJ_{out}). Otherwise, the task should not be performed, and the input object will be passed through the segment and become the output object directly. The *App* element specifies the application that implements the segment task, and it actually provides an address, from which the network node can download the corresponding application module. In addition, an input parameter vector (v) and the supplementary data entry (d) are usually needed for the execution of application.

Definition 3.1: Segment

$$SEG := OBJ_{out} = TSK^{CR,App}(OBJ_{in}, v, d)$$

The application (*App*) is a tool to realize a service workflow. The workflow, provided by the content provider, is to define “What to do” on a content object. The application is chosen by the executor to materialize “How to do” on the content object. The application can be downloaded from the address provided by the workflow or can be chosen by the executor itself given the new application is functionally equivalent to the original one.

For the image transcoding segment shown in Figure 3.4, it has the original image as the input object (OBJ_{in}), an image transcoding program as the application (*App*), and the transformed image as the output object (OBJ_{out}). The parameters (v) to the application include the screen size and the accepted image format of the client’s device. The configuration rule (*CR*) can be defined as detecting the mismatch between the original image and the client’s preference on the display dimensions and the image format. If any mismatch is found, the transcoding application will be executed.

A full workflow can contain multiple segments. The output object from the previous segment becomes the input object to the next segment. We rephrase the segment in a workflow as in the definition 3.2, where the entire workflow is formed by L segments. Varying the parameter v_i values, an original object can generate multiple output objects. OBJ_i denotes the output object from segment SEG_i and the input object to segment SEG_{i+1}. In a workflow, we name OBJ_0 as the original content object, the OBJ_L as the final content presentation, and the OBJ_i, where $1 \le i \le L-1$, as the intermediate content objects.

Definition 3.2: Segments in a Workflow

$$SEG_i := OBJ_i = TSK_i^{CR_i, App_i}(OBJ_{i-1}, v_i, d_i),\ 1 \le i \le L$$

Based on the discussion above, a workflow (*WF*) can be defined as in definition 3.3, where WF^L denotes the workflow has L segments. Feeding the original object OBJ_0 to the workflow, we can obtain the final content presentation OBJ_L through L-1 intermediate objects (from OBJ_1 to OBJ_{L-1}).

Definition 3.3: Workflow

$$WF^L := SEG_1 \bullet \cdots \bullet SEG_i \bullet \cdots \bullet SEG_L = (\bullet)_{i=1}^{L} SEG_i,\ 1 \le i \le L$$

Figure 3.5 summarizes the workflow structure.

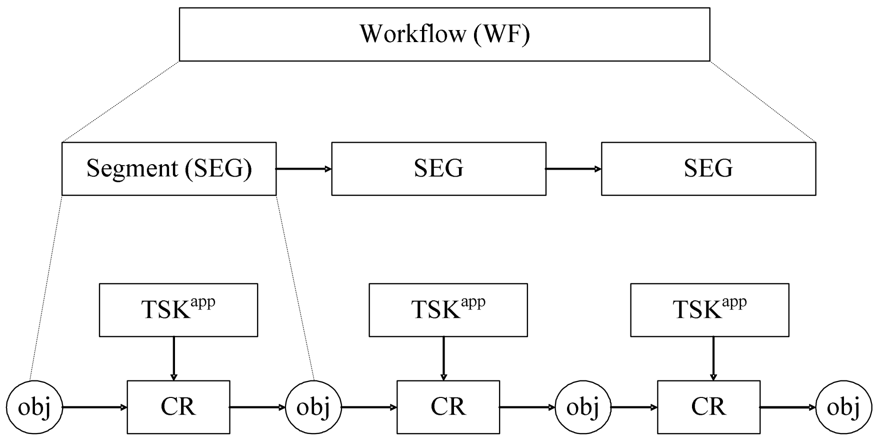


Figure 3.5 Workflow Structure

3.3.2 Workflow Operations

To implement flexible service deployment and instantiation, we define the following workflow manipulation operations.

(1) Workflow Partitioning

According to the workflow definitions, we can infer equations Eq.3.1 and Eq.3.2. The former equation indicates that using the first t segments we can generate an intermediate object OBJ_t from the original object OBJ_0, while the latter equation denotes feeding the intermediate object OBJ_t to the rear L-t segments we can obtain the final content presentation OBJ_L.

$$OBJ_t = (\bullet)_{i=1}^{t} SEG_i(OBJ_{i-1}, v_i, d_i) \qquad \text{(Eq.3.1)}$$

$$OBJ_L = (\bullet)_{i=t+1}^{L} SEG_i(OBJ_{i-1}, v_i, d_i) \qquad \text{(Eq.3.2)}$$

Based on these two equations, we define the workflow partitioning operation as in definition 3.4. The operation (F^t) divides a workflow (WF^L) into a front part, from SEG_1 to SEG_t, and a rear part, from SEG_{t+1} to SEG_L. The front partial workflow is executed on the origin site, while the rear part will be deployed and executed on the intermediate network node.

Definition 3.4 Workflow Partitioning

$$\begin{aligned} F^t(WF^L) &:= (SEG_1 \bullet \cdots \bullet SEG_t) + (SEG_{t+1} \bullet \cdots \bullet SEG_L) \\ &= (\bullet)_{i=1}^{t} SEG_i + (\bullet)_{j=t+1}^{L} SEG_j \end{aligned} \text{, where } 0 \le t \le L$$

The partitioning operation creates more possibilities for content service deployment through dividing the workflow at different segments. The partitioning has two extreme situations by setting the value of t to L and 0 respectively. When $t = L$, it means the entire service is executed at the origin site, which is equivalent to the traditional content service model where the origin site performs all operations and generates the final content presentations. When $t = 0$, the result is equivalent to the current active proxy solutions, where the origin server always delivers the original content to the proxy, which will transform the content.

More importantly, by setting $0<t<L$, our framework can deliver partial service with intermediate content object in the network, which is not provided by the

conventional frameworks. Instead of fixing the application deployment locations at the design time, our framework allows the network nodes to choose suitable deployment strategy dynamically. This decision is usually based on judging the performance gain and the corresponding deployment cost for different workflow partitioning possibilities. This flexibility enables the Application Networking systems to resolve the most efficient service placement strategy under different conditions. Concrete algorithms will be explained in later chapters.

(2) Application Substitution

Each workflow segment specifies an application to implement the segment task. In the segment specification, a location is provided for fetching this application module. However, the framework also allows the network node to substitute the original application module with another one. Such substitution operation can be expressed as in definition 3.5, where the original application (*App*) is replaced by another application (*App*').

Definition 3.5 Application Substitution

$$\begin{aligned} &F^{App'}(SEG) = F^{App'}(TSK^{CR,App}(OBJ_{in}, v, d)) \\ &:= TSK^{CR,App'}(OBJ_{in}, v, d) \end{aligned}$$

The purpose of application substitution has several aspects. Firstly, a network node usually has installed some popular applications. To avoid downloading redundant applications, we allow the node to reuse its local applications. Secondly, the network node may replace the original application with a better-customized one. For example, an edge proxy can replace the general advertisement insertion application with a customized one that posts the localized and targeted advertisements to the clients. Finally, the substitution operation allows a network node to load a usable application if the original application cannot be accessed, so that the node can still manipulate content at the expense of sacrificing some operation fidelity.

(3) Appending

Although our framework emphasizes on delivering the content service defined by the content provider, it also allows the clients or the third parties to perform supplementary operations on the content. The added operations are necessary when the content provider does not supply sufficient directives. However, the added operations may break the content integrity especially when the content already has the associated workflow. Suppose the origin site delivers a content object with a service workflow to the intermediate node. If this node performs some supplementary operations on the content before executing the original service workflow, the result could be unexpected because the input object for the service workflow has been altered by applications that are not in the original workflow. To avoid this, our framework requires the added operations can only be performed after the original workflow has been executed. Thus, we define the “appending operation” as in definition 3.6, where the supplementary segments (from SEG_{L+1} to SEG_{L+t}) are appended to the end of the original workflow WF^L (from SEG_1 to SEG_L).

Definition 3.6 Appending Operation

$$F^{L\rightarrow L+t}(WF^{L}) := ((\bullet)_{i=1}^{L} SEG_i) \bullet ((\bullet)_{j=L+1}^{L+t} SEG_j)$$
$$= (\bullet)_{i=1}^{L+t} SEG_i$$

3.3.3 XML Specification

Our framework specifies the workflow using the XML markup language. The XML description is platform independent, and it facilitates us to apply the Application Networking framework to different content delivery systems. Figure 3.6 shows the specification for the example workflow in Figure 3.4.

```
(01)  <Workflow owner=”abc.com”>
(02)     <Segment id=”transcoding segment”>
(03)        <Rule>
(04)           “client.display_w < img.display_w OR client.display_h < img.display_h”
(05)        </Rule>
(06)        <Task id="image-transcoding"/>
(07)           <App type="public">
(08)              <Codebase url="ftp://abc.com/applications/image-transcoder.class"/>
(09)              <Codebase url=”http://appnet.opensource.org/image-transcoder.class”/>
(10)           </App>
(11)         </Task>
(12)     </Segment>
(13)     <Segment id=”watermarking segment”>
(14)        <Rule>true</Rule>
(15)        <Task id="image-watermarking">
(16)           <App type="private">
(17)              <Codebase url="ftp://abc.com/applications/image-watermarking.class"/>
(18)           </App>
(19)        </Task>
(20)     </Segment>
(21)  </Workflow>
```
Figure 3.6 an Example Workflow XML Specification

The specification is rooted from a <Workflow> element, which has an attribute called “owner” (line 1) to identify the content provider. The workflow element includes a list of <Segment> elements, which should be written in their execution order. Every segment contains a <Rule> element and a <Task> element. The rule element defines the conditions (line 3-5) to trigger the task execution. We leverage the “p-rule language” [BR03] to specify the rules. This language was created for content adaptation, and it provides a set of concise yet descriptive rule declarations, which satisfy our requirements.

The task element contains a <App> element, which specifies the required application in the segment. The <App> element includes the <Codebase> elements that provide the URLs for downloading the application module. To avoid the single point of failure, multiple <Codebase> elements can be specified (lines 8-9). The network node can fetch from any of them. The specification also defines the “type” attribute for the <App> element, to control whether the original application can be substituted. If the value of “type” is “public” (line 7), the intermediate node is allowed to use another application to substitute the original application. If the value is "private" (line 16), it means the content provider requires the specified <Codebase> to be used and the intermediate node should not substitute it.

In addition, if a workflow is partitioned, partial workflow will be delivered. In this situation, the specification elements for the executed segments will be removed from the workflow specification, while the rear partial specification will be delivered with the generated intermediate content object (see Figure 3.10).

3.3.4 Discussion

In summary, the Application Networking framework expresses the content service as the loosely coupled workflow structure, which has several advantages:

(i) The workflow composes the content service using the independent and stateless segments. It enables the workflow to be delivered to the intermediate nodes and instantiated there using the downloaded applications. As the workflow carry the allowed content operations, the intermediate node can manipulate the content without violating the content provider's intentions.

(ii) Moreover, the delivered service directs the intermediate nodes to download new applications and lets these nodes to scale up their functionalities along with the content delivery process. This improves the system scalability in the open Internet environment.

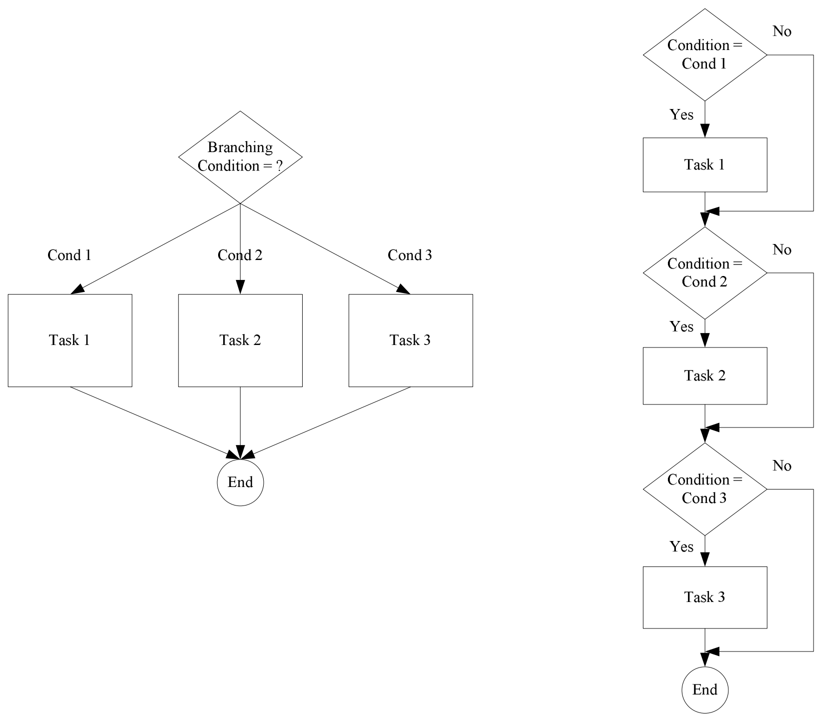
(iii) The workflow also allows the intermediate node to use its local applications to instantiate some segments. The local application's function should accord with the specified segment task. Such compositional service formation avoids downloading the redundant applications.

(iv) The loosely coupled workflow structure enables us to implement flexible service deployment by partitioning the workflow in suitable ways. This flexibility enables the Application Networking systems to derive efficient content delivery strategies.

We acknowledge there are many other workflow structures. A typical example is WS-BPEL [Bpel] that originates from WSFL [Ley01] and XLANG [Tha01]. The WS-BPEL was proposed in the SOA (Service Oriented Architecture) and it describes the business process at the orchestration level. The specification provides the full-featured business process descriptions, including service flow, branching, looping, transaction, and error handling. In comparison, our workflow is much simpler. This is because our workflow is used to describe the content services, which are usually expressed as several operation steps. The content services are not likely to involve transactions, so that the transaction concept is not included in our structure.

The loop logic is possible in the content services, such as an image is transformed by several rounds to fit the client's requirements on the image dimensions, color, resolution and media encoding. However, we intend to let the specific application encapsulate such looping logic into its implementation. Thus, we can keep the workflow loosely coupled, in that each segment is independent with the others and the execution of all segments are stateless. This facilitates the system to implement the deliverable service and the composite service formation.

In addition, though our workflow arranges all segments in sequence, the usage of configuration rules allows us to implement the branching logic in the service. This is achieved by specifying mutually exclusive rule conditions for a group of segments. Then the sequential execution of these segments will render the branching effect over them. This effect is illustrated in Figure 3.7, where the right side shows a workflow.



Branching Logic　　　　Workflow Structure

Figure 3.7 Branching Logic in the Workflow

Regarding the aspects above, we can conclude that the Application Networking workflow provides a simple yet expressive structure to describe most content services. In addition, in the Application Networking framework, the content provider is responsible for creating the workflow specification. Besides writing the XML specification from scratch, a more convenient way is to harness the authoring software to simplify the specification work, such as providing the graphical user interface, enabling workflow customizing based on templates, and reusing workflow for different contents. In this way, we will not cause too much burden on the content providers.

3.4 Metadata based Content Reuse

Our framework makes different versions of content to be delivered, including the original content object, the intermediate content objects, and the final content presentations. Identifying these objects is important for efficient content reuse.

Nevertheless, this issue is not well addressed by the traditional frameworks that were mostly designed for contents with a single presentation. Some adaptive solutions, such as Transquid [MSR01] and PTC [STR02], can cache and reuse the specific types of content. Although such type-oriented approach can achieve precise data reuse but also it requires the network node to install dedicated applications to extract the content's properties, so that the apporach can only be applied to those popular content types. Meanwhile, the content reuse judgment usually involves media decoding operation, so that it is heavy weight.

Our framework is built to be a general platform, which needs to handle different types of contents. The content identifying and reusing scheme should be scalable and lightweight. To achieve this target, we propose a metadata-based approach to identify and reuse content objects in a systematical way.

3.4.1 Metadata Specification

Our method associates one piece of metadata to the content object. The metadata is the aggregation of the object's identifier and zero or more attributes, as given by definition 3.7. The "id" gives the unique identifier (e.g. URI) of the content object. Each attribute entry (*attr*) consists of the attribute name and value. For instance, a text document may have an attribute as (language, "English") to specify its presenting language. Different versioned objects of the same content share the same identifier, but are differentiated through the attributes in the metadata.

Definition 3.7: Object's Metadata

$$\text{Meta}(OBJ) := \{id, \{attr_1, attr_2, \ldots, attr_j\}\}, \text{ where } attr = (name, value)$$

The default metadata for the original content object only contains the identifier with some initial attributes. When the object is transformed, the transforming application is responsible for inserting additional attributes into the metadata or updating the values of the affected attributes. For the workflow in Figure 3.4, we assume the metadata for the original image includes the identifier and the dimensions of the image. When the transcoding application is executed, it will update the dimension attribute with the new image size. Finally, when the watermarking application is executed, it will append a new attribute to declare the content's recipient.

The content metadata is specified in the XML format. An example is shown in Figure 3.8. The metadata is rooted with the <Metadata> element. The "id" attribute of <Metadata> declares with which content this metadata is associated. The <Metadata> contains none or multiple <Attr> elements, each of which defines an attribute entry of the content object.

```
<Metadata id="http://abc.com/breakingnews.jpeg">
   <Attr name="display_w" type=”int”>352</Attr>
   <Attr name="display_h" type=”int”>188</Attr>
   <Attr name=”type” type=”string”>JPEG</Attr>
</Metadata>
```

Figure 3.8 Metadata Specification

Since our framework can deliver both content and the associated service workflow, a <AppNetDeliverable> element is used to encapsulate the metadata of the delivered content and its associated workflow specification. Figure 3.9 shows the specification for our example workflow in Figure 3.4. The content metadata is provided in line 2 and the workflow is given in lines 3~23. Note the metadata only contains the content's identifier which means this original content with the entire service workflow can be reused for all requests for the content. As an option, <AppNetDeliverable> can contain a <Payload> element (lines 24~26) to include the payload of the delivered content object, whereas specific systems can use their own methods to transmit the content payload, such as the HTTP multipart types [HTTP] or the message attachments [Soap-a].

```
(01)  <AppNetDeliverable>
(02)     <Metadata id="http://abc.com/breakingnews.jpeg"/>
(03)     <Workflow owner=”abc.com”>
(04)        <Segment id=”transcoding segment”>
(05)           <Rule>
(06)              “client.display_w < img.display_w OR client.display_h < img.display_h”
(07)           </Rule>
(08)           <Task id="image-transcoding"/>
(09)              <App type="public">
(10)                 <Codebase url="ftp://abc.com/applications/image-transcoder.class"/>
(11)                 <Codebase url=”http://appnet.opensource.org/image-transcoder.class”/>
(12)              </App>
(13)            </Task>
(14)        </Segment>
(15)        <Segment id=”watermarking segment”>
(16)           <Rule>true</Rule>
(17)           <Task id="image-watermarking">
(18)              <App type="private">
(19)                 <Codebase url="ftp://abc.com/applications/image-watermarking.class"/>
(20)              </App>
(21)           </Task>
(22)        </Segment>
(23)     </Workflow>
(24)     <Payload coding=”base64”>
(25)        Original image payload is included here
(26)     </Payload>
(27)  </AppNetDeliverable>
```
Figure 3.9 Specification for <AppNetDeliverable> - Entire Workflow

As mentioned, our framework can also deliver the intermediate content object with the partial service workflow. In this case, the specification for the executed segments needs to be removed and only the specification for the remaining workflow segments is kept. Figure 3.10 shows an example based on Figure 3.9, where we assume the transcoding application has resized the original image. The specification contains the metadata for the resized image (lines 2~5) with the remaining watermarking segment (lines 6~15). Note the transcoding application has added two metadata attributes for the resized image (lines 3&4). The resized image is encapsulated in the <Payload> element.

```
<AppNetDeliverable>
  <Metadata id="http://abc.com/breakingnews.jpeg">
    <Attr name="display_w" type="int">352</Attr>
    <Attr name="display_h" type="int">188</Attr>
  </Metadata>
  <Workflow owner="abc.com">
    <Segment id="watermarking segment">
      <Rule>true</Rule>
      <Task id="image-watermarking">
        <App type="private">
          <Codebase url="ftp://abc.com/applications/image-watermarking.class"/>
        </App>
      </Task>
    </Segment>
  </Workflow>
  <Payload coding="base64">
    Resized image payload is included here
  </Payload>
</AppNetDeliverable>
```

Figure 3.10 Specification for <Deliverable> - Partial Workflow

3.4.2 Content Reuse

Our framework supposes the client's preferences and the relevant request parameters are carried by the request message. When the network node receives a request, it first finds a cached object whose identifier equals to the requested identifier. Then it needs to determine whether this object is reusable for the present request. We treat an object is "reusable" if it is the exact presentation requested by the client, or it can be transformed to the requested presentation using the associated service workflow. In the framework, such reusability judgment is based on matching the content's metadata and the request's attributes.

A possible way for the reusability judgment is the "perfect matching" method, which demands all attributes of the object's metadata and all preference entries and parameters of the request to match exactly. However, this method is too strict to achieve good content reuse, as any difference between the request and the stored content can cause the negative reuse decision. In the open Internet environment, since both the content provider and the client may use some attributes that are not used by the counterpart, the perfect matching could cause the matching process to be over negative.

To avoid this situation, we adopt an alternative method – the “best-effort matching”. This method treats an object is reusable if all the “*common*” attributes of the stored content object and the request match with each other. Suppose a document object has an attribute of {(language, English)} and the client’s request contains three parameters as {(language, English), (display_w, 800), (display_h, 600)}. In this situation, the framework will deem the document to be reusable, because its sole attribute (i.e. language) matches the corresponding parameter in the request, while the other parameters (i.e. display_w and display_h) are ignored since they are not presented in the document’s attributes. Using this best-effort matching policy, we intend to reuse objects to the maximum extent while avoiding those attributes with mismatching values.

According to the previous discussion, when a content object is processed by more workflow segments, more attributes will be added to the object’s metadata and fewer segments will remain in workflow specification. It implies the content object with the associated partial service is becoming more and more specific because more its metadata attributes haven been fixed and the content’s reusability reduces according to our best-effort matching mechanism.

3.4.3 Discussion

Our metadata based content reuse method has several advantages:

(i) The metadata is expressed in the XML. Such system neutral format allows a network node to derive the content’s reusability without interpreting its data format or understanding any application-specific semantics. It makes our metadata method to be general purposed, applicable to different content types.

(ii) Our metadata method can also implement service-oriented content reuse. This is because the content metadata is maintained and updated by the content service applications, and the content reusability judgment is made based on such service-oriented metadata. Thus, we can achieve the service-oriented content reuse, which is not provided by the current type-oriented reuse methods.

(iii) Our method is also lightweight, as the reuse judgment only involves matching the metadata attributes, without needing to decode the content payload to extract the wanted media property.

3.5 Observation and Summary

There are some other intermediary frameworks and one representative is the Open Pluggable Edge Service (OPES) framework [Opes]. OPES [BCH04] provides a general framework that allows the intermediate network nodes to perform value-added operations on the transferred content data. OPES proposes many concepts such as the Proxylet application interface [Wal01], the IRML [BH01] and the P-language [BR03] rules to describe the application invocation conditions, and the OCP callout protocol to access the remote service [Rou05].

Similar to the OPES framework, our Application Networking framework also allows the intermediate network nodes to perform active content services. Our framework leverages some available results from OPES, such as using the P-language to describe the configuration rules in the workflow, and referring the Proxylet specification to define the application APIs (Chapter 4 and Chapter 7). However, our framework differs from OPES in several key aspects.

The most significant difference is that our framework integrates the content delivery and the application deployment into an integrated process. However, the OPES framework treats them as two separate processes, where the pre-deployment strategy [YH01] is used. Due to this difference, our framework allows the intermediate network nodes to scale up their content processing capabilities along with the content delivery process, by downloading new applications dynamically. Moreover, we use the loosely coupled workflow to express the deliverable service, which facilitates the system to implement fine-grained service deployment.

Another difference is that content reuse issue is addressed in our framework, but not in the OPES framework. We let the network nodes to store and reuse the versioned content objects with the associated service. The metadata-based approach is used to achieve the general-purposed and service-oriented content reuse.

In summary, the Application Networking framework, though leveraging some parts from the existing OPES framework, enriches the active intermediary methodology through enabling deliverable content service, compositional service formation, flexible service deployment, and fine-grained content reuse. The main contributions of the framework are (i) it integrates the content delivery and transformation processes as a unified service delivery process; (ii) it broadens the content delivery and reuse spectrum by including not only the original content and its final presentations but also the intermediate content objects.

However, applying the Application Networking framework into the existing content delivery contexts is an intensive exercise. There are many architectural and performance issues to be considered. In the rest of this thesis, we will address more details on building the App.Net system that applies the framework to the Web system, and the AN.P2P system that extends the framework to the P2P networks.

CHAPTER 4

APP.NET - APPLICATION NETWORKING ON THE WEB

4.1 App.Net Architecture

This chapter explains our App.Net system, which applies the Application Networking framework in the Web content delivery context. The system architecture is shown in Figure 4.1, where the App.Net platform is installed on both the origin server and the edge proxy.

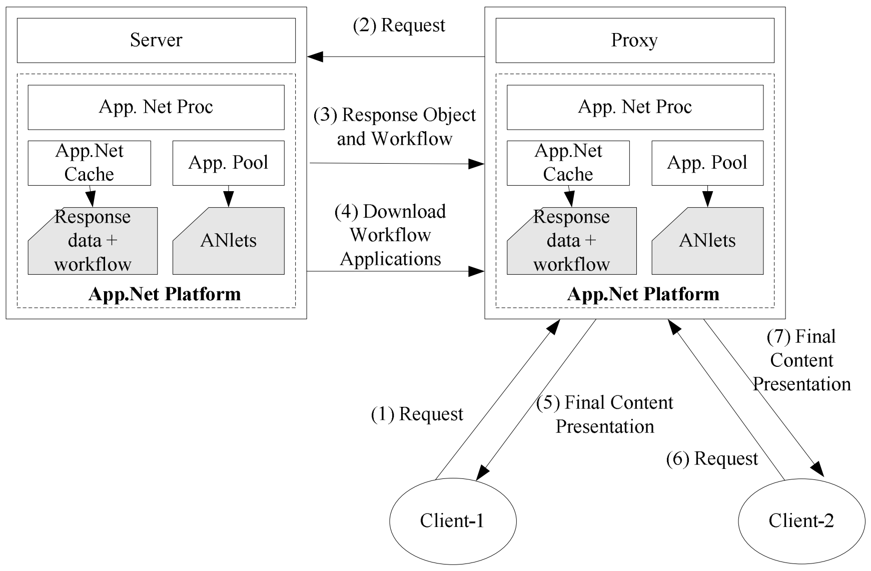


Figure 4.1 Architecture of App.Net System

The App.Net platform has three major components:

(1) App.Net Proc: is the kernel process of the platform. It handles the HTTP requests and replies, and performs necessary operations on them.

(2) App.Net Cache: is to store the HTTP responses, which include the response content objects and their associated workflow specifications.

(3) App Pool: is to store the downloaded or locally installed application modules.

The detailed system operations can be described in four processes: (i) service preparation, (ii) request forwarding, (iii) server response, and (iv) proxy response.

4.1.1 Service Preparation Process

When the content provider publishes the original content on the server, the App.Net system allows him to attach a content service to the published content object. This service is specified as a workflow, as explained in our framework. Meanwhile, the server should have the relevant applications to implement the segment tasks. In the App.Net system, we define a general interface – Anlet – for the workflow applications, so that the system can execute different applications through a set of uniform functions. Since the Web content service may contain the content generation step and the content transformation steps, we divide the ANlet interface into two sub-interfaces: ANlet Handler and ANlet Filter.

ANlet Handler is used for the content generation application, which should reside at the beginning of a workflow. In this interface, we define the *generate_Response* function, as given by Api.4.1, which allows the application to generate the initial content object. The Handler application functions as the traditional presentation layer script (e.g. JSP or ASP) that generates the dynamic Web contents by querying the backend database or business logic result. However, a major feature of ANlet Handler is that it can be delivered from the server to the proxy and be executed there. We can use the Handler application to implement the "mobile presentation tier" of a Web site, which cannot be provided by the traditional presentation layer scripts.

Api.4.1: ANlet Handler Execution Function

ContentObject generate_Response(*HTTPRequest request*)

ANlet Filter interface is used for the content transformation applications. In this interface, we define the *modify_Response* function, as given by Api.4.2, which allows the application to transform the content object. The Filter application receives the content object and the client's request message as input, and it will transform the content object according to the request's parameters and output the new content object. For a workflow, multiple Filter applications will be executed in sequence, and they will transform the original object to be the final content presentation.

Api.4.2: ANlet Filter Execution Function

ContentObject modify_Response(*ContentObject input_obj*, *HTTPRequest, request*)

In our system, a workflow can start from a normal content object (e.g. a text file or an image) or a Handler application, as illustrated in Figure 4.2. We treat the Handler application as an active object that needs to be executed to get the content data. Having this particularity, the following discussion will not deliberately distinguish whether the workflow is started from a normal content object or a Handler application, unless it is necessary. For concise expression, we just say any workflow starts from an original content object.

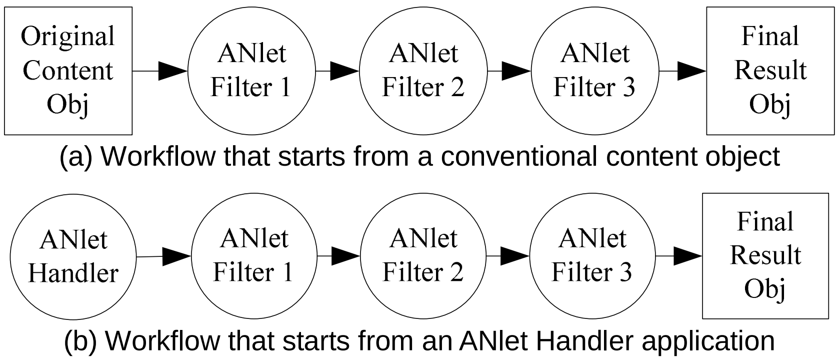


Figure 4.2 App.Net Workflow

4.1.2 Request Forwarding Process

When a client requests the content on the server, the request contains not only the name of the Web resource but also the relevant request parameters, such as the service-specific parameters or the client's preference. The App.Net system allows the request message to carry these parameters in either the query string or the header fields of the HTTP message. An example request is shown in Figure 4.3. According to HTTP 1.1 [Http], "http://abc.com/images/pic.jpg?display_w=352&display_h=288" is the Request-URI, where "http://abc.com/image/pic.jpg" is the Web resource identifier and "display_w=352&display_h=288"is the query string. All the following fields, from line 03 to line 06, are the request headers of the HTTP message.

```
(01)  GET http://abc.com/images/pic.jpg?display_w=352&display_h=288 HTTP/1.1
(02)
(03)  Host: abc.com
(04)  Accept: text/*, audio/*, img/png
(05)  User.network: ADSL
(06)  User.profile.template: www.xyz.com/profile/device.xml
```

Figure 4.3 Request Message from Client

Using query string to carry the request parameters is a traditional technology. It lets the content service specify parameters for the client by rewriting the HTML hyperlinks or the URLs of the embedded objects. When the client clicks these hyperlinks or fetches the embedded objects, the parameters will be automatically included into the HTTP query string.

In the App.Net system, we also allow the browser to insert the client's preference into the HTTP request as headers. HTTP 1.1 provides some default request headers, whereas extended headers can also be inserted. For example, in Figure 4.3, the message uses the standard "Accept" header (line 4) to declare the media types accepted by the client. It also contains two extended headers: the first header, "user.network" (line 05), specifies the client's network connection type, and the second header, "user.profile.template" (line 06), provides the URL for fetching the profile [Ccpp and Uapr] of the client's device.

When the client's request reaches the App.Net proxy, the proxy will insert an extended HTTP header, "Via-AppNet" into the request (line 07 Figure 4.4), and forward the request to the server.

```
(01)   GET http://abc.com/images/pic.jpg?display_w=352&display_h=288 HTTP/1.1
(02)
(03)   Host: abc.com
(04)   Accept: text/*, audio/*, img/png
(05)   User.network: ADSL
(06)   User.profile.template: www.xyz.com/profile/device.xml
(07)   Via-AppNet: true
```

Figure 4.4 Request Message from Proxy

The BNF definition of the Via-AppNet header is given in Definition 4.3. It enables the proxy to declare that the request is forwarded by an App.Net enabled proxy, so that the server can deliver workflow in the response. If the request is forwarded by only the conventional proxies, the server will not see this header in the request, and it should fully execute the content service and put the final content presentation in the response. Thus, the usage of the Via-AppNet header makes the App.Net enabled server and proxy to be compatible with the conventional proxy.

Definition 4.3: Via-AppNet

Via-AppNet = "Via-AppNet: true"

4.1.3 Server Response Process

When the server receives the request, it can generate the response in one of three operation modes:

(1) Full transformation: the server executes all workflow applications associated with the requested content to generate the final content presentation for the client and sends this final presentation to the proxy.

(2) Non-transformation: the server directly sends the original content object and its associated workflow specification to the proxy.

(3) Partial transformation: the server partitions the workflow into two parts and feeds the original content to the applications of the front part workflow to generate an intermediate response object. Then, the server returns the intermediate object and the specification for the rear part workflow to the proxy.

According to Chapter 3, any operation mode above can be described as a workflow partitioning possibility. The server should determine how many workflow segments to be executed and how many segments to be delivered to the proxy. We call this judgment process as the "*service placement*". The server chooses the service placement strategies according to the performance optimization and security enforcement requirements.

After getting the response content object, the server will construct an HTTP response message to the proxy. An example response is shown in Figure 4.5. The message is tagged with "Content-Type: App-Net-Workflow" (line 04) to indicate this response includes both the content object and the workflow specification. The response body is coded as XML format, which contains three elements: <Metadata>, <Workflow> and <Payload>, as explained in Chapter 3.

In the actual network, above compositional response message may be cached by the conventional proxy that does not support App.Net. This situation is likely to happen when the conventional proxy resides between an App.Net server and an App.Net proxy. When this conventional proxy transmits the response from the App.Net server to the App.Net proxy, it will cache the response. As this proxy cannot interpret the response correctly, it would wrongly send it to the clients or other conventional proxies, which definitely cannot understand the compositional response. To avoid such situation, the App.Net server inserts a Vary header with value

"Via-AppNet" into the response (line 06 in Figure 4.5). According to the HTTP protocol, a cache cannot serve a cached response with a "Vary" header unless the new request contains the same header, which is specified in the "Vary" header of the cached response, as the origin request. Because only the App.Net proxy will insert the "Via-AppNet" header into the request, the usage of "Vary: Via-AppNet" header in the response makes sure this response can only be sent to the App.Net enabled proxies, even if the response is cached by a conventional proxy.

```
(01)  HTTP/1.1 200 OK
(02)
(03)  Date: Sun, 25 Apr 2004 06:25:24 GMT
(04)  Content-Length: 26012
(05)  Content-Type: text/App-Net-Workflow
(06)  Vary: Via-AppNet
(07)
(08)  <?xml version="1.0" encoding="UTF-8"?>
(09)  <!DOCTYPE AppNet SYSTEM "http://comp.nus.edu.sg/~sumu/AppNet.dtd">
(10)  <AppNetContent>
(11)     <Metadata id=" http://abc.com/images/pic.jpg">
(12)        <Attr name="display_w">352</Attr>
(13)        <Attr name="display_h">288</Attr>
(14)     </Metadata>
(15)     <Workflow owner="abc.com">
(16)        <Segment id="watermarking segment">
(17)           <Rule>true</Rule>
(18)           <Task id="image-watermarking">
(19)              <App type="private">
(20)                 <Codebase url="ftp://abc.com/applications/image-watermarking.class"/>
(21)              </App>
(22)           </Task>
(23)        </Segment>
(24)     </Workflow>
(25)     <Payload type="img/jpeg" coding="base64">
(26)        The image payload is here, using the base64 coding
(27)     </Payload>
(28)  </AppNetContent>
```

Figure 4.5 Response Message from Server

4.1.4 Proxy Response Process

When the proxy receives the response from the server, it first checks the content type of the response. If the type is not "App-Net-Workflow", it means the response only contains the content object, and the proxy will send the response to the client directly. However, if the type is “ App-Net-Workflow”, the proxy will execute the delivered workflow to generate the final presentation for the client.

In specific, to perform the workflow tasks, the proxy needs to load the relevant applications used by the workflow. As shown in Figure 4.5 line 20, the workflow specification provides the addresses to download the ANlet applications. By default, the proxy will fetch the applications from these addresses. Our system also allows the proxy to substitute some original applications by using its locally installed applications that perform the equivalent functionalities. After loading all applications, an instance of workflow is created. Then, the proxy feeds the response content object to the instantiated workflow, executes the applications with respect to the workflow specification, and generates the final content presentation according to the client’s parameters. In the end, the proxy sends the response to the client, as in Figure 4.6.

```
(01)  HTTP/1.1 200 OK
(02)
(03)  Date: Sun, 25 Apr 2004 06:25:28 GMT
(04)  Content-Length: 62920
(05)  Content-Type: img/jpg
(06)
(07)  The final image payload is here
```

Figure 4.6 Response Message from Proxy

Meanwhile, the proxy stores the received response message into the App.Net Cache, and stores the downloaded ANlet applications into the App. Pool. When the proxy receives a new request, it will search the local cache for a reusable response and serve the new client directly. The cache reuse details will be explained later.

4.1.5 System Security

Some restrictions are put on the ANlet to enforce the security of the system.

(1) The system treats the locally installed applications and the downloaded applications differently. The local application can access most system resources. However, the downloaded application can only manipulate the content object passed

to it, and it is restricted from accessing the file systems or executing system-level commands (e.g. scanning port status, etc).

(2) We restrict the downloaded application from connecting to the third-party site other than the origin server. In this way, the client's information will only be propagated to the original server he requested, and the retrieved data is solely from the original server which prevents the application from inserting unexpected third-party information into the response content.

Other security mechanism can also be leveraged in the App.Net system, such as application sandbox, secured content adaptation [LRY02], and the distributed server-proxy authentication. However, they are not the focus of this thesis.

4.2 App.Net Caching Scheme

Caching and reusing the response objects on the proxy is an important way to improve the user perceived latency and reduce the network backbone traffic. The traditional Web caching scheme is based on the HTTP 1.1 specification [Http]. However, this scheme has limitations to identify the versioned responses that are delivered in the App.Net system. In this section, we first examine two aspects of content caching: (i) cache identifier and (ii) versioned response, and then propose the App.Net caching solution.

4.2.1 Cache Identifier

The traditional caching scheme uses the Request-URI to identify the response. An example is "http://website.com/image.jpeg?display_w=352&display_h=288", where the front part, "http://website.com/image.jpeg", gives the Web resource name and the rear part, "display_w=352&display_h=288", is called query string and it carries the request parameters. The cached response will be reused if the new request has the same Request-URI as the cached response.

However, in the App.Net system, using Request-URI as the cache identifier may reduce the reusability of some responses. As explained, the App.Net server can deliver the final content presentations as well as the original or intermediate responses, and all of them may be cached by the proxy.

The final presentation is particularly for one group of requests with the exactly same Web resource name and query string parameters, so that only the exactly matched requests can reuse the cached final content presentations. In this case, the Request-URI can be used as the cache identifier.

However, the response containing original or intermediate object can be reused for not only the exactly matched requests but also requests for the same Web resource but with different query string parameters. In these cases, using Request-URI as the cache identifier would improperly reduce the reusability of such responses, since the full characters match binds the reusability of the original or intermediate response with the query string parameters in the original request. For example, an original request has the Request-URI as “http://website.com/image?display_w=352&display _h=288”, where the query string specifies the preferred display size for the client who issued this request. Suppose the server delivers the original image and the image adaptation workflow to the proxy, which caches this response and identifies it using the original Request-URI above. Now, another client requests this image but with a different display size, and the new Request-URI is http://website.com/image? display_w=176&display_h=144. In this case, the proxy cannot reuse the cached response, as the new Request-URI does not match the original Request-URI exactly. As the result, even though there is a reusable response, the proxy deems it as not reusable.

To solve this problem, our system adopts an extended response header, RE-URI (i.e. URI for the Response Entity), to identify the response. The value of RE-URI is usually a string prefix of the Request-URI by removing some or all query string parameters those are irrelevant to specify the current response. The BNF definition for RE-URI is given in Definition (4.4).

Definition 4.4: RE-URI

RE-URI="http://"host[":"port][path["?"query]]

In our caching scheme, RE-URI takes precedence over the Request-URI as the response's identifier. If the response contains an RE-URI header, the App.Net cache will use it as the identifier; otherwise, the original Request-URI will be used. Figure 4.7 shows a pair of request and response, where a client requests an image that will be displayed on his PDA device (figure (a)) and the server delivers the original image with the associated workflow (figure (b)) to the proxy. The response has an RE-URI header http://website.com/image, which removes the query string parameters in the original request. When the proxy receives this response, it will use the RE-URI field to identify the cached response.

When the proxy performs a cache lookup for the new request, the identifier of the cached response should exactly match or be a string prefix of the new Request-URI. According to this rule, the proxy can reuse the cached image object and its workflow to serve new requests for this image but with different display size, such as http://website.com/image?display_w=176&display_h=144.

```
GET http://website.com/image?display_w=352&display_h=288 HTTP/1.1

Host: website.com
Accept: text/*, audio/*, img/*
Via-AppNet: true
```

(a) Request Message

```
HTTP/1.1 200 OK

Date: Sun, 25 Apr 2004 06:25:24 GMT
Content-Length: 3126012
Content-Type: text/App-Net-Workflow
RE-URI: http://website.com/image

<?xml version=”1.0” encoding=”UTF-8”?>
<!DOCTYPE AppNet SYSTEM “http://comp.nus.edu.sg/~sumu/AppNet.dtd”>
<AppNetContent>
   <Obj type=”img/jpeg” encoding=”base64”>
      The image payload is here, using the base64 coding
   </Obj>
   <Workflow owner=”website.com”>
      Workflow segments are included, for details see Chapter 3.
   </Workflow>
</AppNetContent>
```

(b) Response Message with RE-URI Header

Figure 4.7 Example Messages for the RE-URI Header

4.2.2 Versioned Response

The traditional caching scheme uses a “Vary” response header, to distinguish different versions of response, such as a Web page can have a Chinese version and an English version. HTTP 1.1 gives the BNF definition for “Vary” as in Definition 4.5. The protocol specifies the value of Vary header indicates the set of request-header fields that are selected to determine whether the cache is permitted to use the response to reply to a subsequent request – the cache will reuse a cached response if all of the selected request-headers presented in the new request match the corresponding headers in the original request.

Definition 4.5: Vary

Vary = “Vary”“:”(“*”|1#field-name)

For example, if the original request contains header “Accept-Language: en” and the response contains header “Vary: Accept-Language”, the cached response can only be reused for the subsequent requests that also have header “Accept-Language: en”. Supposing a new request carries header “Accept-Language: fr”, the cached response should not be reused.

However, in the App.Net system, “Vary” is not sufficient to distinguish the versioned responses in all situations, because some request parameters may locate in the query string. Therefore, we propose an extended response header, VaryByParam, whose BNF definition is given in Definition 4.6. The header field indicates the set of parameters, resided in the query string, selected to identify a particular version of response. The cache will reuse a cached response if all the query string parameters selected by the VaryByParam header have the same values in the new request and the original request.

Definition 4.6: VaryByParam

VaryByParam=“VaryByParam”“:”(“*”|1#para_name)

Figure 4.8 shows a pair of request and response. The Request-URI (Figure 4.8 (a), line 1) is “http://website.com/image?s=bob207&display_w=352&display_h=288”, where the first parameter “s=bob207” gives the session id and the last two parameters “display_w=352&display_h=288” give the client’s preferred display sizes. The session id is normally used for tracing a client’s transaction and has no effect on the image presentation. We suppose the server transcode the original image according to the client’s display size and put the resized image in the response. In the traditional Web cache, the Request-URI is used as the identifier and this would result in that the response has no reusability on the proxy because the non-repeatable session id causes the constant mismatch between the cache identifier and the new Request-URI. However, as shown in Figure 4.8 (b), the App.Net server inserts the RE-URI header (line 6) to identify the response, and uses the VaryByParam header (line 7) to declare that the query string parameters display_w and display_h are selected to specify the version of current response object. Thus, this cached response can be reused to a request like "http://website.com/image?s=alice422&display_w=352&display_h=288", where the unselected query string parameter s=alice422 is ignored.

```
(01)   GET http://website.com/image?s=bob207&display_w=352&display_h=288 HTTP/1.1
(02)
(03)   Host: website.com
(04)   Accept: text/*, audio/*, img/*
(05)   Via-AppNet: true
```

(a) Request Message

```
(01)   HTTP/1.1 200 OK
(02)
(03)   Date: Sun, 25 Apr 2004 06:25:24 GMT
(04)   Content-Length: 3126012
(05)   Content-Type: img/jpeg
(06)   RE-URI: http://website.com/image
(07)   VaryByParam: display_w, display_h
(08)
(09)     The image payload is here, using the base64 coding
```

(b) Response Message with RE-URI and VaryByParam Headers

Figure 4.8 Example Messages using VaryByParam

In summary, through using the standard "Vary" header and the extended "VaryByParam" header, we can distinguish various response versions according to their HTTP message headers as well as the query string parameters.

4.2.3 App.Net Caching Scheme

The App.Net Cache retrieval policy can be summarized as follows:

(1) When the cache performs a lookup, it firstly checks the identifier of the cached response. If the identifier is got from the Request-URI of the original request, it should exactly match the Request-URI of the new request. If the identifier is got from the RE-URI header of the response, it should be a string prefix of the new Request-URI.

(2) Once the identifier checking succeeds, the cache will make sure the cached response is the reusable version for the current request by checking the "Vary" and "VaryByParam" headers. The cached response cannot be reused unless all

of the selected request-headers (or query string parameters) presented in the new request match the corresponding header fields (or query string parameters) in the original request.

If all criterions above are satisfied, the App.Net Cache will reuse this cached response.

In addition, the App.Net cache leverages the conventional replacement algorithms to replace the cached responses. Typical algorithms [CI97] include LRU, LFU, Greedy-Dual, and so on. Besides replacing objects due to the spatial limitation, the cache also evicts responses when an ancestor response object of them is to be stored, since the evicted objects can be generated from the new response object. This mechanism can save the limited cache space to store more objects. At the result, the cache can cover more requests and consequently improve the hit ratio. In particular, we can determine the new response object is an ancestor of a cached response object through the following steps.

(1) The identifier of the new response object should be the same as or a string prefix of the identifier of the cached response.

(2) If the new response contains the “Vary” header, all the selected request-headers should also exist in the “Vary” header of the cached response. Each selected request-header’s value for the new response should equal to the corresponding request-header’s value for the cached response.

(3) If the new response has the “VaryByParam” header, all the selected query string parameters should also exist in the “VaryByParam” header of the cached response. Each selected parameter’s value for the new response should equal to the corresponding parameter for the cached response.

If all these criterions are satisfied, we deem the new response object to be the ancestor of the stored response object. For example, we suppose the cache already stores the response as shown in Figure 4.8. Now a new response, as shown in Figure 4.7, is to be stored into the cache. The cache can determine the new object is the ancestor of the cached object based on the judgment criterions above.

4.3 Performance

A major advantage of the App.Net system is its flexible service placement mechanism. Such flexibility enables the system to achieve efficient content delivery strategy considering the potential performance gain and the corresponding delivery cost of content service. In this section, we first define the cost model for a single application, and then propose a optimization model for the workflow with multiple applications. Finally, an algorithm to resolve the optimal service placement is provided.

4.3.1 Cost Model

The Web server and the edge proxy are usually located remotely, so that the limited network transmission capability mostly affects the overall content delivery performance. We believe the network transmission cost should be a significant performance factor and it is not easily to be bypassed, whereas other factors like the computation load could be addressed by enriching local hardware resource. In general, less transmission cost implies less data is transmitted over the network and less content retrieval delay. Therefore, our model uses the “transmission cost”, the amount of content transmission traffic averaged over the time, as the performance metric.

Placing and executing an application on the server and the proxy would result in different system performance. According to the definition in Chapter 3, we can illustrate the structure of an application as in Figure 4.9, where an object o_0 is inputted to the application *f* and *K* distinct objects o_j ($1 \leq j \leq K$) are outputted. Each output object o_j has size s_j and lifetime t_j. In addition, the application may need supplementary input data, *d*, during execution. We denote the size and lifetime of the supplementary data as s_d and t_d respectively.

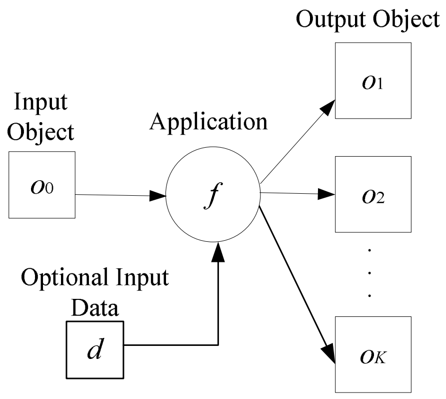


Figure 4.9 Model for a Single Application

Suppose at the proxy the request probability to the output object o_j is p_j and the total request rate to all *K* objects is λ. Thus, we have Eq.4.1 and Eq.4.2, where λ_j denotes the request rate to object o_j.

$$\sum_{j=1}^{K} p_j = 1 \qquad \text{(Eq.4.1)}$$

$$\lambda_j = \lambda \cdot p_j \ (1 \leq j \leq K) \qquad \text{(Eq.4.2)}$$

If we execute application *f* on the server (server-side placement), the original object o_0 will be transformed at the server and the output objects will be transmitted to the proxy. For each object, the transmission cost is multiplying the object's size with the request rate to this object or the object's expiry rate whichever is smaller. Thus, we have the summed transmission cost between the server and proxy as in Eq.4.3.

$$C = \sum_{j=1}^{K} \left\{ s_j \times \min(\lambda_j, \frac{1}{t_j}) \right\} \qquad \text{(Eq.4.3)}$$

On the other hand, if we deliver the original object o_0 and application *f* to the proxy (proxy-side placement), the transmission cost between the server and proxy is the summed transmission traffic for object o_0, application f, and the optional data d. The transmission cost C' can be calculated as in Eq.4.4, where λ_0 denotes the cumulative request rate to object o_0, and s_f and t_f denote the size and lifetime of application f.

$$C' = s_0 \min(\lambda_0, \frac{1}{t_0}) + s_f \min(\lambda_0, \frac{1}{t_f}) + s_d \min(\lambda_0, \frac{1}{t_d}) \qquad \text{(Eq.4.4)}$$

$$\text{where } \lambda_0 = \sum_{j=1}^{K} \lambda_j = \sum_{j=1}^{K} \lambda \cdot p_j = \lambda$$

To determine whether to apply the server-side placement or the proxy-side placement, we can compute the difference between C and C', as in Eq.4.5. We should choose the proxy-side placement if *CostSave*>0; otherwise, the server-side placement should be used.

$$\begin{aligned} CostSave &= C - C' \\ &= \sum_{j=1}^{K} s_j \min(\lambda_j, 1/t_j) - s_0 \min(\lambda_0, 1/t_0) - s_f \min(\lambda_0, 1/t_f) - s_d \min(\lambda_0, \frac{1}{t_d}) \end{aligned} \qquad \text{(Eq.4.5)}$$

$$\text{where } \lambda_0 = \sum_{j=1}^{K} \lambda_j = \sum_{j=1}^{K} \lambda \cdot p_j = \lambda$$

4.3.2 Optimization Model

A workflow involves multiple segments, each of which is mapped to one application. This section proposes the optimization model for the service workflow placement.

For each individual service, many versioned objects will be generated from the original object. For simplicity, our discussion uses term “object” to refer any version of these content objects. To express the relationship among them, we construct an “Objects-Tree”. The tree is rooted from the original object o_0. All objects directly generated from an object are listed as the children nodes of this object. The final content presentations are located as the leaf nodes. Using the breath-first traversal method [CLR90], any object in the tree can be indexed as o_j, where $0 \le j \le n$ and the total number of tree nodes is n+1. We denote the full objects-tree as T_0, and any sub-tree rooted at o_j as T_j. An example objects-tree is shown in Figure 4.10.

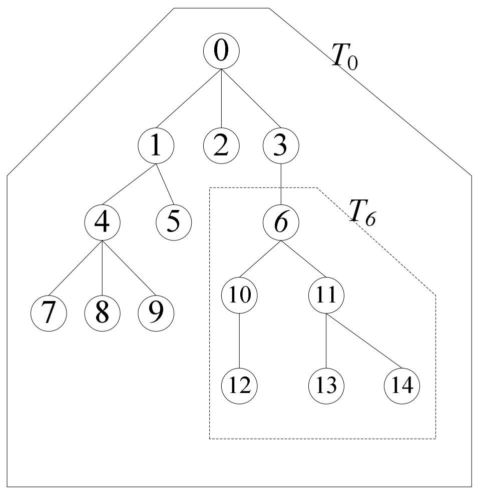


Figure 4.10 An Example Objects-Tree

We also define $ancestor(o_j)$ to be the set of objects that can generate object o_j, as given in Eq.4.6. Obviously, the original object doesn't have any ancestor, so that $ancestor(o_0) = \phi$.

$$ancestor\ (o_j) = \{o_a \mid o_j \in T(o_a)\} \qquad \text{(Eq.4.6)}$$

Now, the service placement problem can be rephrased as selecting some objects in the objects-tree and placing them with the associated applications to the proxy. Once object o_j is placed onto the proxy, all its descent objects in sub-tree T_j can be generated at the proxy. We name this phenomenon as the "Virtual Placement", and use vector $X[x_0 \ldots x_j \ldots x_n]$, which is called virtual placement vector, to represent the placement of the objects-tree's nodes on the proxy. x_j=1 if object o_j is actually or virtually placed on the proxy. Otherwise, x_j=0 if object o_j is neither actually nor virtually placed on the proxy.

Correspondingly, we also define the "Effective Placement", which can be represented as a vector $Y[y_0 \ldots y_j \ldots y_n]$, where any j with y_j=1 should satisfy the condition in Eq.4.7. It means that if an object is effectively placed on the proxy, this object should not have any ancestor object to be placed on the proxy. Only these effectively placed objects will be transmitted to the proxy.

$$\{ x_j = 1 \ \text{ AND } \ \forall o_a \in ancester(o_j) \rightarrow y_a = 0 \} \Rightarrow y_j = 1 \qquad \text{(Eq.4.7)}$$

The effective placement *Y* can be drawn from the virtual placement *X* by setting elements, which have ancestor objects placed on the proxy, to value zero. In general, we call object o_j is "effectively placed" on the proxy if y_j=1 and of course x_j=1; and object o_j is "virtually placed" on the proxy if x_j=1 but y_j=0. For example in Figure 4.10, if we place o_5, o_6, o_7, o_8 and o_9 to the proxy, the corresponding virtual placement *X* and the effective placement *Y* are shown in Table 4.1. Objects from o_5 to o_8 and o_9 are effectively placed and objects from o_{10} to o_{14} are virtually placed, while objects from o_0 to o_4 are not placed on the proxy.

Table 4.1 Virtual and Effective Placement Vectors

Obj	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
X	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
Y	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0

If denoting the request probability and the request rate to object o_j at the proxy as p_j and μ_j respectively, we have Eq.4.8 and Eq.4.9, where λ denotes the total request rate to the object-tree.

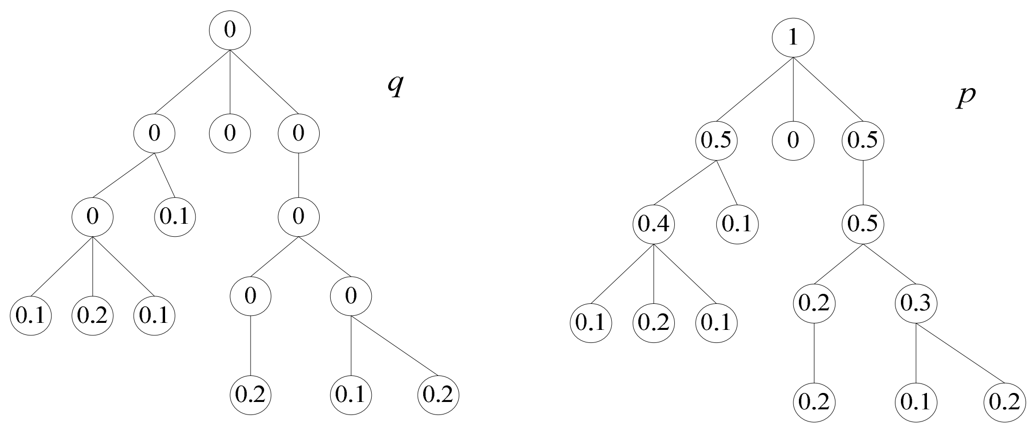
$$\sum_{j=0}^{|T_0|-1} p_j = 1, \text{ where } |T_0| \text{ is the number of nodes in tree } T_0 \qquad \text{(Eq.4.8)}$$

$$\mu_j = \lambda \cdot p_j \qquad \text{(Eq.4.9)}$$

To measure how frequently an object is directly requested by external nodes or indirectly requested to generate a descent object, we denote the request probability of sub-tree T_j as q_j, which is calculated by summing the request probabilities for all objects in T_j, as given by Eq.4.10. Correspondingly, the request rate, λ_j, to the sub-tree T_j can be calculated by Eq.4.11. For the example tree in Figure 4.10, if the request probability to each object is shown as in Figure 4.11(a), the request probability of each object's sub-tree is calculated as in Figure 4.11(b).

$$q_j = \sum_{i \in T_j} p_i \tag{Eq.4.10}$$

$$\lambda_j = q_j \cdot \lambda \tag{Eq.4.11}$$



(a) Request Probabilities of the Objects (b) Request Probabilities of the Sub-Trees

Figure 4.11 Request Probabilities of Objects and Sub-Trees

Based on the cost model for an application, we can express the optimization model for the objects-tree placement as in Eq.4.12 and Eq.4.13. Eq.4.12 indicates that the optimization problem is to resolve a placement *X* that provides the minimum cost *C*(*X*), the summation of the transmission cost $m_j(X)$ for all objects in the objects-tree T_0. Eq.4.13 calculates the server-to-proxy transmission cost $m_j(X)$ for object o_j, where the first term denotes the transmission cost for the object o_j, the second term denotes the transmission cost for the application associated with o_j, and the third term denotes the transmission cost for the supplementary data during the application execution.

$$\min(C(X)) = \min\left(\sum_{j=0}^{|T_0|-1} m_j(X)\right) \tag{Eq.4.12}$$

$$m_j(X) = s_j \cdot d_j(Y) \cdot \min(\lambda_j, \frac{1}{t_j}) + s_j^{(f)} \cdot d_j^{(f)}(X) \cdot \min(\lambda_j - \mu_j, \frac{1}{t_j^{(f)}})$$
$$+ s_j^{(d)} \cdot d_j(X) \cdot \min(\lambda_j - \mu_j, \frac{1}{t_j^{(d)}}) \tag{Eq.4.13}$$

To calculate the transmission cost for object o_j, we use λ_j as the total request rate to object o_j. For any object, we deem it is accessed if this object is directly requested by other nodes, or one of its descent objects is requested since in this case the descent object needs to be generated from the current object. Therefore, the sub-tree T_j's request rate λ_j can correctly reflect how frequently object o_j is requested directly and indirectly. In addition, d_j denotes the length of path [RRR98] for the proxy to fetch object o_j from the server. Firstly, if object o_j is not placed on the proxy (i.e. x_j=0 and definitely y_j=0), d_j should be 0 because the proxy will never retrieve this object under such placement. Secondly, if the object is effectively placed on the proxy (i.e. y_j=1 and definitely x_j=1), d_j should be 1, denoting the one-hop HTTP retrieval. Thirdly, if the object is virtually placed on the proxy (i.e. y_j=0 but x_j=1), the object can be generated from one of its ancestor objects that is effectively placed on the proxy, and the proxy will fetch that effectively placed object from the server. Thus, we should set d_j=0 to avoid recounting the transmission costs for the two objects. In summary, we should set d_j as a function of y_j, as given in Eq.4.14.

$$d_j(Y) = \begin{cases} 1, when: y_j = 1 \\ 0, when: y_j = 0 \end{cases} \qquad \text{(Eq.4.14)}$$

The second term of Eq.4.13 calculates the transmission cost for the application associated with object o_j, where $s_j^{(f)}$ denotes the application's size, $d_j^{(f)}$ denotes the length of path to fetch the application, $t_j^{(f)}$ is the application's lifetime, and $\lambda_j(X)$-$\mu_j(X)$ is the application's execution rate. The value of $d_j^{(f)}$ is set as in Eq.4.15. We set $d_j^{(f)}$ as a function of x_j, because when a proxy transforms object o_j it needs to fetch the application associated with the object, no matter this object is effectively or virtually placed on the proxy.

$$d_j^{(f)}(X) = \begin{cases} 0, if: x_j = 0 \\ 1, if: x_j = 1 \end{cases} \qquad \text{(Eq.4.15)}$$

The third term of Eq.4.13 calculates the transmission cost for the supplementary data fetched during application execution, where where $s_j^{(d)}$ denotes the size of the data, $d_j(X)$ denotes the length of path to fetch the data, $t_j^{(d)}$ is the lifetime of the data, and $\lambda_j(X)$-$\mu_j(X)$ is the application's execution rate.

From above discussion, we can simplify Eq.4.13 as Eq.4.16. Finding the minimum cost for equations Eq.4.12 and Eq.4.16 will render us the optimal placement of content service.

$$\begin{aligned} m_j(X) &= s_j \cdot y_j \cdot \min(\lambda_j, \frac{1}{t_j}) + s_j^{(f)} \cdot x_j \cdot \min(\lambda_j - \mu_j, \frac{1}{t_j^{(f)}}) \\ &+ s_j^{(d)} \cdot x_j \cdot \min(\lambda_j - \mu_j, \frac{1}{t_j^{(d)}}) \end{aligned} \qquad \text{(Eq.4.16)}$$

4.3.3 Optimization Algorithm

This section explains an algorithm to resolve the optimal service placement X. The algorithm is executed by the server, and we suppose the server knows the average size and lifetime of its content objects and the corresponding application modules. Moreover, we suppose the request rate of the content objects at the proxy is also known by the server. This information can be got by requiring the proxy to report to the server, or allowing the server to derive the frequencies according to its own access log. The algorithm is given in Figure 4.12, which is based on the dynamic programming technique [CLR90].

```
/*the main function to resolve the optimal placement of tree $T_0$*/
Main()
{
    Allocate two global vectors $X$ and $Y$, which are accessible by all sub-functions
    Set $Y$ as the effective placement vector for $T_0$, initializing all elements of $Y$ to zero
    Set $X$ as the virtual placement vector for $T_0$, initializing all elements of $X$ to zero
    $min\_cost$ = Optimal_Func($T_0$);
    Print out $Y$, which is the resolved effectively placement of $T_0$
    Print out $X$, which is the corresponding virtual placement of $T_0$
    Print out $min\_cost$, which is the expected minimum cost for $T_0$
}

/* The optimization function */
/* Input: the objects-tree $T_j$ to be resolved */
/* Output: the resolved minimum cost value */
float Optimal_Func($T_j$)
{
    $cost_1$ = Cost_Func($T_j$); /* compute the cost given $y_j$=1 */
    if ($o_j$ is a leaf object) {
```

```
(20)        Set $y_j = 1$ in vector $Y$ and correspondingly $x_j=1$ in vector X;
(21)        return $cost_1$;
(22)      }
(23)    Set $cost_2 = 0$;
(24)    for each child object of $o_j$ { /* compute the minimum cost given $y_j=0$ */
(25)        $cost_2$ += Optimal_Func($T_{child}$); /*compute the minimum cost for sub-tree $T_{child}$
(26)    }
(27)    if ($cost_1 >= cost_2$) {
(28)        Set $y_j=0$ in vector $Y$ and $x_j=0$ in vector $X$;
(29)        return $cost_2$;
(30)    } else {
(31)        Set $y_j=1$ in vector $Y$, and $x_j=1$ in vector $X$;
(32)        For all descent nodes in tree $T_j$, set $x=1$ in vector $X$ and $y=0$ in vector $Y$;
(33)          return $cost_1$;
(34)    }
(35)  }
(36)
(37)   /* compute the cost of the objects-tree $T_r$, when it is placed on the proxy*/
(38)   float Cost_Func ($T_r$) {
(39)     /*compute cost for the root object $o_r$*/
(40)    Set total_cost = $s_r \cdot \min(\lambda_r, \frac{1}{t_r}) + s_r^{(d)} \cdot \min(\lambda_r - \mu_r, \frac{1}{t_r^{(d)}}) + s_r^{(f)} \cdot \min(\lambda_r - \mu_r, \frac{1}{t_r^{(f)}})$
(41)    for all descent objects of $o_r$ {
(42)        total_cost += $s_j^{(d)} \cdot \min(\lambda_j - \mu_j, \frac{1}{t_j^{(d)}}) + s_j^{(f)} \cdot \min(\lambda_j - \mu_j, \frac{1}{t_j^{(f)}})$;
(43)    }
(44)    return total_cost;
(45)  }
```

Figure 4.12 Optimization Algorithm

The algorithm starts from the Main function (line 2), which first initializes the effective placement vector Y and the virtual placement vector X to the zero vectors (line 4 ~ line 6). Then it invokes the "Optimal_Func" function to resolve the optimal placement for the whole objects-tree T_0 (line 7).

"Optimal_Func" works in a recursive way to resolve the minimum cost for the inputted objects-tree T_j. Firstly, it calls the "Cost_Func" function (line 18) to compute the cost for placing T_j to the proxy. The computation of "Cost_Func" assumes the current sub-tree root o_r is effectively placed to the proxy, and correspondingly all its descent objects are virtually placed. According to Eq.4.16, the "Cost_Func" first calculates the expected transmission cost for o_r as in line 40, given y_r=1 and x_r=1. Next, it calculates the transmission cost for each descent object as in lines 41~42, given y_j=0 and x_j=1. The summed transmission cost for all objects is the transmission cost for the sub-tree T_r, which is returned to function "Optimal_Func" and recorded in $cost_1$ (line 18).

If o_j is a leaf object (line 19 ~ line 22), the "Optimal_Func" function will return $cost_1$ directly and set x_j=1 and y_j=1, because the final content presentation should always be delivered to the proxy.

However, if o_j has children objects, the algorithm needs to compute the minimum cost for all children's sub-trees, assuming not placing o_j to the proxy. To compute the minimum cost for any sub-tree T_{child}, the algorithm invokes the "Optimal_Func" function recursively (line 23 ~ line 26). The cumulative cost value for all children's sub-trees is stored in $cost_2$. Next, the algorithm compares the values of $cost_1$ and $cost_2$. If $cost_1 >= cost_2$ (line 27 ~ line 29), the algorithm determines not placing o_j to the proxy and thus sets x_j=0 and y_j=0. However, if $cost_1 < cost_2$ (line 30 ~ line 33), the algorithm determines placing o_j to the proxy will be beneficial, and it sets y_j=1 and x_j=1 (line 31). Meanwhile, the algorithm sets the effective and virtual placement for the non-root objects in sub-tree T_j as y=0 and x=1 (line 32). The "Optimal_Func" function returns the minimum cost value for sub-tree T_j. Finally, the Main function will resolve the optimal placement for the whole objects-tree T_0.

To estimate the time complexity of our algorithm, we assume the objects-tree is a perfect k-ary tree of depth H [CLR90]. At each depth h, the number of nodes is k^{h-1}, where $1 \le h \le H$. The total number of nodes is calculated by Eq.4.17.

$$n = \frac{k^H - 1}{k - 1} \qquad \text{(Eq.4.17)}$$

If denoting the time complexity for resolving the placement from the tree root (i.e. h=1) as $F(1)$, we can infer Eq.4.18, where $F(2)$ denotes the time complexity for resolving a sub-tree rooted at the depth of 2 (i.e. h=2). This is because to get the minimum cost for the whole tree our algorithm needs to compute the minimum costs for all *k* sub-trees rooted at depth 2 and compare their summed cost with "CostFunc(T_0)". Similarly, the time complexity for any sub-tree at depth *h* can be recursively resolved by Eq.4.19. Since the iteration of our algorithm stops at the leaf objects, we have Eq.4.20.

$$F(1) = k \cdot F(2) + 1 \quad \text{(Eq.4.18)}$$
$$F(h) = k \cdot F(h+1) + 1, where: 1 \leq h \leq H - 1 \quad \text{(Eq.4.19)}$$
$$F(H)=1 \quad \text{(Eq.4.20)}$$

Thus, we can infer the time complexity for resolving the whole objects-tree as in Eq.4.21. Comparing Eq.4.21 and Eq.4.17, we see the time complexity of algorithm is O(*n*), where *n* is the total number of objects.

$$F(1) = \frac{k^H - 1}{k - 1} \quad \text{(Eq.4.21)}$$

In addition, the execution of algorithm requires two arrays, each of size *n*, to store the resolved effective and virtual placement vectors. The algorithm also needs two other arrays, each of size *k*, to store the intermediate cost values for the sub-trees in two consecutive iteration levels. Thus, the spatial complexity of our algorithm is O(2*n*+2*k*).

After getting the *X* and *Y* vectors, the App.Net server will take the most beneficial operation manner for each proxy and each content object. Supposing the server receives a request for a particular content object, it will check the *X* and *Y* vectors. If the requested object is to be placed to the proxy effectively, the server will generate this object and send it with the associated workflow to the proxy. However, if the requested object is not to be placed to the proxy effectively, the server will find this object's ancestor that is to be effectively placed to the proxy. Then, the server will generate that object and send it with the associated workflow to the proxy, which will generate the final content presentation at the proxy.

We let the server execute the placement algorithm for each content service periodically to adjust the placement strategy to the changing environment. The server will use the resolved placement strategy for a period until the next round of placement adjustment.

4.3.4 Performance Modeling for Static Content

The performance modeling above assumes the content objects have limited lifetime. However, Web also has large amounts of static contents, such as the static text documents and the photographic images. This section setups the performance model for the static objects. We suppose both the static contents and their applications will not expire in the cache unless they are replaced by other objects.

The expected transmission cost for a content object is normally proportional to the size (*s*) and request probability (*p*) for this object. A rough estimation for the transmission cost is $s \times p$. Meanwhile, considering the existence of cache proxy, if an object can be used to generate multiple output objects and it has high request rate, this object should be much likely to be cached by the proxy. Combining above aspects, we propose a "weighted transmission cost" as in Eq.4.22, and use it as the performance metric for the static objects. The denominator of the formula is a caching trickle-off factor, where *W* is the number of output objects generated from the object under consideration and λ is the request rate for this object. It means if an object is intensively requested and it can be reused to generate many objects, this object is be much likely to be cached by the proxy, and this would potentially trickle off the transmission cost.

$$C = \frac{s \cdot p}{W^{\lambda}} \qquad \text{(Eq.4.22)}$$

Consequently, we can formulize the optimization model for the static contents in an objects-tree as Eq.4.23. The optimization is to resolve a placement *X* that provides the minimum weighted transmission cost. In the equation, Y denotes the effective placement vector, q_j denotes the request probability to sub-tree T_j, and d_j denotes the length of content fetching path. The definitions and calculations for all formula elements are the same as in section 4.3.1 and 4.3.2. In particular, for the leaf objects, W_j=1.

$$\min(C(X)) = \min(\sum_{j=0}^{|T_0|-1}[\frac{s_j q_j}{W_j^{\lambda_j}} \cdot d_j(Y)]) \qquad \text{(Eq.4.23)}$$

We can apply the same optimization algorithm as in section 4.3.3 to resolve the optimal placement for the static content objects. However, in this situation, the cost calculation for the objects-tree (line 37~45 in Figure 4.12) should be replaced by $\sum_{j=0}^{|T_0|-1}(\frac{s_j q_j}{W_j^{\lambda_j}} \cdot d_j(Y))$.

4.4 Summary

This chapter proposes the App.Net system that applies the Application Networking framework in the Web content delivery context. Compared to the traditional Web systems that fix content applications to either the original server or the edge proxy, our system provides more flexibility by enabling the server to deliver the intermediate content object as well as the partial service workflow to the proxy. This difference makes our system to be more scalable and efficient for pervasive content delivery. Dedicated caching scheme is also proposed, which extends the HTTP 1.1 protocol to cache different versions of responses. Furthermore, Performance model and optimal service placement algorithm are provided to minimize the transmission cost between server and proxy.

CHAPTER 5

PERFORMANCE EVALUATION IN THE APP.NET SYSTEM

5.1 Implementation

We build the App.Net prototype based on Jigsaw [Jigsaw], which is a standard edge server recommended by the WWW Consortium [W3C]. Building App.Net upon the Jigsaw platform provides a good reference implementation of the Application Networking framework for Web content delivery. In addition, Jigsaw is based on Java that supplies good support to mobile code, which is required in our App.Net system.

The prototype structure is illustrated in Figure 5.1, which has three tiers: the Jigsaw platform tier, the App.Net Proc. tier, and the ANlet applications tier.

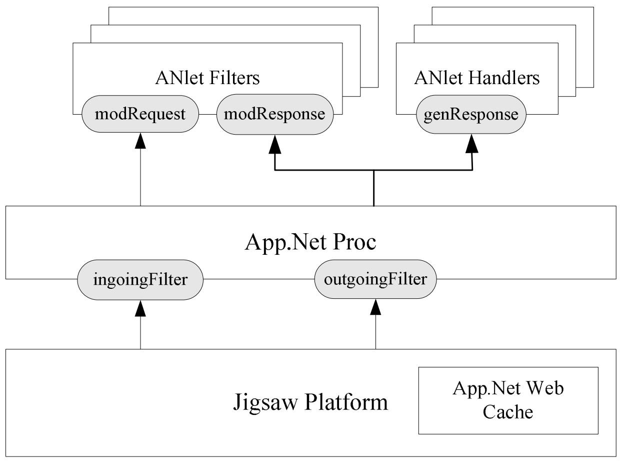


Figure 5.1 App.Net Prototype

The underlying Jigsaw, working either as the Web server or the Web proxy, implements the basic HTTP message transmission. We also show the Web Cache module as part of Jigsaw but it has been improved with the App.Net caching scheme. The App.Net Proc tier performs the main procedures, such as processing the workflow, fetching ANlet modules, executing applications. The ANlet modules are downloaded by the App.Net Proc to perform specific tasks in the service workflows.

The App.Net Proc. implements two APIs for Jigsaw, the ingoingFilter function and the outgoingFilter function. When Jigsaw receives a request or is about to send a response, it will call these functions to relay the request or response to the App.Net Proc. The App.Net Proc will manipulate the received message and return the new message to Jigsaw, which will forward the new message to the server or the client. More details are explained in the following two scenarios: (i) request modification and (ii) response transformation.

5.1.1 Request Modification

When the Jigsaw platform, usually as a proxy, receives a request, it will call the ingoingFilter function of the App.Net Proc. The definition of the ingoingFilter function is given by API.5.1, where the input parameter carries the original request message. Within the ingoingFilter function, the App.Net Proc first finds a service workflow responsible for current request; and then executes the relevant ANlet applications through their exposed modRequest function, as defined in API.5.2. The application will accept the original request message and return the modified request to the App.Net Proc and finally to Jigsaw as the return of the ingoingFilter function.

API.5.1 ingoingFilter

RequestMessage ingoingFilter(RequestMessage request)

Comment: Exposed by the App.Net Proc., invoked by the Jigsaw Platform

API.5.2 modRequest

RequestMessage modRequest(RequestMessage request)

Comment: Exposed by the ANlet Filter, invoked by the App.Net Proc.

As an example, we implemented a utility service on the App.Net proxy, which allows the clients to set their preference at the proxy. When the proxy receives the client’s request, it will invoke the ANlet application of the utility service to insert this client’s preferences into the request message as the extended HTTP headers. This utility service enables the client to clarify preference when his browser cannot do this.

5.1.2 Response Transformation

When the Jigsaw platform, either a server or a proxy, is going to send out a response message, it will call the outgoingFilter function exposed by the App.Net Proc. The definition of the outgoingFilter is given in API.5.3, which shows the Jigsaw platform relays the response together with the request to the App.Net Proc. The App.Net Proc will execute the service workflow associated with the response object and return the new response to the Jigsaw platform, which will then send it to the clients.

API.5.3 outgoingFilter

ReplyMessage outgoingFilter(RequestMessage request, ReplyMessage response)

Comment: Exposed by the App.Net Proc, invoked by the Jigsaw platform.

In specific, the App.Net Proc. uses either the modResponse function or the genResponse function to invoke the ANlet application, depending on the application is an ANlet Filter or an ANlet Handler. For an ANlet Filter, which is used to transform the response content, the App.Net Proc should invoke the modResponse function. The definition of the function is given in API.5.4, where the function accepts both the request and response messages and outputs the transformed response to the App.Net Proc.

API.5.4 modResponse

ReplyMessage modResponse(RequestMessage request,ReplyMessage response)

Comment: Exposed by the ANlet Filter, invoked by the App.Net Proc

This is a Jigsaw-based implementation of modify_Response in Api.4.2

For an ANlet Handler, which always locates at the beginning of a workflow and is used to generate the response content, the App.Net Proc will invoke the genResponse function. The definition of the function is given in API.5.5, where the request message is the only the input. The ANlet Handler application will generate the response content and return it to the App.Net Proc.

API.5.5: genResponse

ReplyMessage genResponse(RequestMessage request)

Comment: Exposed by the ANlet Handler, invoked by the App.Net Proc

This is a Jigsaw-based implementation of generate_Response in Api.4.1

In summary, the App.Net prototype allows us to build the App.Net server or proxy through plugging the App.Net Proc onto the Jigsaw server or proxy. The prototype implements most operations in the App.Net architecture, and is used in our simulation experiments.

5.2 Simulation Environment

We set up a simulation environment to measure the performance of the App.Net system, as shown in Figure 5.2. The main components have an App.Net server, an App.Net proxy, and a client-simulator. The client-simulator is implemented to emulate massive clients to request and retrieve contents, and it can insert corresponding request parameters into HTTP request according to a request profile. In simulation, the client-simulator generates the Poisson [CB96] distributed requests at varying rates.

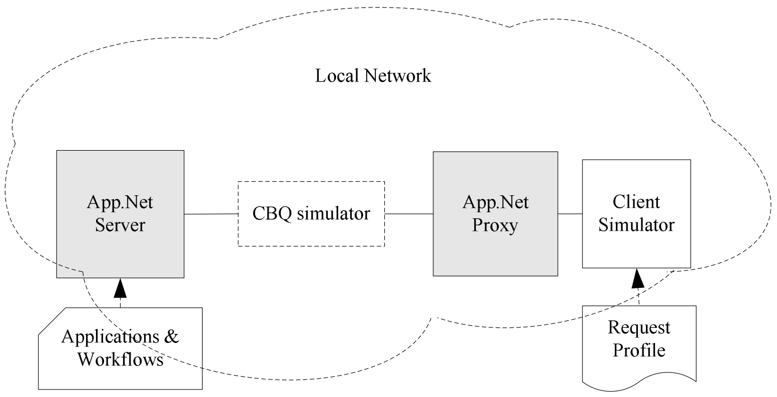


Figure 5.2 Simulation Environment

In our simulation, the server, the proxy, and the client-simulator are installed on three computers respectively, which are all located within a local 100Mbps Ethernet network. Because in the actual Web systems, the clients and their connected proxy are normally located near to each other while the proxy and the server are usually situated remotely, we use the CBQ simulator [Cbq01] to emulate the remote networking characters between the proxy and the server. In such network condition, the transmission cost between the clients and the proxy becomes negligible compared to that between the proxy and the server. Thus, reducing the transmission cost between the proxy and the server will improve the overall system performance.

Meanwhile, we implemented a wide range of ANlet applications, and the sizes of these applications mostly range from 30Kbit to 70Kbit. The applications normally have rather long lifetimes, so that the App.Net proxy can store and reuse them repeatedly. The overhead for transmitting these applications is much likely to be amortized in their long-term reuse. Hence, our performance evaluation treats the transmission cost for the content objects as the main factor, while omitting the transmission cost for the ANlet applications.

Our simulation adopts the following metrics to assess the system performance:

(i) Transmission cost: the volume of network traffic for response transmission, averaged by the simulation time.

(ii) System throughput: the amount of requests that are successfully finished within certain time interval.

(iii) Server throughput: the amount of requests that are successfully replied by the server within certain time interval. It reflects the load on the server.

(iv) User perceived latency: the time elapsed from when the client issues the request to when the response is fully received by him.

(v) Request drop ratio: the percentage of requests that cannot be served successfully.

5.3 Simulation for Applications

The first group of simulations measures the system performance for some representative Web applications. We emphasize on comparing the difference of placing the application on the server and the proxy, so that the potential advantages of the App.Net system for these applications can be explored. Because of this purpose, this part of simulation does not apply the service placement algorithm.

5.3.1 Application Taxonomy

In order to study the performance of existing Web applications in a systematic way, we divide them into several categories. For each category, a representative application is chosen and implemented. Our simulations are performed upon these applications to explore the potential performance benefits of the App.Net system for each category of applications.

According to our performance model, the execution of an application needs an input content object (Obj_{in}) and optionally needs supplementary input data (d), and the output is the transformed content object (Obj_{out}). As our performance model evaluates the content transmission cost, we classify the applications according to the sizes of the input and output for the application, as shown in Figure 5.3.

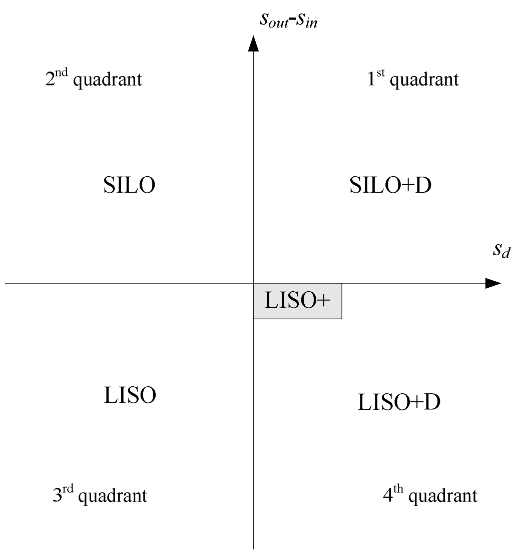


Figure 5.3 Application Taxonomy

The x-axis (s_d) represents the size of the supplementary data entry, and the negative section of the axis represents the application does not require supplementary data. The y-axis (s_{out}-s_{in}) represents the size difference between the output and input content objects. As a result, the two axes form four quadrants. We name the applications in the second quadrant as SILO (Small Input and Large Output) applications and those in the third quadrant as LISO (Large Input and Small Output) applications. In the first and fourth quadrants, the applications need supplementary data during execution. We name the applications in the first quadrant as SILO+D (SILO with supplementary Data) applications and those in the fourth quadrant as LISO+D (LISO with supplementary Data) applications. In particular, there is one group of LISO+D applications is rather popular among existing Web applications. The group is called LISO+, which needs supplementary data during execution, but the size of the supplementary data is much smaller than that of the input or output content objects. Table 5.1 summarizes the classification condition for each category of applications in the taxonomy, together with the example applications for each category. Our simulation will be based on these applications.

Table 5.1 Application Taxonomy Details

Type	Condition	Example
SILO	size(Obj_{in})<size(Obj_{out}),size(d)=0	Stock Chart Generator Document Rendering

LISO	size(Obj_{in})≥size(Obj_{out}),size(d)=0	Image Distillation;
		Doc Trimmer;
		Advertisement Rotator

SILO+D	size(Obj_{in})+size(d)≤size(Obj_{out})	Page Assembler
LISO+	size(Obj_{in})≈size(Obj_{out})>>size(d)>0	Image Watermarking; Video Encryption

5.3.2 SILO Application – Chart Generator

Many stock websites, such as NYSE (http://www.nyse.com), generate price charts at the server. Because the size of the price dataset is normally much smaller than that of the generated chart, if deploying the chart generation application to the proxy, the transmission traffic between the server and the proxy will be reduced.

We implemented a chart generation ANlet, which can generate the chart image according to the input price dataset. Our simulation measures the performance of the conventional server-side chart generation method and the App.Net method that puts the application to the proxy and generates the chart there. According to NYSE, we set the average size for the price dataset and the chart image to 4KB and 10KB respectively, and none object is cacheable. The server-to-proxy bandwidth is set to 1Mbps, and the simulation results are as follow.

Figure 5.4 presents the transmission cost for both methods. The App.Net method causes much less transmission cost than the server-side method, as the size of the transmitted price dataset is much smaller than that of the generated chart image. When the request rate exceeds 13/sec, the server-side method saturates the network bandwidth.

Figure 5.5 shows the system throughput for both methods. The throughput of the App.Net method enhances continuously with the increase of request rate. In contrast, the server-side method reaches the maximum throughput when the request rate is 13/sec, afterwards the system starts to drop requests due to the network saturation as shown in Figure 5.6.

We also compare the user perceived latency for both methods, as shown in Figure 5.7. The result indicates the App.Net method outperforms the server-side method significantly. In particular, the latency for the App.Net method keeps low until the request rate arrives 25/sec. Afterwards, the latency of this method increases considerably, because (i) the proxy reaches its maximum computation capability due to the ANlet execution and (ii) the bandwidth is going to be saturated by the transmission of price dataset.

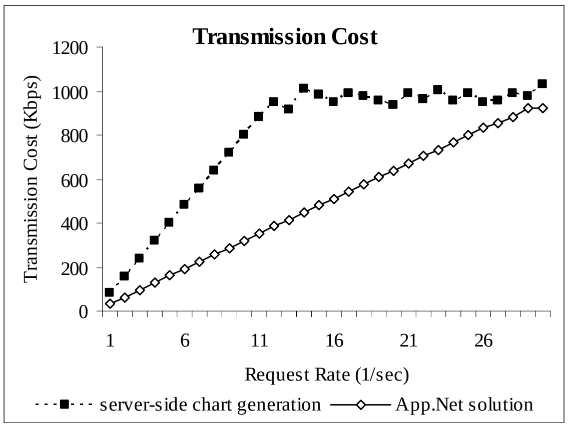


Figure 5.4 Transmission Cost

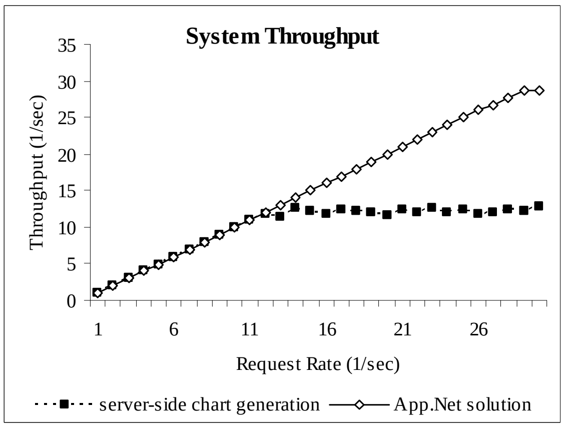


Figure 5.5 System Throughput

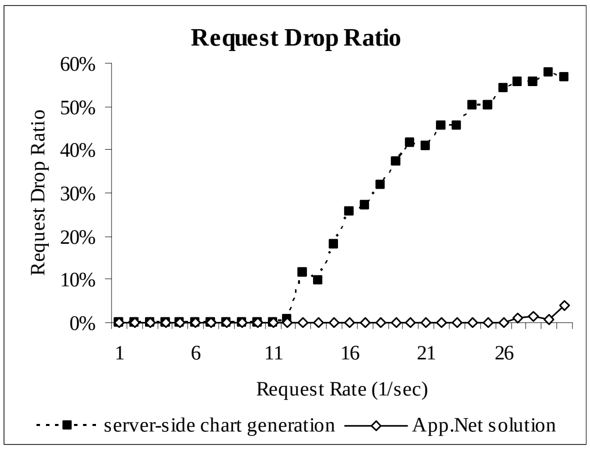


Figure 5.6 Request Drop Ratio

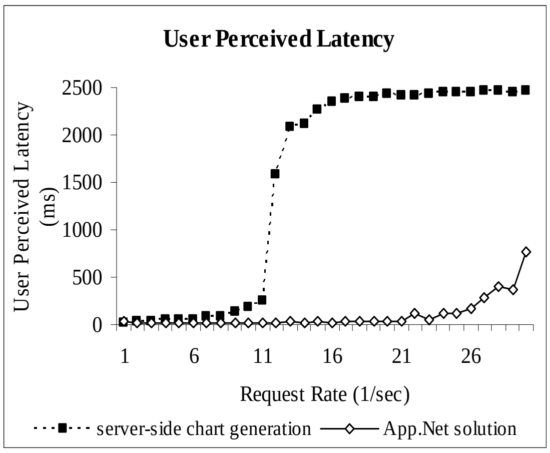


Figure 5.7 User Perceived Latency

The simulation results indicate the App.Net method is more efficient than the conventional server-side method, because transmitting the source dataset causes less network traffic than transmitting the generated chart image. Moreover, the results suggest that in the App.Net system deploying the SILO kind applications to the proxy can reduce the transmission cost, increase the maximum system throughput, and improve the user perceived latency.

5.3.3 SILO+D Application – Page Assembler

Delivering dynamic contents is a challenging topic for many Web systems. Traditional technology generates the dynamic Web pages on the server. However, these pages usually have poor cacheability, and they make the traditional method to be inefficient. Many researchers [DHR97 and SCK03] found that a significant portion of the dynamic page is usually static content, while the remaining parts are composed of contents that are more dynamic. According to this observation, we designed a page assembler ANlet, which is associated to a template page that contains the essential layout, the static information of the Web page and the links to the dynamic fragments. To form the final page, the ANlet will fetch the dynamic fragments from the server and compose them into the template page. Thus, if we deliver the template page and the assembler ANlet to the proxy, the proxy will cache and reuse them to generate the dynamic pages.

Compared to some existing assembly markup languages, such as the Edge Side Inclusion [Esi] and the HPP [DHR97], our assembler ANlet solution is more flexible from some perspectives. Firstly, the markup language solutions require the proxy to install a language processor. However, not all proxies installed those processors. Even for those installed proxies, they may use different processors, which causes the template page created using one language cannot be interpreted by the proxy with a different language processor. We implement the basic page assembly logic in a dynamic application, which enables the proxy to download it whenever necessary. Secondly, the markup language only implements page assembly functions, whereas our solution can implement advanced functions in the ANlet, such as the specific purpose computation, the client interaction logic, and the session management. These added functions make the ANlet to be more flexible in handling dynamic content.

A challenge for executing the assembler ANlet on the proxy is it needs to fetch page fragments from the server during each execution. How will this overhead affect the system performance? We performed a set of simulations to measure the performance gain and the corresponding overhead. In the simulation, the size and the TTL of the template page are 6KB and 5 seconds respectively. The template requires two fragments, each is of size 2KB and non-cacheable, to compose the final page. The final page is of size 10KB and non-cacheable. The server-to-proxy bandwidth is 512Kbps. Our simulation compares the server-side page assembling method and the App.Net method using the assembler ANlet. The simulation results are as follow.

Figure 5.8 presents the transmission cost for both methods. The server-side method causes significant transmission cost because the delivered final page has no cacheability at the proxy. In contrast, the App.Net method causes much less transmission cost due to the reuse of page template, and after the request rate reaches 14/sec the network starts to be saturated by the transmission of the page fragments.

Figure 5.9 shows the system throughput for both methods. The result indicates the App.Net method achieves higher maximum throughput than the server-side method. For the server-side method, when the request rate exceeds 13/sec, all requests are dropped due to serious traffic congestion.

We also measure the user perceived latency, as shown in Figure 5.10. The result indicates that the App.Net method can achieve lower latency than the server-side method. This is because the App.Net proxy only needs to transmit the document fragments for each request, while the server-side method needs to transmit the full page for each request and consequently prolongs the delay. In addition, our assembler ANlet is designed to fetch the page fragments simultaneously, so that the data fetching overhead is diminished.

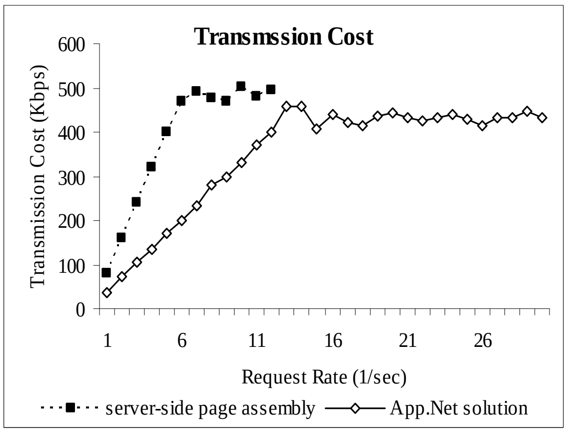


Figure 5.8 Transmission Cost

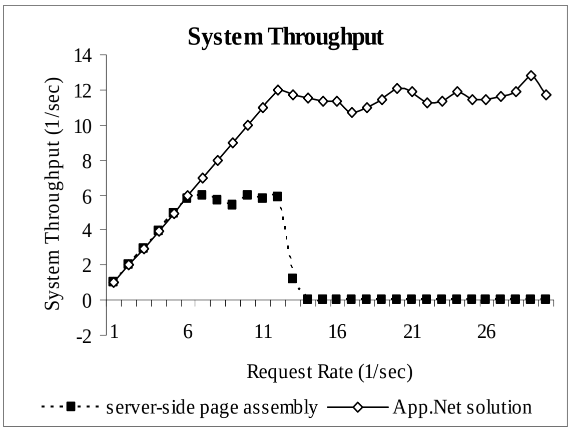


Figure 5.9 System Throughput

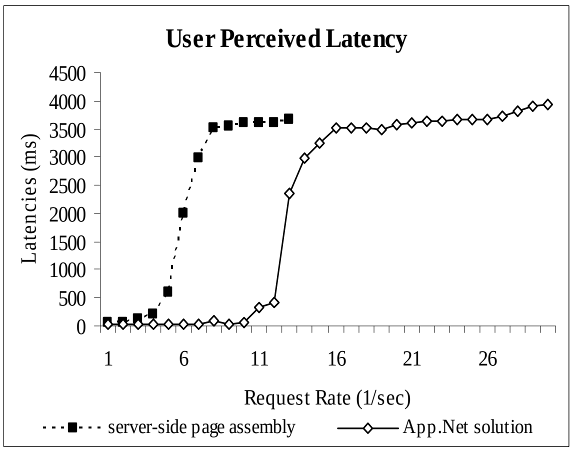


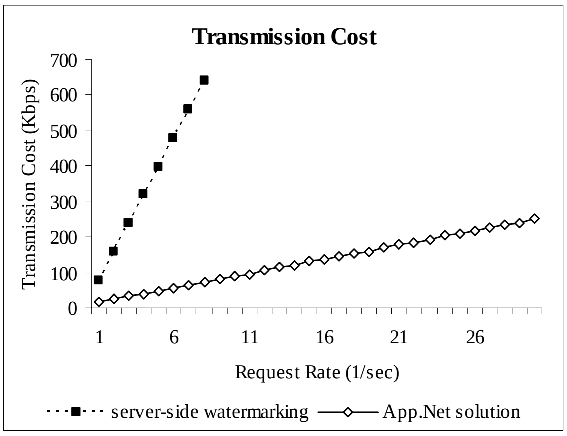
Figure 5.10 User Perceived Latency

In summary, the simulation results show that assembling dynamic pages on the proxy is more efficient than the traditional server-side assembling method. Moreover, the results suggest deploying the SILO+D kind applications to the proxy could be more efficient if we can reuse the input object at the proxy and reduce the data fetching overhead through parallelizing fetching process.

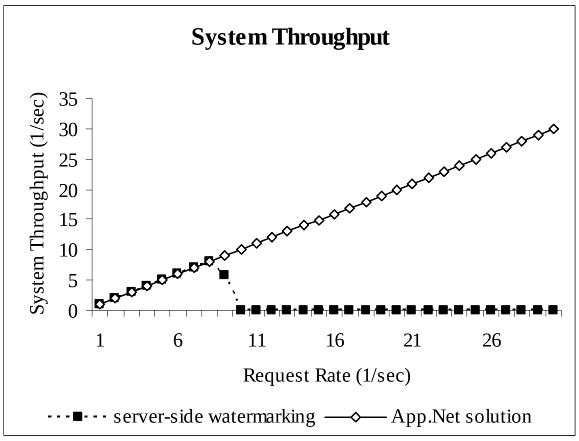
5.3.4 LISO+ Application – Watermarking

Similar to the SILO+D applications, the system performance can be improved if deploying the LISO+ applications to the proxy given the input object can be reused for multiple requests. We implemented a watermarking ANlet that can insert watermark into the image object. For each execution, the ANlet needs to fetch a certificate key from the server.

We measure the performance for the server-side watermarking method and the App.Net method that runs the ANlet on the proxy. In the simulation, the sizes of the image and the key are set to 10KB and 1KB respectively. The original image has TTL of 10 seconds, while the watermarked image is non-cacheable. In addition, the bandwidth between the server and the proxy is set to 1Mbps. The simulation results are summarized in Figure 5.11, which shows the App.Net method achieves less transmission cost, higher system throughput, and lower user latency. Since the results present the same trend as that of the SILO+D application, we do not repeat the explanation here.



(a) Transmission Cost



(b) System Throughput

Figure 5.11 Performance for the Watermarking Application

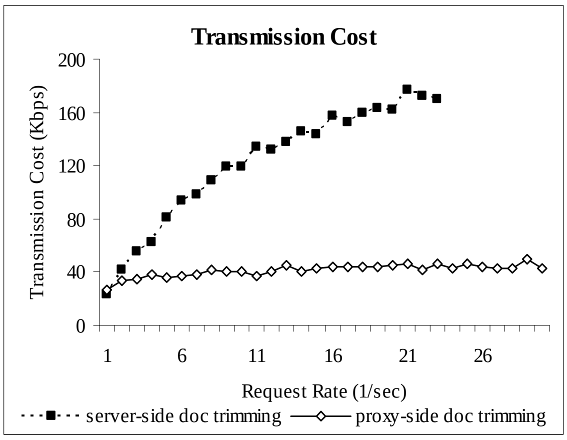
5.3.5 LISO – Document Trimmer

In this section, we use the Web document trimming application as an example to study the performance of the LISO applications. Nowadays, the services for adapting Web pages to display on handheld devices are rising up. A popular method is to trim the original page into small fragments and display them one by one according to the client's demand. Many solutions [HKO99, CZS01 and CMZ03] were proposed to perform such document trimming operation on the proxy, which needs to retrieve the original page from the server. We call them the proxy-side method in the following discussion.

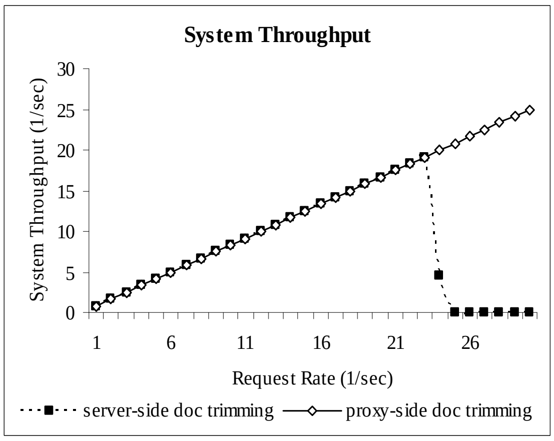
Different from the previous scenarios, in which the proxy-side method has significant advantages over the server-side method, the simulation results in this section will show that, for the LISO applications, the proxy-side method does not always get better performance than server-side method. The selection between the two methods should be dynamic according to the request pattern of clients.

To show the advantages and disadvantages of the server-side and proxy-side methods, we performed simulations to measure their respective performance under different request scenarios. We implemented a document trimmer ANlet, which is associated with a web page that will be partitioned into 20 fragments. The original page and the fragments are of size 10KB and 3KB respectively, and all of them have the TTL of 2 seconds. Our simulation also constructed the heterogeneous and the homogeneous request scenarios by varying the client request probabilities over the output fragments. For the heterogeneous scenario, the requests are evenly distributed over all fragments. However, for the homogeneous scenario, the requests are highly skewed, where we let 90% of the requests target to one fragment and the other 10% requests scatter over the rest fragments. We execute the ANlet on the server and the proxy respectively to simulate the server-side and proxy-side document trimming methods.

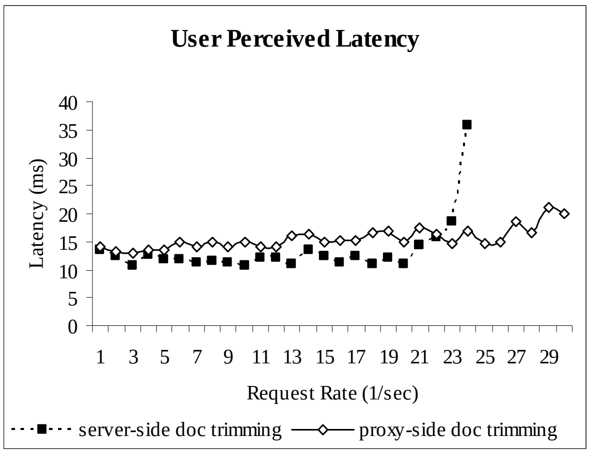
The simulation results for the heterogeneous scenario are shown in Figure 5.12. Figure 5.12(a) shows the proxy-side method causes much less transmission cost than the server-side method, because the former method can reuse the original page to serve the clients' requests for different fragments. Furthermore, Figure 5.12(b) shows when the request rate reaches 23/sec, the server-side method approaches the maximum system throughput, and afterwards the throughput drops sharply due to bandwidth saturation. However, the system throughput for the proxy-side method can enhance with the increase of request rate due to the low transmission cost of this method.



(a) Transmission Cost



(b) System Throughput

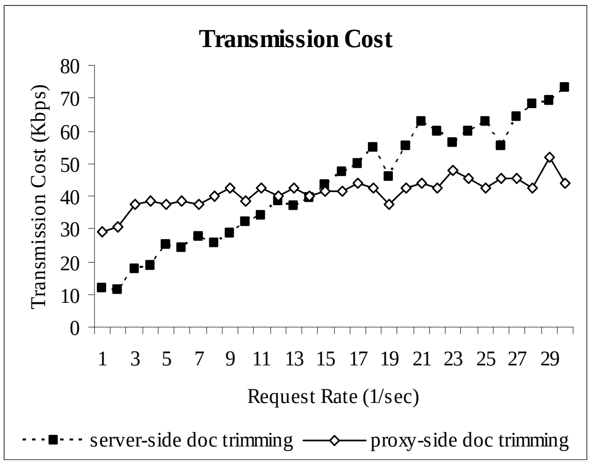


(c) User Perceived Latency

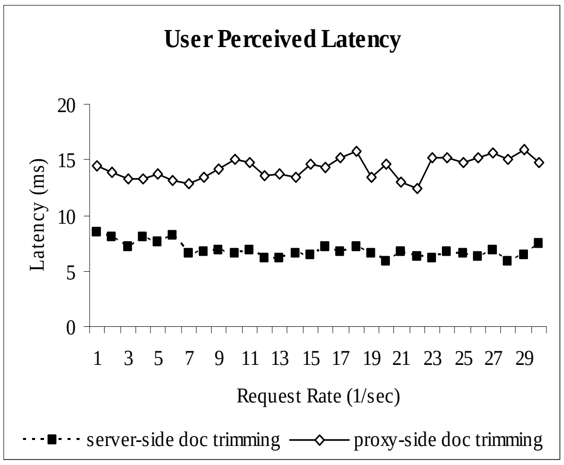
Figure 5.12 Performance of Document Trimmer in Heterogeneous Scenario

In addition, Figure 5.12(c) shows both methods have the similar user latency when the request rate is low. However when the request rate exceeds 23/sec, the latency for the server-side method increases greatly, due to the bandwidth saturation. In particular, we notice that the latency of the proxy-side method is around 5ms larger than that of the server-side method before the bandwidth is saturated. This is due to the overhead of ANlet execution on the proxy. For the server-side method, the proxy caches the final page fragments. Thus, when the proxy receives the client's request, it can reply the cached fragment directly. However, for the proxy-side method, the proxy caches the original page and it needs to execute the trimmer ANlet for each request. This makes the proxy-side method to cause slightly longer delay than the server-side method. For the document trimming application, this minor computation overhead seems to be negligible compared to the significant improvement on the transmission delay. However, the overhead may become significant for some computation demanding applications. A solution is to allow the cache to store multiple versions of content objects to reduce the average computation time at the expense of consuming more storage space. However, such caching management is not the focus of our App.Net study.

Figure 5.13 shows the simulation results for the homogeneous scenario, and it presents quite different trends from those in the heterogeneous environment. Figure 5.13(a) shows under the homogeneous environment the proxy-side method causes more transmission cost than the server-side method when the request rate is low. It means transmitting the original page results in considerable overhead, and under low request rate the original page cannot be fully reused for generating different page fragments. However, with the increase of the request rate, this situation is changed in that the transmission cost of the server-side method enhances quickly and finally exceeds that of the proxy-side method. The result indicates in the homogeneous request scenario, the proxy-side method is only beneficial when the request rate is high enough so that the transmission cost saving due to reusing the original page can counteract the retrieval overhead for the original page. Figure 5.13(b) shows the user perceived latency for both methods. The figure presents the similar trend as in the heterogeneous environment, in that the proxy-side method causes slightly longer latency due to the overhead of ANlet execution.



(a) Transmission Cost



(b) User Perceived Latency

Figure 5.13 Performance of Document Trimmer in Homogeneous Environment

In summary, from the simulation results we can see that for the LISO application it is error-prone to apply either the static server-side or the static proxy-side method. We need to adopt an adaptive strategy that can resolve the suitable application placement policy according to the changing request pattern.

5.3.6 Discussion

Using the representative applications, we have evaluated the performance for each category of applications in the taxonomy. The simulation results suggest many existing Web applications can be deployed to the proxy to achieve higher content delivery performance.

- For the SILO applications, deploying them to the proxy can reduce the content transmission cost and improve the user perceived latency.
- The SILO+D applications and LISO+ applications can be deployed to the proxy to improve the content delivery performance through reusing the input object to generate multiple output objects. However, the applications should be designed to reduce the data fetching overhead.
- The deployment of the LISO applications should consider more factors, including the request rate to the content, and the requests' distribution over the output objects. To achieve beneficial strategy in different conditions, we need to apply the service placement algorithm, which will be examined in the next section.

5.4 Simulations for Service Placement

5.4.1 Overview

The second group of simulations evaluates the App.Net system's effectiveness on handling content services. Our evaluation is based on the "device-independent authoring service" (DIA), which is to provide proper content presentations on diverse client devices. The DIA service usually consists of many functions [HM04], such as layout adaptation, language rendering, navigation design, user interaction customization, and so on. Although many server-side and proxy-side solutions have been proposed, current research has not given much attention to the content delivery performance of the DIA systems. To fill this gap, we designed a DIA service workflow, which contains two ANlet applications, as shown in Figure 5.14. The first is a document trimmer ANlet, which partitions the original document into particular deck fragments. Each deck fragment [Wap00] is a transmission unit and to be displayed on the client's device separately. The second is a language rendering ANlet that generates the final web page using one of three rendering languages accepted by the client's device. The other functions, such as the navigation and user interaction customization, have no direct impact on the content delivery performance, so that they are not included in our workflow.

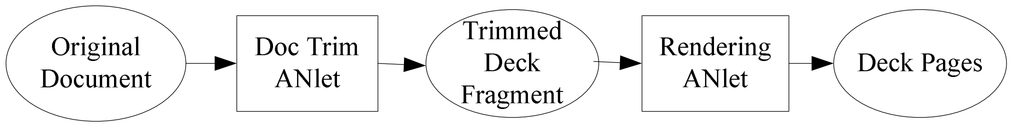


Figure 5.14 Device-Independent Authoring Service Workflow

To show how the App.Net system can deliver such DIA service efficiently, we conducted a series of simulations. These simulations are arranged in three sets: (i) we first evaluate the system performance for a single service with dynamic content; (ii) then we evaluate the performance for multiple services with the dynamic content; and finally (ii) we evaluate the performance for multiple services with static content.

5.4.2 Simulation on Single Service

The simulations in this section examine the optimal placement algorithm applied to a single DIA service. We constructed an example document that is of size 15KB. This original document can be trimmed by the Trimmer ANlet to 10 deck fragments, each of which is of size 4KB. The average size of fragments is larger than 1/10 of the original document's size, because the fragments contain some add-in parts such as (i) the header and footer, which provide general information about the document, and (ii) the fragments index, which facilitates the client to browse between different fragments. The Rendering ANlet can transform any deck fragment into the final deck page by selecting one of three rendering languages, and the average size of the final page is set to 6KB. The average size of the final deck page is larger than the corresponding fragment, because the Rendering ANlet will insert markup tags into the fragment and make it a presentable Web page. In addition, we set all objects have the

TTL of 2 seconds. In the simulation, the clients request all final deck pages randomly. Our simulation compares three methods: (i) the server-side method that provides the DIA service on the sever; (ii) the proxy-side method that provides the DIA service at the proxy; and (iii) the App.Net method that applies the service placement algorithm. The simulation results are as follow.

Figure 5.15 shows the transmission costs between the server and the proxy. The result shows the App.Net method achieves the minimum transmission cost. In particular, when the request rate is less than 1.8/sec the App.Net method causes less transmission cost than the other two methods. This is because the optimal placement algorithm resolves to deliver the intermediate deck fragments and the language rendering ANlet to the proxy, whereas the traditional server-side or proxy-side methods cannot perform such delivery strategy. When the request rate exceeds 2.1/sec the App.Net method achieves the same transmission cost as the proxy-side method. It is because in this situation the placement algorithm delivers the original document with the full workflow to the proxy, so that the App.Net method becomes equivalent to the proxy-side method.

Figure 5.16 shows the user perceived latencies. The App.Net method achieves the least latency because it can get the least transmission cost. In contrast, the server-side method causes the longest user latency. When the request rate exceeds 1.5/sec, the latency for this method increases greatly, due to the massive transmission of the final deck pages.

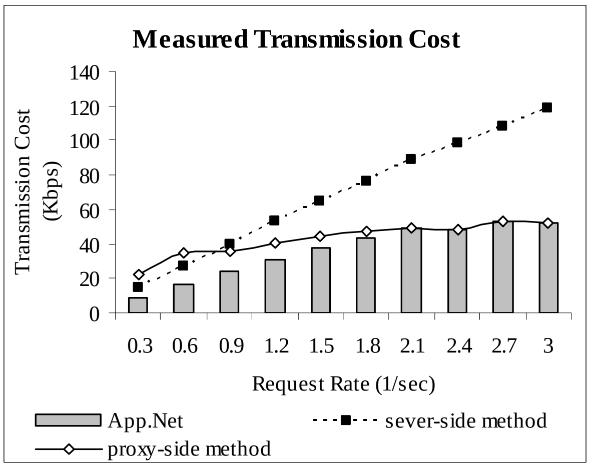


Figure 5.15 Transmission Cost

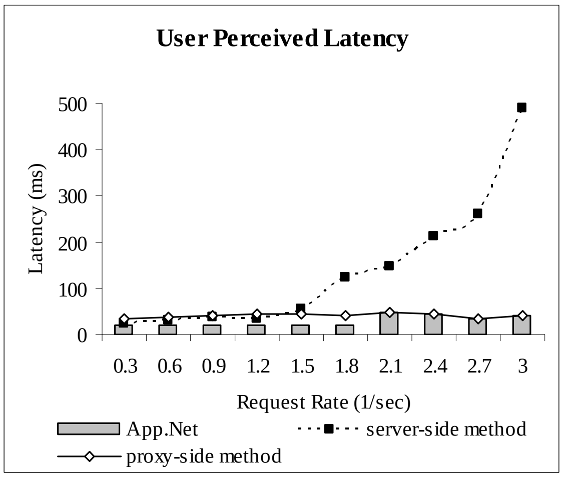


Figure 5.16 User Perceived Latency

Moreover, we extend the request rate to 30/sec to examine the system's processing capacity, where the server-to-proxy bandwidth is set to 160Kbps. The simulation results are as follow.

Figure 5.17 shows the system throughput, where the server-side method achieves the lowest system throughput. When the request rate reaches 5/sec, the server-side method reaches its maximum throughput, and afterwards more and more requests are dropped due to the saturation of network. In contrast, the App.Net method and the proxy-side method can achieve much higher system throughput, up to 26/sec. Afterwards, both methods start to drop requests, because the proxy reaches its maximum computation capability.

In addition, we also measure the server throughput, as shown in Figure 5.18. The server-side method causes the highest server throughput, because using this method all final deck pages have to be fetched from the server. After the request rate exceeds 7/sec, the server drops most requests due to network congestion. In contrast, the proxy-side method causes the lowest server throughput as the proxy only retrieves the original documents from the server. The App.Net method causes higher server throughput than the proxy-side method when the request rate is less than 3/sec. This is because in the request rate scope from 0 to 4/sec, the placement algorithm delivers the intermediate deck fragments to the proxy. However, when the request rate continues to increase, the algorithm changes to deliver the original document to the proxy. As the result, the App.Net method achieves the same server throughput as the proxy-side method.

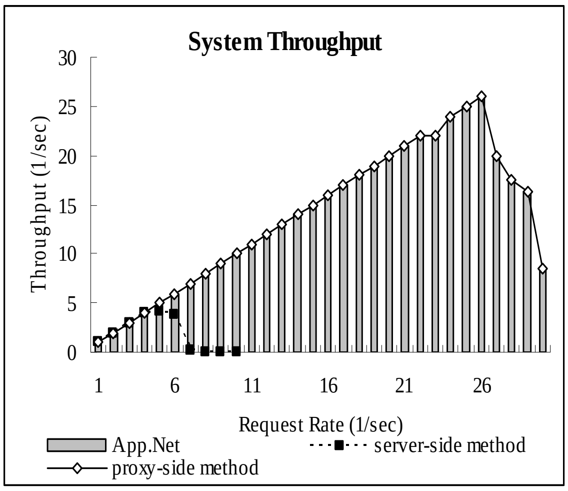


Figure 5.17 System Throughput

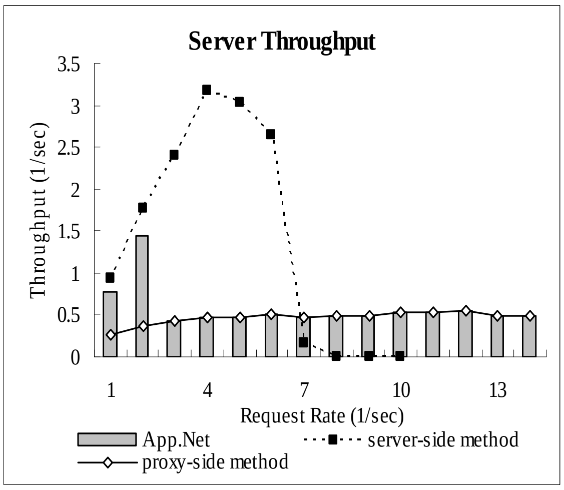


Figure 5.18 Server Throughput

In summary, the simulation results indicate that using the service placement algorithm the App.Net system can achieve better performance than the server-side and proxy-side methods, which place the content service at the server and the proxy statically. In the next two sections, we will examine the performance of the App.Net system when handling multiple services.

5.4.3 Simulation on Multiple Services for Dynamic Content

This section evaluates the App.Net system performance for handling multiple services, which start from dynamic content objects. We published 200 documents on the server. Every document is associated with the DIA workflow to form a distinct DIA service. For each service, the number of the generated content objects and the average size of these objects are the same as in the previous section. We assume all content objects for all services have the same lifetime of 60 seconds. In the simulation, the clients' requests to different services follow the Zipf distribution [BCF99], where

$\lambda_{zipf} = 0.7$. Meanwhile, we set the size of the proxy cache to 500KB, which is 1/6 to the size of the simulated working set. The cache applies the LRU replacement algorithm. Our simulations compare the performance of the App.Net method with the conventional server-side and proxy-side DIA methods. The simulation results are as follow.

Figure 5.19 shows the transmission cost, where the App.Net method achieves considerably less transmission cost than the other two methods. For instance, at the request rate of 10/sec, the transmission cost of the App.Net method is around 1/2 of the server-side method and 1/3 of the proxy-side method. This suggests our service placement algorithm can resolve efficient placement for multiple services. Although the algorithm resolves the placement for different services independently, for each service, it can obtain the optimal placement for all objects in the objects-tree. As the result, for all services, we can reduce the overall transmission cost to the reasonable extent.

Moreover, the figure shows that along the increase of request rate the transmission cost increment for the App.Net method is lower than that of the other two methods. This is because our method can automatically adjust the service placement according to the changing request rate while the other two methods apply the static service placement.

The figure also shows the proxy-side method causes the highest transmission cost. This is because the request probabilities for all 200 different services vary according to the Zipf distribution. Given a fixed total request rate, only those most frequently requested services are suitable to be fully placed to the proxy, whereas for the other services, fully placing them to the proxy is not beneficial. Therefore, the proxy-side method, which strictly places all services to the proxy, actually causes quite a lot of transmission overhead.

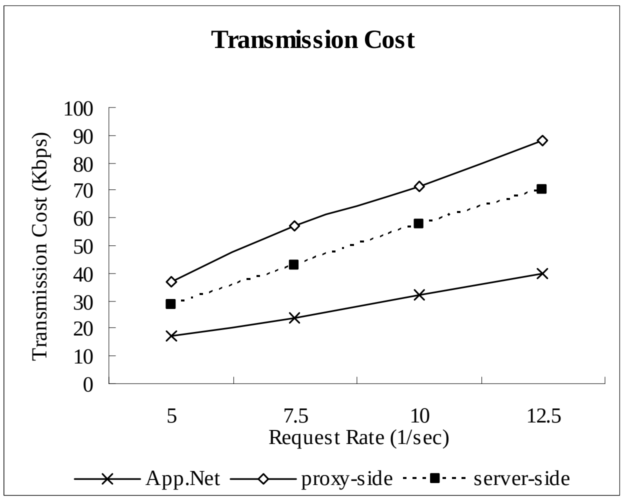


Figure 5.19 Transmission Cost

Figure 5.20 displays the cache-hit ratio at the proxy. The hit ratio for the server-side method is extremely low, because this method delivers the final deck pages, which have poor reusability at the proxy. On the other hand, the proxy-side method has the highest hit ratio, as it delivers the original documents to the proxy so that the cached items are likely to cover more requests. Nevertheless, such high hit ratio is at the expense of high content transmission cost as have been shown in Figure 5.19. In addition, the hit ratio for the App.Net method is lower than the proxy-side method. This is because our method allows many intermediate objects to be delivered to the proxy. Consequently, it decreases the coverage of the cached items. Furthermore, from the figure we see that the cache-hit ratio of the App.Net method enhances with the increase of request rate. It is because the placement algorithm adjusts the service placement under the increasing request rate by placing more numbers of original objects to the proxy. It causes the hit ratio enhances quickly. However, for the other two methods, the cache-hit ratio does not change due to their static placement policy.

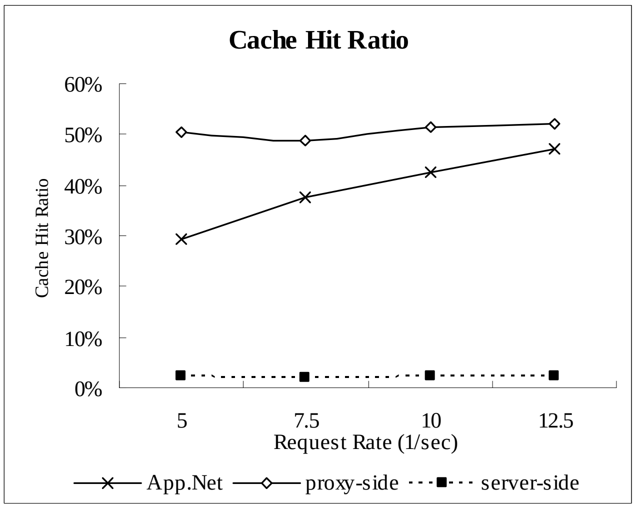


Figure 5.20 Cache Hit Ratio

Our simulation also measures the user-perceived latency, as shown in Figure 5.21. The server-side method causes the longest latency since this method generates at the server the final deck pages, which have poor reusability on the proxy. It causes massive content retrieval between the server and the proxy, and correspondingly enlarges the user latency. The figure also shows when the request rate is 5/sec the proxy-side method has slightly smaller latency than the App.Net method. This is because when the request rate is low, the placement algorithm in the App.Net method delivers the intermediate deck fragments to the proxy to ensure the lowest transmission cost. In contrast, for the proxy-side method, the proxy only needs to retrieve the original document for each service. As the result, in the App.Net method, the proxy needs to issue more requests to the server than in the proxy-side method and causes more TCP connection delay and request propagation delay, which finally prolongs the user perceived latency.

However, the simulation result also shows that with the increase of request rate, the user latency for the App.Net method reduces slightly, compared to the latency increase of the proxy-side and server-side methods. This is because our algorithm adjusts the service placement to keep the transmission cost as low as possible. With the increase of request rate, more original-documents are placed to the proxy, and the new placement makes the proxy to issue fewer requests to the server. Consequently, the cumulative TCP connection and request propagation overheads within the system are reduced, and this contributes to the reduction of the user latency.

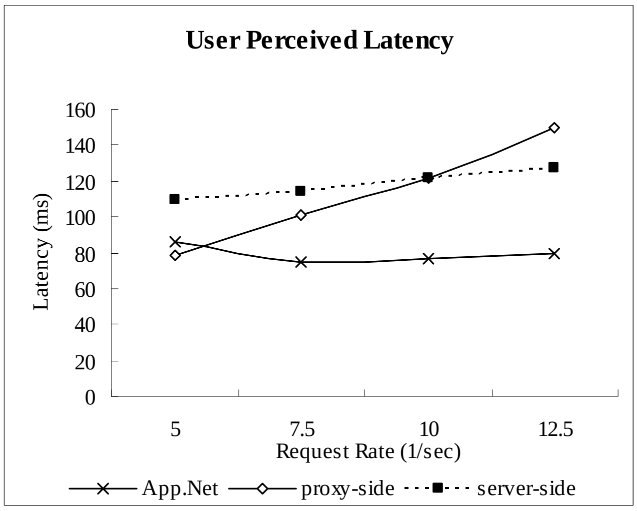


Figure 5.21 User Perceived Latency

Moreover, we also measure the system performance under different request patterns, including a heterogeneous requesting scenario and a homogeneous requesting scenario. In the heterogeneous scenario, the clients request for different deck fragments and different rending languages with uniform probabilities. In the homogeneous scenario, 85% of the requests to each service target to one main deck fragment and one major rendering language, while the remaining 15% of the requests are scattered over the rest deck fragments and rending languages. To achieve such varying request patterns, we prepare two request profiles for both scenarios. The client simulator first issues requests according to the heterogeneous profile for 30 minutes, and then requests according to the homogeneous profile for another 30 minutes. In both scenarios, the request distribution over different services follows the Zipf law ($\lambda_{zipf} = 0.7$), and the request rate is 10/sec. The simulation results are shown in Figure 5.22 and Figure 5.23.

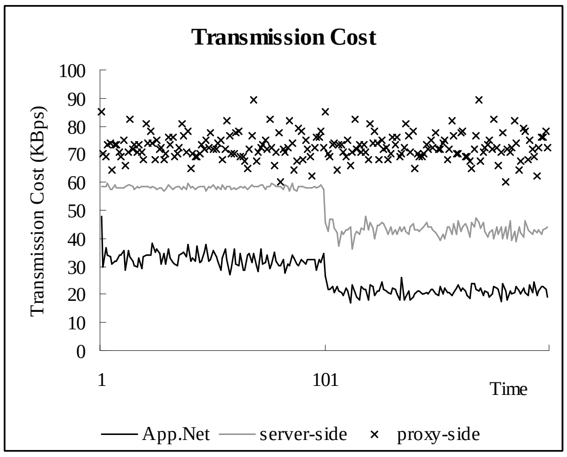


Figure 5.22 Transmission Cost under Different Request Patterns

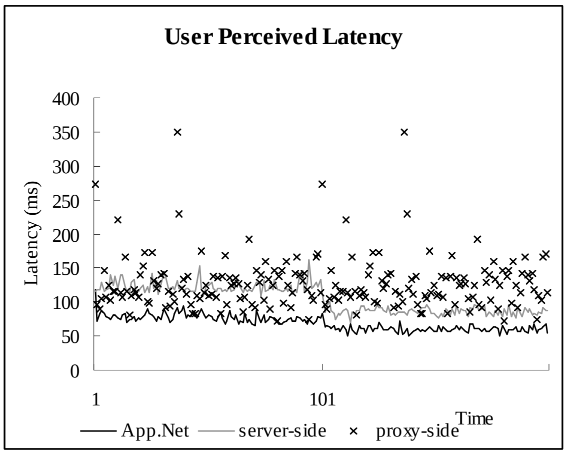


Figure 5.23 User Perceived Latency under Different Request Patterns

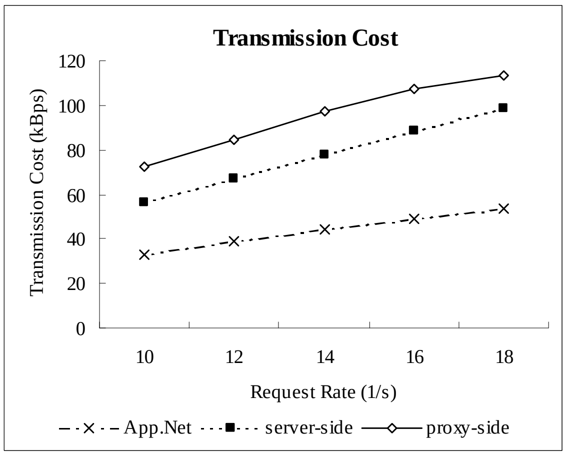
Figure 5.22 displays the transmission cost under different request scenarios, where the first half period is the heterogeneous scenario and the rear half is the homogeneous scenario. The App.Net method achieves the least transmission cost under both scenarios. When the request pattern changes, our method can adjust the placement accordingly. In contrast, the proxy-side method presents poor adaptation to the changing environment, as it always delivers the original content objects to the proxy without considering whether this is beneficial. Meanwhile, the transmission cost for the server-side method resides between the other two methods. When the request pattern changes, the transmission cost reduces correspondingly. This is because when the requests become homogeneous the proxy is more efficient to cache those popular objects and the transmission traffic is reduced naturally.

Figure 5.23 shows the user perceived latency under different scenarios. Similarly, we see the App.Net method achieves the least latency in both scenarios. The proxy-side method causes much longer latency and the latency jitters significantly during the simulation period. It suggests transmitting the original object for all services to the proxy is wasteful in this situation and it compromises the overall system performance. In addition, the latency for the server-side method falls between the former two methods, and the latency reduces when the request pattern changes from heterogeneous to homogeneous due to the reduction of transmission cost as explained above.

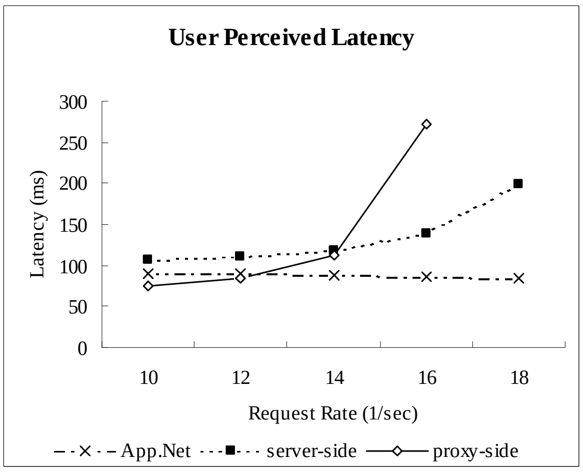
In summary, the simulation results indicate the App.Net system can perform efficient placement for multiple services, and it verifies the effectiveness of our performance model for dynamic content service and the optimal placement algorithm.

5.4.4 Simulation on Multiple Services for Static Content

This section evaluates the App.Net system performance for handling multiple services, which start from static content objects. The simulation is still based on the DIA service while the published 200 documents are all static content. The simulation results are summarized in Figure 5.24. We also measure the system performance under the heterogeneous and homogeneous requesting scenarios, as shown in Figure 5.25, where the first half of simulation period is the heterogeneous scenario and the rear half is the homogeneous scenario. The simulation results show the same trend as our previous simulations for the dynamic content services, and it verifies the effectiveness of our performance model for the static content services. The detailed explanation will not be repeated here.

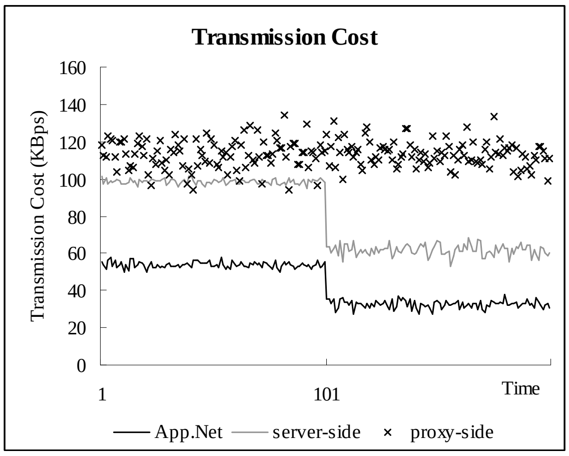


(a) Transmission Cost

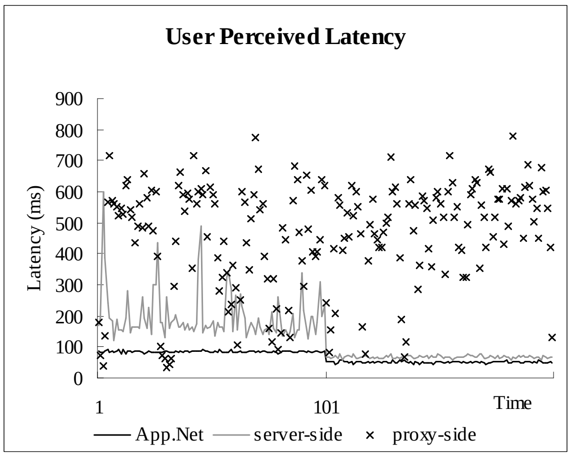


(b) User Perceived Latency

Figure 5.24 System Performance for Static Contents



(a) Transmission Cost



(b) User Perceived Latency

Figure 5.25 System Performance under Different Request Patterns

5.5 Summary

This chapter explains our system implementation and the performance simulations. Our simulations are organized as two steps. The first set of simulations is based on some representative Web applications in the application taxonomy. The results indicate our App.Net mechanism has potential advantages to a wide range of Web applications. The second set of simulations evaluates the effectiveness of the App.Net system when dealing with content services. The results show our system outperforms the traditional server-side and proxy-side methods, due to the usage of the optimal service placement algorithm. From the performance evaluation, we can conclude that the App.Net system can achieve higher performance than the existing active Web systems, because it can deliver intermediate content and partial service according to the performance requirement, whereas this advantage is not provided by the conventional frameworks.

CHAPTER 6

APPLICATION NETWORKING METHODOLOGY IN PEER-TO-PEER NETWORK

6.1 Motivation

P2P networks provide scalable data sharing and query mechanisms, which are helpful to implement the efficient content delivery systems. Typical systems have the file sharing systems [DKK01, RD01, Kaz, BitC], the P2P based Web content delivery systems [IRD02], and the multicast systems [CDK03]. However, recently, these P2P systems witness more user-customized queries, due to the emergence of diverse user devices and network connections, and the additional personalized demands. For instance, a peer node residing on a PDA may prefer to retrieve an image cropped to fit its small screen.

Such heterogeneous requirement motivates the P2P systems to render the customizable content services to provide the best-fit content. A straightforward method is shown in Figure 6.1. In this method, the content provider publishes the content service, which contains the original content object and its transformation applications, at the home node. When the home node receives a query, it will execute the service applications to transform the original content for the client. The system will deliver the customized response object to the client, and at the same time replicate this object in the P2P network.

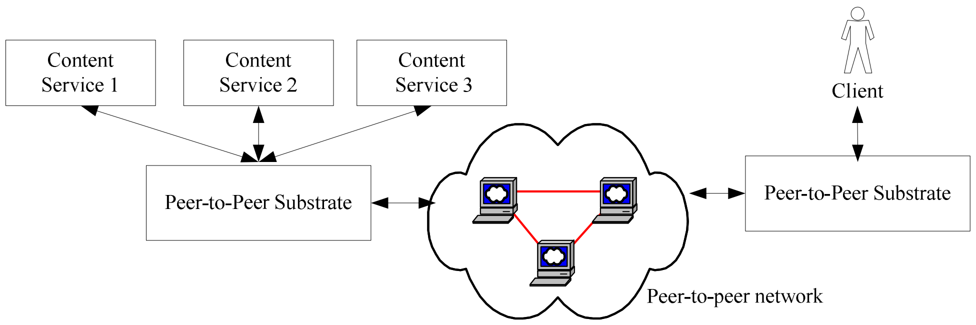


Figure 6.1 A Straightforward Content Service Methodology in the P2P Network

However, the main drawback of this method is that the customized response objects can only achieve limited peer sharing in the network, since it is not reusable for other user groups with different requirements. The low peer-sharing capability will compromise the system performance from several aspects:

(1) The low peer-sharing capability causes more nodes to be involved to find the correct version of content for a query. The enlarged query scope naturally causes longer query latency or even the failure of query.

(2) In the proximity-aware networks, where the query is propagated from the nearby to remote nodes, the enlarged query scope results in that the found destination node locates distantly from the requesting node and this normally implies larger content transmission cost and longer retrieval latency.

(3) The reduced peer-sharing capability causes more queries, which cannot find the sharing peer, to be served by the home node of the content service. This will burden the home nodes especially those publish the popular content services.

To address above drawbacks, we propose to apply the Application Networking framework in the P2P network. To render the customizable content service and at the same time improve the peer-sharing capability, we use the P2P network to propagate not only the response objects but also the content services. The content service includes the original content data, the service workflow, and the relevant applications. By reusing these components, the recipient nodes will be able to serve clients with heterogeneous requirements. We name this methodology as AN.P2P. This chapter emphasizes on explaining the methodology, while a detailed AN.P2P system will be studied in the next chapter.

6.2 Methodology

6.2.1 Overview

The general architecture of AN.P2P is shown in Figure 6.2, where the AN.P2P platform performs as a middle tier between the upper content services and the underlying P2P substrate. The P2P substrate is responsible for content delivery, and the content service is responsible for generating the correct response object. The AN.P2P platform is responsible service execution and replication, and it integrates the content delivery process with the content transformation process.

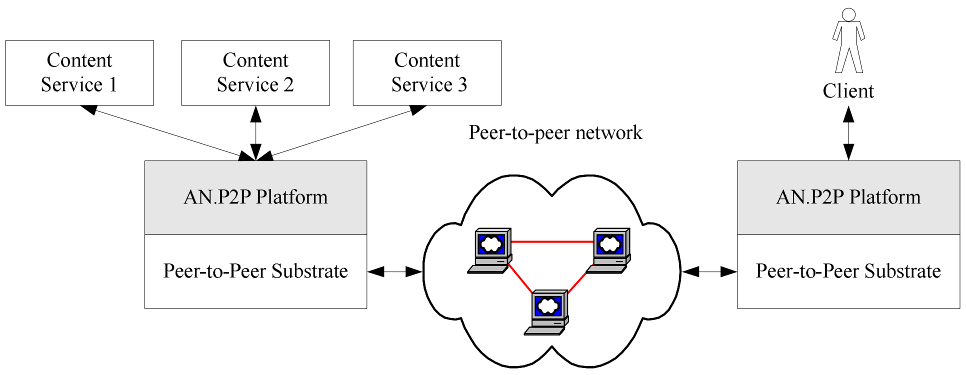


Figure 6.2 AN.P2P Methodology in the P2P Networks

The AN.P2P platform describes the content service as a workflow, as defined in Chapter 3. Feeding the original content object to the workflow applications, we can generate the customized response object for the client. The platform provides storage space for the published content service and renders the runtime for executing its applications. The platform also uses some spare space to cache the replicas of content services populated from other nodes. When the P2P substrate receives a query message from the network, it will relay the query to the AN.P2P platform, which will find out whether the present node publishes this content service or stores a replica of the service. In either situation, the platform will load the service components from the local storage and execute them to serve the query.

Our research uses Pastry, a representative DHT network, as the P2P substrate. This is due to the following reasons:

(1) The Pastry network is intensively studied by researchers. A lot of Pastry based systems have been or are being developed, such as PAST [RD01] for file sharing, Squirrel [IRD02] for distributed Web caching, and SplitStream [CDK03] for data multicast. The recent µTorrent [Utor], BitComet [BitC], and BitSpirit [BitS] systems also use Bamboo DHT [Bam], a revised Pastry protocol, to implement the tracker-less Bit-Torrent [BT] system. Thus, our AN.P2P study based on Pastry will have a widespread effect to these content systems.

(2) The Pastry routing is based on an identifier prefix algorithm and the routing table is constructed using the node's proximity awareness. These features are helpful to implement efficient content service replication in the network.

(3) The open source for Pastry is Free-Pastry [Pastry], which is developed using Java that provides well support for loading and executing the dynamic ANlet modules. Hence, our prototype system is implemented upon the Free-Pastry platform.

Our following discussion calls the node that publishes the original content service as the "home node", the node initiating the query as the "requesting node", all nodes forwarding the query message as the "intermediate nodes", and the node that finally serves the query as the "serving node". In the AN.P2P system, the serving node can be the home node of the queried content service or an intermediate node that stores a replica of the service. To distinguish these two possibilities, we call them as the "home serving node" and the "intermediate serving node" respectively.

6.2.2 Pastry Network

Pastry network assigns each node an m bit identifier (id), where m=128. The node ids are generated using a uniform hashing function, so that the resulting set of ids is uniformly distributed in the m-bit identifier space. Moreover, the Pastry routing protocol represents each b bits in id as one digit, so that the id can be seen as m/b digits, with base 2^b, in sequence. We call b as the "id-base". Meanwhile, the data's id is organized in the same way as the node id, and it shares the same identifier space.

To employ the Pastry routing protocol, each node maintains a routing table R, a leaf set L, and a neighborhood set M. The leaf set L contains $|L|$ nodes whose ids are numerically closest to the present node's id. The neighborhood set M contains $|M|$ nodes that are physically closest to the present node. Typical value of $|L|$ and $|M|$ is 2^b or 2^{b+1}. Supposing the whole network contains N nodes, the routing table would contain $\lceil \log_{2^b} N \rceil$ rows with 2^b-1 entries in each row. Each entry at row l refers to a node whose id shares the present node's id in the first l digits, but whose l+1th digit has one of the 2^b-1 possible values other than the l+1th digit in the present node's id. Pastry provides a mechanism, making use the neighborhood set of local node or propagated from other nodes, to enable each node to fill its routing table with the nodes that are physically close to the present node.

Each Pastry message contains a key (i.e. id of data), and the network will route the message to the node whose id is numerically closest to the given key. The routing process is accomplished in a hop-by-hop manner. In each hop, the node first checks whether the key is covered by the node's leaf set. If so, the message will be sent to the numerically closest node in the leaf set. Otherwise, the node will look up its routing table. The next-hop node's id should share with the key a prefix that is at least one digit longer than the prefix that the key shares with the present node's id. Such a node should exist on row l=shl(k,id)+1 of the routing table, where shl(k,id) denotes the length of the prefix shared between the key and the present node's id. Then, the algorithm chooses from the selected row the next-hop node, whose l+1th digit equals to the l+1th digit of the key. If no such a node is found, the message will be forwarded to a node whose id shares a prefix with the key as long as the present node's id, but is numerically closer to the key. Supposing the accurate routing information and no recent node failure, the message should reach the destination node within $\lceil \log_{2^b} N \rceil$ hops.

In the Pastry network, the expected distance traveled by a message during each successive routing hop is exponentially increasing. The reason is that any entry in the routing table's row l is chosen from a node set with size $N/2^{bl}$, which decreases exponentially by rows. Given the random and uniform distribution of node id in the network, the expected physical distance from the present node to the nodes referred by each successive row of the routing table will increase exponentially.

The Pastry network also provides replication mechanism that allows a node to replicate its data to k numerically closest nodes, where k is a configurable variable and its value usually satisfies $k \leq |L|$. These data replicas are mainly used for improving the data availability in case the home nodes leave the network. The replicas can distribute the queries over several nodes, so that they can achieve somewhat load balance in the network. However, such load balancing effect is quite limited, because the replica placement is static and the number of replicas cannot reflect the popularity of the data. Therefore, some Pastry systems (e.g. PAST [RD01]) replicates the delivered contents along the query path. These replicas will be reused to the subsequent queries with the same key. As a result, the more frequently a content object is queried, the more its replicas will be populated. Such system level replication can achieve much better load balance than the preliminary replication provided by the Pastry network. However, as explained, populating the customized content objects is not efficient to serve queries with heterogeneous requirements. This issue will be addressed by our AN.P2P methodology.

6.2.3 System Operations

The system operations for the AN.P2P methodology are illustrated in Figure 6.3. In the figure, we suppose node P_4 is the home node for a content service, and node P_0 is a requesting node that requests the content service on P_4. The figure also shows nodes P_0, P_1, P_2, P_3 and P_4 form a query path, and the length of this path is 4, in term of routing hops.

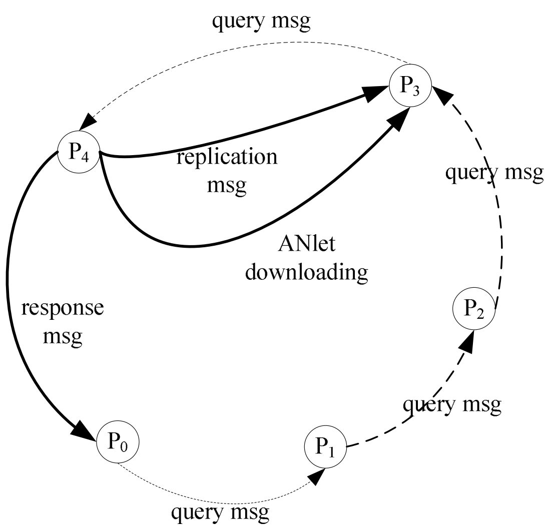


Figure 6.3 AN.P2P over the Pastry Network

When a content service is published by the content provider, it is assigned with a unique id. The service is organized as a workflow, including the original content object, the workflow specification, and the relevant ANlet modules. All these components are stored at the home node. We assume at the beginning only the home node stores the content service.

The Pastry network will route the first few queries to the home node. When the home node (e.g. P_4) receives the query, its Pastry substrate relays the query to the upper AN.P2P platform. The AN.P2P platform will fetch the queried content service from the local storage, and execute the service applications to transform the original content into the customized response object with respect to the query parameters. The response object will be sent to the requesting node (e.g. P_0).

Meanwhile, the home node can replicate the queried content service to other nodes. Some Pastry systems, like PAST [RD01], replicate contents to all intermediate nodes in the query path (e.g. P1, P2 and P3). However, in AN.P2P, we choose the last intermediate node in the query path (e.g. P3) to replicate the content service, because of the following reasons.

1) Since a content service involves one original content object and multiple applications, significant replication overhead will be caused if replicating them to all intermediate nodes in the query path.
2) According to Pastry routing protocol, the potential replica reuse rate will decrease about 2^b times per hop, from the rear (e.g. P_3) to the beginning (e.g. P_0) of the query path. Therefore, replicating service to the last intermediate node will be much more beneficial than the previous nodes in the query path.
3) In the Pastry network, the distance traveled by a message during each successive routing step is exponentially increasing. Therefore, replicating service to the last intermediate node can most significantly improve the content retrieval proximity because this node (e.g. P_3) is usually dispersed to the home node (e.g. P_4). However, the improvement for replicating to the front nodes (e.g. P_2 and P_1) becomes insignificant.

Therefore, we use the so-called “last-hop replication” method in AN.P2P. To accomplish the service replication, the home node sends a replication message, containing the original content object and the workflow specification, to the replication node (e.g. P_3). When that node receives the replication message, its Pastry substrate will relay the message to the upper AN.P2P platform, which will store the content object and its workflow specification into the local cache. Meanwhile, the node also downloads the required applications according to the workflow specification. After the replication node downloads those applications and stores them into the local cache, the content service is fully replicated on the node. Any node maintains the replicated services in an autonomous manner. It can evict low utility service replicas while retaining the high utility ones. We can leverage the existing cache replacement algorithms [CI97] to perform service replacement.

When the AN.P2P node (e.g. P_3) receives a query message, it first determines if it is responsible for the queried key. If it is, that means this node is the home node for the queried service and it will serve the query as discussed above. However, if the node is not responsible for the key, it will search the local cache to check if there has a replica of the queried service. If there is no replica, the node will forward the query through the Pastry substrate. However, if there is a replica hit, the node will perform as the “intermediate serving node” for this query. It will load the service applications, generate the response object, and send it back to the requesting node.

Meanwhile, this node will replicate the queried content service to the last intermediate node. It means that every time the service is queried, it will be replicated by the serving node. Through such process, the content service is propagated within the network. It makes the frequently queried services to have more replicas than those rarely queried services. Such distribution is helpful to improve the load balance and content delivery efficiency in the whole network, since the numbers of service replicas are proportional to the popularities of these services.

6.3 Performance Analysis

This section evaluates the performance of the AN.P2P methodology. We assume the network has N nodes and M distinct content services. Each node has the probability δ of being down, including the graceful leave and the unexpected failure. Supposing the Pastry hashing function can distribute these N nodes uniformly on the id ring, the approximate numerical distance between two adjacent nodes is $2^m / N$, where m is the length of id bit.

We denote content service as o_i ($1 \le i \le M$), and suppose the service can generate w response content objects, o_{ij} ($1 \le j \le w$). Let S_X and S_Y denote the average size of the content services and the response objects respectively. Let λ_{ij} denote the request probability to object o_{ij}, and λ_i denote the request probability to service o_i. As we consider the heterogeneous environment, where the clients' requests are distributed over different output objects, we have Eq.6.1, where the requests to all response objects form the entire requests set.

$$\sum_{i=1}^{M}\sum_{j=1}^{w}\lambda_{ij} = 1 \text{ and } \lambda_{ij} = \frac{1}{w}\lambda_i \qquad \text{(Eq.6.1)}$$

We choose the performance metric for the Pastry network as the "number of query hops" (QH), which is the number of nodes involved in the query-forwarding path. Our analysis starts from providing a utility algorithm to calculate the expected query hops for one piece of data. Based on the utility algorithm, we will compute the expected query hops for the service replication method and the object replication method respectively.

6.3.1.1 Utility Algorithm

In this section, we calculate the expected query hops for a piece of data, which has *r* replicas in the network. Given a requesting node and a randomly generated key, let the random variable *H* denote the number of query hops needed to reach the serving node. Let *T* be the random variable that is the clockwise distance, in term of the number of nodes, from the requesting node to the home node for the key, so that $0 \le T \le N-1$ and the probability $P(T=n)=\frac{1}{N}$.

Let $E_r(H)$ denote the expected hops for querying a piece of data with *r* replicas in the network. We can calculate $E_r(H)$ through equation Eq.6.2, which means that the calculation on the values of $E_r(H|T=n)$ suffices to compute the final $E_r(H)$. To simplify the notation, we use $h_r(n)$ to represent $E_r(H|T=n)$.

Note that $h_r(0)=0$, as shown in Eq.6.3, since a loop-back query goes to the local node directly. In addition, a query to the present node's leaf set, which is of size $|L|=2^b$, can reach the responsible node in one hop, so that we have Eq.6.4, where $1 \le n \le \frac{2^b}{2}$ refers to the $\frac{|L|}{2}$ nodes those are numerically closest to and with larger id than the present node, and $N-\frac{2^b}{2} \le n \le N-1$ refers to the $\frac{|L|}{2}$ nodes those are numerically closest to and with smaller id than the present node.

$$\begin{aligned} E_r(H) &= \sum_{n=0}^{N-1} P(T=n) E_r(H \mid T=n) \\ &= \frac{1}{N}\sum_{n=0}^{N-1} E_r(H \mid T=n) \\ &= \frac{1}{N}\sum_{n=0}^{N-1} h_r(n) \end{aligned} \qquad \text{(Eq.6.2)}$$

$$h_r(0)=0 \qquad \text{(Eq.6.3)}$$

$$h_r(n) = 1, \text{ where } 1 \le n \le \frac{2^b}{2} \text{ or } N - \frac{2^b}{2} \le n \le N-1 \qquad \text{(Eq.6.4)}$$

The Pastry routing process can be described as finding the correct row in the routing table and the correct column in that row. The process of selecting the correct row can be described as choosing a best stride of routing in the id ring. The possible strides can be divided into m/b levels. Each stride level, denoted by *l*, corresponds to one row of the routing table. The minimum stride is 2^b, which corresponds to the m/bth row of the routing table, and this is the first level of strides (i.e. $l = 1$). The maximum stride is $\frac{2^m}{2^b}$, which corresponds to the first row of the routing table, and this is the m/bth level of strides (i.e. $l = m/b$). The stride increases by 2^b times for each pair of successive rows from the m/bth row to the first row. Let l_n denote the best stride level to forward the message, where $0 \le n \le N-1$. The "best" means the chosen stride should go to a node that is numerically closest to the query key among all the m/b possibilities. Therefore, we can resolve l_n as in Eq.6.5, and the $\frac{m}{b} - l_n + 1$th row is the selected row.

$$l_n = \max\{l \mid (2^b)^l < \frac{2^m n}{N}, \quad and \quad 1 \le l \le \frac{m}{b}\} \qquad \text{(Eq.6.5)}$$

Then, the node needs to determine the next hop node from the 2^b columns in this row. As the entry whose id shares the first $\frac{m}{b} - l_n + 1$ digits with the key will be chosen, we can calculate the best column j_n as in Eq.6.6, where j_n is the space between the columns occupied by the present node and the chosen column.

$$j_n = \max\{j \mid j \cdot (2^b)^{l_n} < \frac{2^m n}{N}, and \quad 0 \le j \le 2^b\} \qquad \text{(Eq.6.6)}$$

If this best node is available, the query will advance $\left\lceil \frac{(2^b)^{l_n}}{2^m / N} \right\rceil \cdot j_n$ nodes on the id ring. However, if this best node happens to be down, a sub-optimal node will be chosen from the $j_n - 1$ columns between the best column and the column occupied by the present node. If the sub-optimal node is found at the Δj th adjacent column to the best column, the query will advance $\left\lceil \frac{(2^b)^{l_n}}{2^m / N} \right\rceil \cdot (j_n - \Delta j)$ nodes on the id ring, where $1 \le \Delta j \le j_n - 1$.

However, if the node cannot find the next hop node from above steps, it will choose from the leaf set. In average, the query will advance $\frac{2^b / 4}{2^m / N}$ nodes, a quarter of the leaf set's width.

Supposing there are *r* replicas of the queried data in the network, the query will be served if the next hop node has a replica. Considering all possibilities above, we can compute $h_r(n)$ as in Eq.6.7.

$$\begin{aligned} h_r(n) &= 1 + \sum_{\Delta j=0}^{j_n-1} q_n(\Delta j)(1 - p_r(n'_{\Delta j}))h(n'_{\Delta j}) + q_n(L)(1 - p_r(n'_L))h_r(n'_L) \\ &\text{where } 2^{b-1} < n \le N - 2^{b-1}, \\ &n'_{\Delta j} = n - \left\lceil \frac{(2^b)^{l_n}}{2^m / N} \right\rceil \cdot (j_n - \Delta j), \\ &\text{and } n'_L = n - \frac{2^b}{4} \end{aligned} \qquad \text{(Eq.6.7)}$$

$n'_{\Delta j}$ denotes the new value of variable *n* if the message is forwarded to the node at the Δj th adjacent column. Meanwhile, n'_L denotes the new value of variable *n* if the message is forwarded to a node from the leaf set. Eq.6.7 shows the computation of $h_r(n)$ can be reduced to the computation of all $h_r(n'_{\Delta j})$ and $h_r(n'_L)$ values.

In Eq.6.7, $q_n(\Delta j)$ denotes the probability that the Δj th column is chosen as the next hop. The value of $q_n(\Delta j)$ can be calculated by Eq.6.8, where δ denotes the probability a node is down. In addition, let $q_n(L)$ denote the possibility that the next hop node is chosen from the leaf set. The value of $q_n(L)$ should be δ^{j_n}, as in Eq.6.9, which means all nodes of the j_n columns in the chosen row are down.

$$q_n(\Delta j) = \delta^{\Delta j}(1-\delta) \qquad \text{(Eq.6.8)}$$

$$q_n(L) = \delta^{j_n} \qquad \text{(Eq.6.9)}$$

In addition, $p_r(n'_{\Delta j})$ in Eq.6.7, denotes the probability that the Δj th sub-optimal node has a data replica, and $p_r(n'_L)$ denotes the probability the chosen leaf set node has a data replica. We calculate their values as the following steps.

Pastry is a sparse network, where $N << 2^m$. Thus, the routing table, which can contain at most m/b rows, is usually not fully filled. Only the first t_N rows are used, where t_N is calculated by Eq.6.10.

$$t_N = \min(t \mid (2^b)^t > N) \qquad \text{(Eq.6.10)}$$

According to the Pastry algorithm, the replica distribution $p_r(n)$ should follow trend $p_r(2^{bl}) \approx 2^b \cdot p_r(2^{b(l+1)})$, where $1 \le l \le t_N - 1$. This is because a node, whose numerical distance to the query key is of $2^{b(l-1)} < n \le 2^{bl}$, is likely 2^b times being queried for this key than a node in scope $2^{bl} < n \le 2^{b(l+1)}$. Considering $p_r(n) \le 1$, we have Eq.6.11, where p is a probability factor to be resolved.

$$p_r(n) = \begin{cases} p, when(2^{(t_N-1)b} < n\frac{2^m}{N} \le 2^{t_N b}) \\ \min(2^b p,1), when(2^{(t_N-2)b} < n\frac{2^m}{N} \le 2^{(t_N-1)b}) \\ \cdots \\ \min(2^{(t_N-2)b} p,1), when(2^b < n\frac{2^m}{N} \le 2^{2b}) \\ \min(2^{(t_N-1)b} p,1), when(0 < n\frac{2^m}{N} \le 2^b) \end{cases} \qquad \text{(Eq.6.11)}$$

To resolve Eq.6.11, we omit constraint $p_r(n) \le 1$ first, and resolve the initial value of *p* by Eq.6.12. Then, the initial values of $p_r(n)$ for all scopes can be calculated. For those scopes, whose $p_r(n)>1$, we fix their $p_r(n)=1$, which means each node in these scopes has one replica of data. Then, we reevaluate Eq.6.12 to get the new *p* value for the remaining unresolved scopes. Such iteration is convergent as each time there are less unresolved scopes left, and it stops when the $p_r(n)$ for all scopes are not greater than one. These final $p_r(n)$ values provide the $p(n'_{\Delta j})$ and $p(n'_L)$ in Eq.6.7.

$$r = \sum_{i=1}^{t_n} p \cdot 2^{(t_N-i)b} \cdot (2^{ib} - 2^{(i-1)b}) \qquad \text{(Eq.6.12)}$$

Combining the results above, we can calculate all the $h(n)$ values in Eq.6.7 and finally the value of $E_r(H)$ in Eq.6.2.

- Example Computation

Figure 6.4 shows the computed $E_r(H)$ values in a Pastry network with 10,000 nodes, where b=3. The curves plot the $E_r(H)$ values under five node-down rates. $E_r(H)$ reduces with more data replicas are populated in the network, and enhances with the increase of node-down rate.

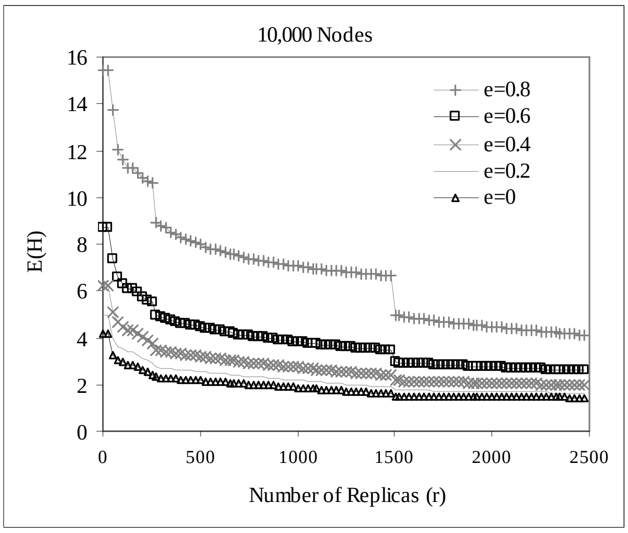


Figure 6.4 Predicted Query Hops in the Pastry Network

6.3.1.2 AN.P2P Performance Prediction

Based on the utility algorithm, this section computes the expected query hops, $E(QH)$, for the service replication method and the object replication method. We suppose M content services are in the network and each can generate w response objects. Each service or response object would have a specific number of replicas (r). For concise presentation, $E_r(H)$ is written as $F(r)$ in the following discussion, which means the number of query hops for the considered service or object is a function of its replica number. Our discussion considers three scenarios: (i) the conventional Pastry network, (ii) the AN.P2P network, and (iii) the hybrid network with both the AN.P2P enabled nodes and the conventional nodes.

In the conventional Pastry network, the content services are executed at their home nodes and the response objects are populated in the network. Let r_{ij} denote the number of replicas for object o_{ij}, and we have $r_{ij} = \lambda_{ij} \frac{S_T}{S_Y}$, where λ_{ij} is the query probability for o_{ij}, S_T is the total network storage size, and S_Y is the average size of response objects. The $E(QH)$ for the whole network is calculated by Eq.6.15.

$$\begin{aligned} E_1(QH) &= \sum_{i=1}^{M}\sum_{j=1}^{w}[\lambda_{ij}F(r_{ij})] \\ &= \sum_{i=1}^{M}\sum_{j=1}^{w}[\lambda_{ij}F(\lambda_{ij}\frac{S_T}{S_Y})] \end{aligned} \qquad \text{(Eq.6.15)}$$

In the AN.P2P network, the content services are replicated. Let r_i denote the number of replicas for service o_i, and we have $r_i = \lambda_i \frac{S_T}{S_X}$, where λ_i is the query probability to service o_i, S_X is the average size of service. The $E(QH)$ for the whole network is calculated by Eq.6.16.

$$\begin{aligned} E_2(QH) &= \sum_{i=1}^{M}\lambda_i F(r_i) \\ &= \sum_{i=1}^{M}[(\sum_{j=1}^{w}\lambda_{ij})\cdot F(\sum_{l=1}^{w}\lambda_{il}\cdot\frac{S_T}{S_X})] \\ &= \sum_{i=1}^{M}\sum_{j=1}^{w}\left[\lambda_{ij}F(w\lambda_{ij}\frac{S_T}{S_X})\right] \end{aligned} \qquad \text{(Eq.6.16)}$$

Considering $F(r)$ is a decreasing function of variable r, if condition $wS_Y > S_X$ is satisfied, we can infer that $E_1(QH) > E_2(QH)$. This means in the heterogeneous environment, where the clients' queries are scattered over different content objects, if the cumulative size of the response objects is larger than the size of the content service, replicating the content services will be more beneficial than replicating the response objects, in term of reducing the number of query hops in the network. In actual situation, condition $wS_Y > S_X$ can be satisfied by many content services that reuse an original content to generate many response objects. In most cases, the total size of the output objects is much larger than the summed size of the original object with the service applications. Thus, we can see that for a wide range of content services our AN.P2P method is more efficient than the conventional Pastry systems to serve the heterogeneous queries.

Finally, we consider a hybrid network with both the conventional nodes and the AN.P2P nodes. In this scenario, the response objects are propagated on the conventional nodes and the content services are populated on the AN.P2P nodes. Let S_H denote the total storage size of all AN.P2P nodes, and correspondingly the total storage size of all conventional nodes is $S_L=S_T-S_H$. Since a query can be served if it reaches a replica of service or a replica of the wanted object, we can compute $E(QH)$ for the whole network as in Eq.6.17. Supposing $\lambda_{ij} \approx \frac{\lambda_i}{w}$ and $wS_Y > S_X$, we can infer that the value of $E_3(QH)$ will reduce from $E_1(QH)$ to $E_2(QH)$, when S_H enhances from 0 to S_T. This conclusion means with more nodes are AN.P2P enabled, the average number of query hops in the network will reduce correspondingly.

$$\begin{aligned} E_3(QH) &= \sum_{i=1}^{M}\sum_{j=1}^{w}\lambda_{ij}F(r_i + r_{ij}) \\ &= \sum_{i=1}^{M}\sum_{j=1}^{w}[\lambda_{ij}F(\sum_{l=1}^{w}\lambda_{il}\frac{S_H}{S_X} + \lambda_{ij}\frac{S_T - S_H}{S_Y})] \end{aligned} \qquad \text{(Eq.6.17)}$$

6.4 Simulation

6.4.1 Overview

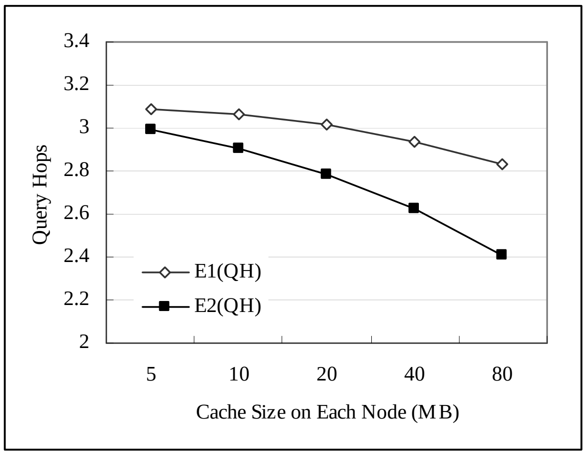
We performed a set of simulation experiments to explore the advantages of the AN.P2P methodology. Our simulation is based an AN.P2P prototype implemented on the Free-Pastry platform [Pastry]. The simulation uses the pervasive image provision as the sample service. The service workflow is originated from a Jpeg image and contains two workflow ANlet applications. The first ANlet can resize the original image to 5 different dimensions, including the original dimension, to fit the display size of the clients' devices. The second ANlet can transcode the image from Jpeg format to Gif and BMP formats. One of these three formats will be chosen according to the client's preference.

As the P2P overlay is generated by the network topology emulator of Free-Pastry, to lessen the impact on the simulation results due to the topology change, we performed each set of simulations repeatedly and collected the average results. The simulation network has 5000 nodes and 5000 Jpeg images. Each image is associated with the service workflow to form a piece of image service. The sizes of the original images follow the Pareto distribution [CB96], where $\alpha = 1.25$ and the minimum and maximum sizes are 500KB and 10MB respectively. The requesting nodes are randomly chosen in the network, and they randomly query an image presentation with specific dimension and format. The request probabilities over different image services follow the Zipf distribution [BCF99], where $\lambda = 0.7$.

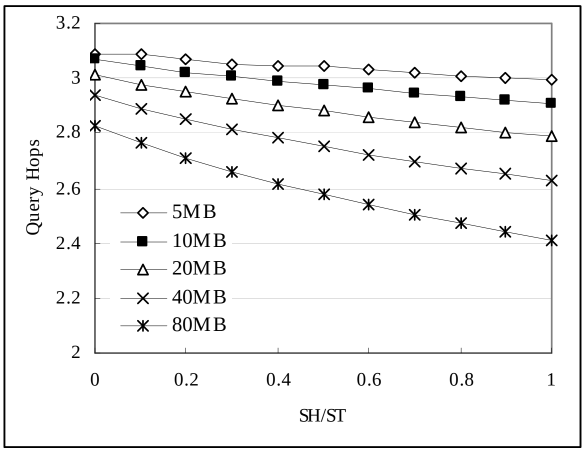
Our simulations compare the AN.P2P method with the plain P2P method, which executes the content service at the home node and populates the response objects in the network.

6.4.2 Computation Result

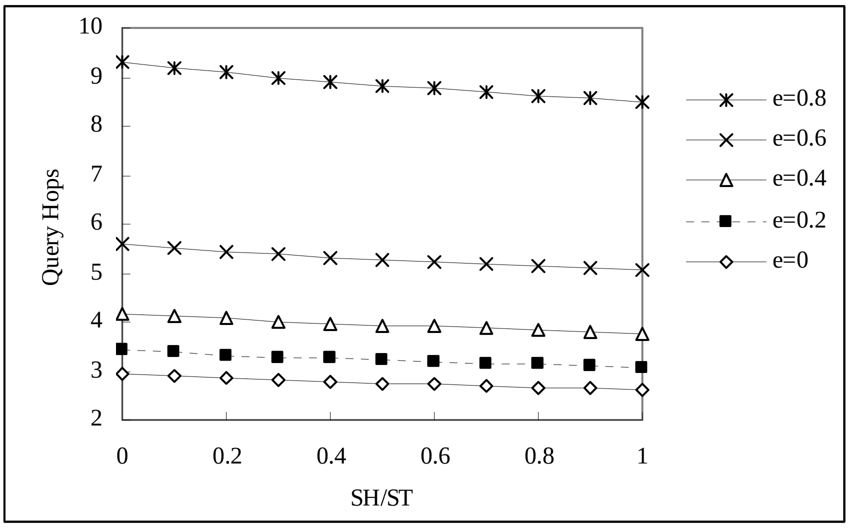
Before giving the simulation results, we present the computation results, using our mathematical model, as in Figure 6.5. Figure (a) plots the values of $E_1(QH)$ and $E_2(QH)$ under different node cache sizes. The results indicate the Pastry network with only the AN.P2P nodes ($E_2(QH)$) achieves less query hops than the network with only the conventional nodes ($E_1(QH)$). Figure (b) plots the $E_3(QH)$ under different S_H/S_T ratios, where S_H=0 means the network consists of only the conventional nodes and the S_H=S_T means the network only contains the AN.P2P nodes. The result shows that with more AN.P2P nodes join the network the average query hops reduce accordingly. Figure (c) presents the $E_3(QH)$ under different node-down rates δ, where the node cache size is 40MB. The results show higher failure rate causes larger $E(QH)$ value. The network with only the AN.P2P nodes (right end of the curve) outperforms the network with only the conventional nodes (left end of the curve) under all node failure rates. The advantage of the AN.P2P method becomes significant under high node-down rate.



(a) Query Hops under Different Storage Capacities



(b) Query Hops under Different SH/ST Ratios



(c) Query Hops under Different Node Failure Rates

Figure 6.5 Computation Results

6.4.3 Simulation Group 1

The first group of simulations compares the general performance difference between the AN.P2P method and the plain Pastry method. In the simulation, we varied the cache size on each node from 5MB to 80MB. Queries are issued into the network till the service or object replicas saturate all node caches, by then the network becomes stable and the system performance is measured. The results are as follow.

Figure 6.6 illustrates the average query hops for the two methods. In the simulation, we set *b*=3, so that the queries can be routed to the home node in $\log_8 5000 \approx 4$ hops. However, due to the replication of service or object, the figure shows both methods can achieve less than 4 hops. When larger caches are applied, the number of query hops reduces, as more replicas are distributed in the network. The figure also shows the AN.P2P method gets less query hops than the plain Pastry method. When larger caches are applied, the difference between the two methods becomes more significant.

Figure 6.7 shows the replica-hit ratio, which is counted as the percentage of queries served by the intermediate nodes that have replicas of the queried services (for the AN.P2P method) or the wanted response objects (for the plain Pastry method). The AN.P2P method achieves about 20% higher replica-hit ratio than the plain Pastry method. It means more queries can be served by the intermediate nodes using the AN.P2P method than the plain Pastry method, because the image services have higher reusability than the customized image presentations. Moreover, with larger caches being applied, the advantage of the AN.P2P method becomes more significant.

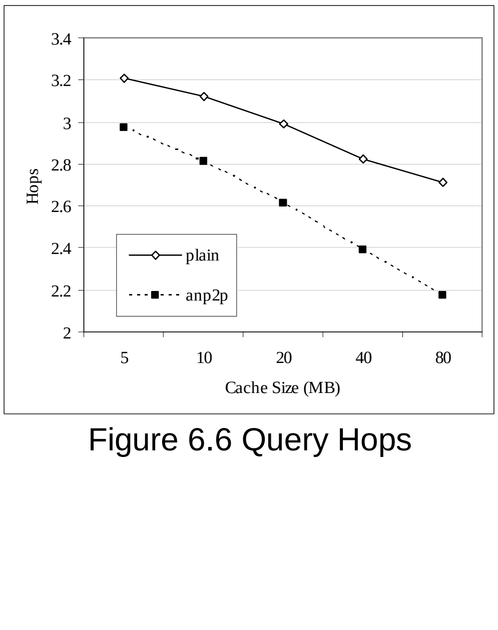


Figure 6.6 Query Hops

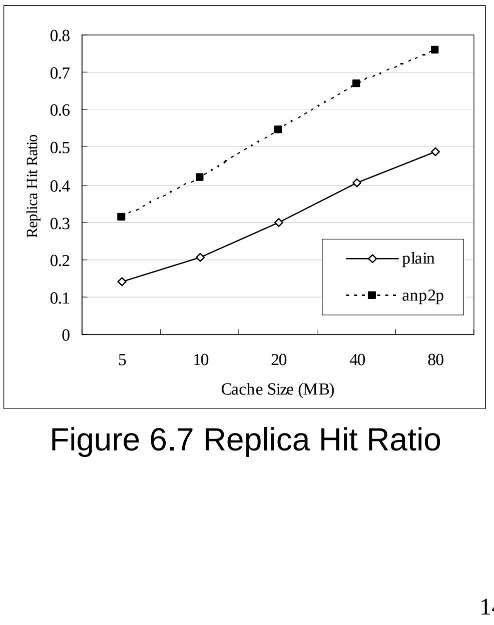


Figure 6.7 Replica Hit Ratio

Figure 6.8 shows the content retrieval distance, which is a ratio obtained from dividing the physical distance between the requesting node and the serving node by the distance between the requesting node and the service's home node. The value of retrieval distance should range from 0 to 1, reflecting the content retrieval proximity within the system, where smaller retrieval distance implies better content retrieval proximity. The result shows the AN.P2P method achieves smaller retrieval distance than the plain Pastry method. It suggests that in an AN.P2P enabled Pastry network the requesting nodes are more likely to get content from the nearby nodes. In the actual network, such better retrieval proximity usually implies less transmission traffic and shorter content download delay.

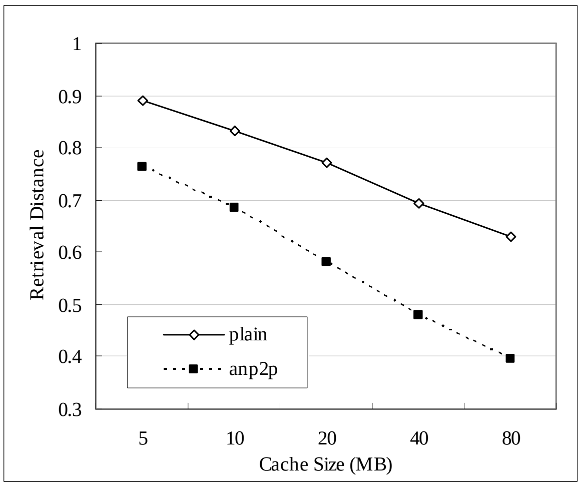


Figure 6.8 Retrieval Distance

This result is due to the co-effect of the AN.P2P method and the proximity-aware routing of Pastry. On one hand, the AN.P2P method can improve the replica reuse through replicating the content services. Hence, the requesting node can find the serving node with less query hops. On the other hand, the proximity aware routing mechanism of Pastry can make sure the message is forwarded from the nearby nodes to the remote nodes. It means with more hops being involved the message always arrives a node that is farther to the requesting node than the previous node in the query path. Combining these two aspects, as the AN.P2P method reduces the number of query hops, the physical distance between the requesting node and the serving node should reduce accordingly.

6.4.4 Simulation Group 2

The second group of simulations measures the change of system performance along the time. We continuously issued queries into the network and measured the system performance for every 1000 queries. In the simulation, the cache size on each node was fixed to 20MB, and the results are as follow.

Figure 6.9 shows the replica hit ratio for both methods. The results indicate the replica hit ratio for the AN.P2P method increases much faster than the plain Pastry method. The figure also shows the hit ratio for the 10 most frequently queried services, which count for 10% of all queries. For the AN.P2P method the hit ratio increases sharply at the beginning of simulation, and after 10,000 queries nearly all the subsequent queries to the top 10 services are served by the intermediate nodes. In contrast, the plain Pastry method cannot achieve comparable hit ratio. The result indicates our AN.P2P method can offload the burst queries in short period of time, and this is important for those popular content services to distribute the bursting load as fast as possible.

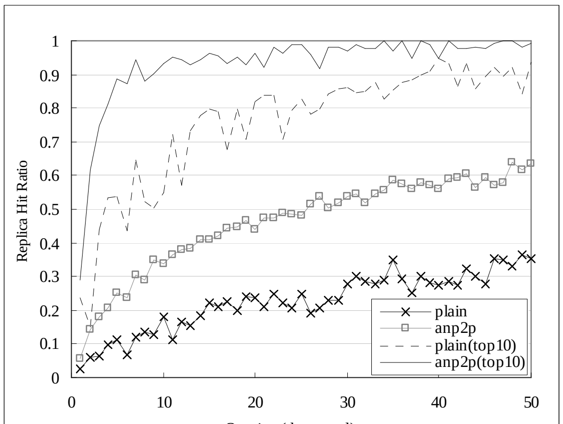


Figure 6.9 Replica Hit Ratio

Figure 6.10 presents the content retrieval distance. At the beginning of simulation, the retrieval distances for both methods equal to one, which means the queries at this stage retrieve the contents from the home nodes. With more queries are issued, the retrieval distance for the AN.P2P method drops much faster than the plain Pastry method. The figure also presents the retrieval distance for the top 10 services. The retrieval distance for the AN.P2P method drops sharply to 20% of the original distance after 10,000 queries are issued, whereas the distance decrement for the plain method is much slower. This result enforces our argument that the AN.P2P method can distribute the burst queries to the intermediate nodes much faster than the plain Pastry method.

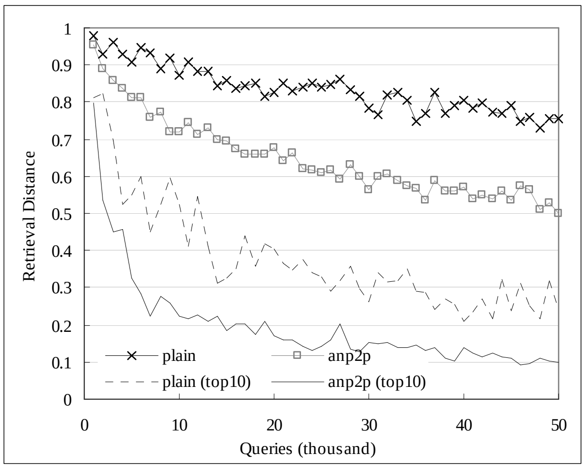


Figure 6.10 Retrieval Distance

Figure 6.11 presents the two methods' replication cost, which is calculated by multiplying the size of the replicated data with the distance between the serving node and the replication node. For the AN.P2P method, the replication size is the total size of the original content and the service applications; and the replication node is the last intermediate node. For the plain Pastry method, the replication size is the size of the response object, and the replication nodes are all intermediate nodes along the query path. The result shows the AN.P2P method causes much less replication cost than the plain Pastry method, as the latter method replicates to all intermediate nodes while the former only replicates to the last intermediate node.

Moreover, the replication cost for the plain Pastry method fluctuates greatly during the simulation period, compared to the graceful reduction of the AN.P2P's the replication cost. This is because the response objects, replicated by the plain method, have poor reusability at the intermediate nodes. As result, these objects are likely to be replicated, evicted and re-replicated. Such "replica thrash" phenomenon is the main reason for the fluctuation of the replication cost. However, the AN.P2P method causes much smoother replication cost during the simulation period, since the replicated services have better reusability on the intermediate nodes so that replica thrash effect is not significant. The graceful reduction of the replication cost is because, according to the proximity routing, the early replications happen between distant nodes while the later replications happen between nearby nodes.

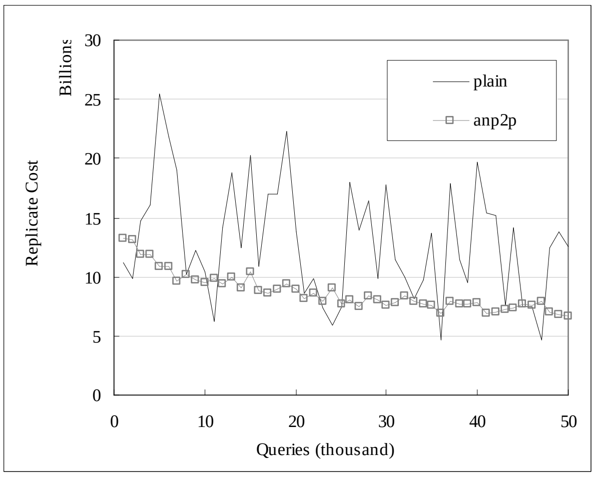


Figure 6.11 Replication Cost

6.4.5 Simulation Group 3

In the third group of simulations, we continuously reduced the percentage of AN.P2P nodes in the whole network. The system performance was measured for different percentages of AN.P2P nodes remaining in the network. The results are given in Figure 6.12, where the cache size on each node is 20MB. The figure shows with the dropping percentage of AN.P2P nodes, the system performance (replica hit ratio and retrieval distance) declines accordingly. When no AN.P2P node is in the network, the whole system degrades to be a conventional Pastry network. The results suggest we can deploy the AN.P2P nodes into the existing Pastry network. With more AN.P2P nodes join, the overall system performance will improve gradually.

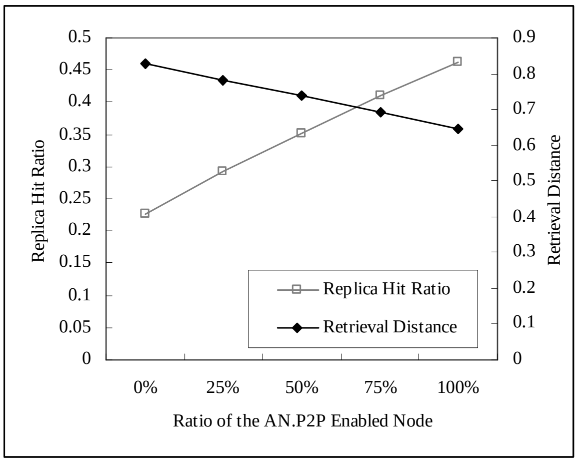


Figure 6.12 Performance Under Different Percentages of AN.P2P Nodes

6.4.6 Simulation Group 4

The last group of simulations compares the two methods' performance under different node down rates. To observe the query failure, we disabled the initial replication function of Pastry. Hence, all services only have one original copy at the beginning of simulation. If the home node happens to leave the network, those queries that cannot find suitable replicas will fail. The simulation results are as follow.

Figure 6.13 presents the percentages of the failed queries under node down rate of 5% and 20%. The figure shows that more queries fail under high node down rate. This phenomenon is significant at the beginning of simulation, because the early queries have to retrieve content from the home nodes. The absence of these nodes directly causes the query to fail. With more queries being issued, the failure ratio reduces, because many of these queries can be served by the intermediate replication nodes those are active in network. In particular, the result indicates the AN.P2P method gets lower query failure than the plain Pastry method. This is because the AN.P2P method can achieve more effective replica reuse through service replication, so that more queries can hit the nodes those have replicas of services and are still active in the network. As the result, the down of home node would not affect the AN.P2P method so significantly as the plain Pastry method.

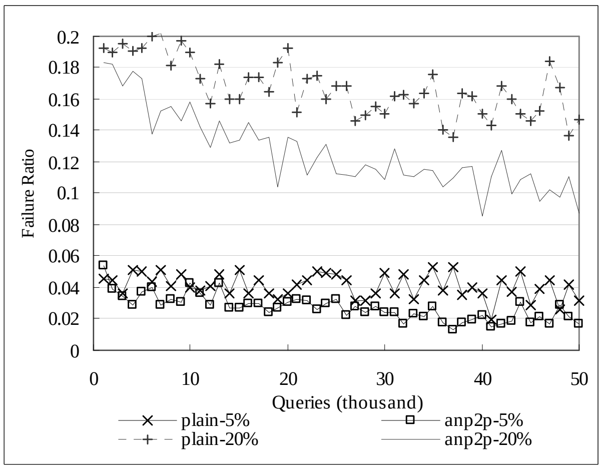


Figure 6.13 Query Failure Ratio

Finally, Figure 6.14 and Figure 6.15 present the replica hit ratio and the retrieval distance for both methods. In general, the results show that the overall system performance for both methods declines with higher node down rate. This is because the node failure or recent node leave causes stale information in the routing table. It makes the messages to be routed to the sub-optimal nodes. Thus, the length of query path will be prolonged, so as the content retrieval distance (Figure 6.15). Meanwhile, as shown in Figure 6.14, the increase of node down rate also reduces the replica hit ratio for both methods, because the downed nodes store replicas of service or object, which otherwise could be reused for the requesting nodes.

Moreover, the results show our AN.P2P method outperforms the plain Pastry method under all node-down rates. In particular, the AN.P2P method at the node down rate of 20% even achieves better performance than the plain Pastry method at the node down rate of 5%. This suggests the AN.P2P supported system is more resistant to the node down, in virtue of service replication.

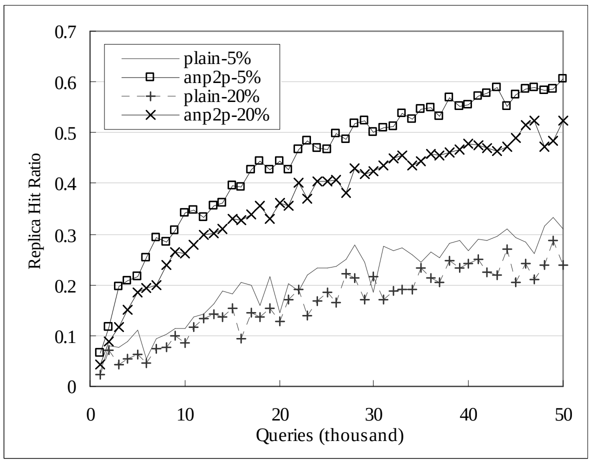


Figure 6.14 Replica Hit Ratio

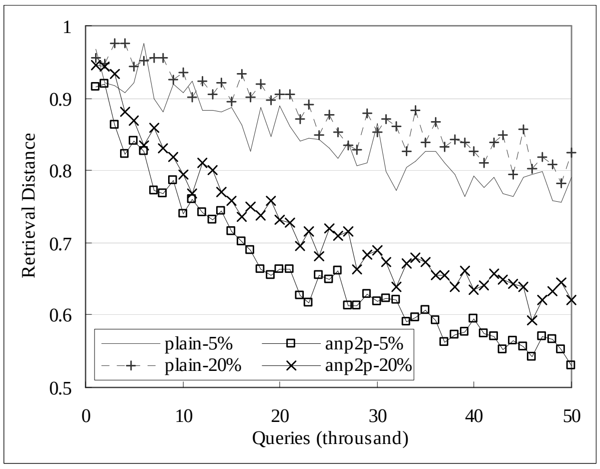


Figure 6.15 Retrieval Distance

6.5 Summary

This chapter proposes the AN.P2P methodology, which applies the Application Networking framework into the P2P network. The method improves the conventional P2P network, which was used for sharing the content objects, by allowing it to populating the content services. We build mathematical model to estimate the performance of the AN.P2P methodology, and the analysis results indicate our method can be applied to a wide range of content services to gain overall performance improvement in the heterogeneous query environment. Finally, our simulation results show the AN.P2P method is more efficient than the conventional P2P method at different aspects.

CHAPTER 7

AN AN.P2P SYSTEM OVER THE PASTRY NETWORK

7.1 Overview

In the fundamental AN.P2P methodology, the replication process does not consider the potential utility and storage overhead of the content service. It causes the system performance to be suboptimal. The basic methodology neither reuses the response content objects fully, so that it needs to execute the content services repeatedly. In this chapter, we propose a concrete AN.P2P system, which provides three specific mechanisms to solve above limitations:

(i) Selective Replication: It utilizes the Pastry routing information to replicate service only to the nodes with high service reuse rate.

(ii) Partial Replication: It extends the selective replication by allowing replicating service workflow partially to improve the replication flexibility.

(iii) Pointers Cache Scheme: It reuses the response objects through propagating the soft-state pointers to these objects, so that the computation load can be reduced.

This chapter first explains our system implementation on Free-Pastry, and then discusses the three mechanisms followed by the simulation results. All simulations are based on the same content service and the network environment as in Chapter 6.

7.1.1 Implementation

We implemented the AN.P2P system based on the Free-Pastry platform [Pastry]. The system structure is shown in Figure 7.1. The system is composed of the Pastry substrate, the AN.P2P platform, the content services, and the client. Besides the basic storage space for the content services published on the present node, the platform also maintains three caches: (i) the service cache, (ii) the response cache, and (iii) the pointers cache. The service cache is to store content services replicated from other nodes, while the response cache and the pointers cache will be explained in the pointers-cache mechanism.

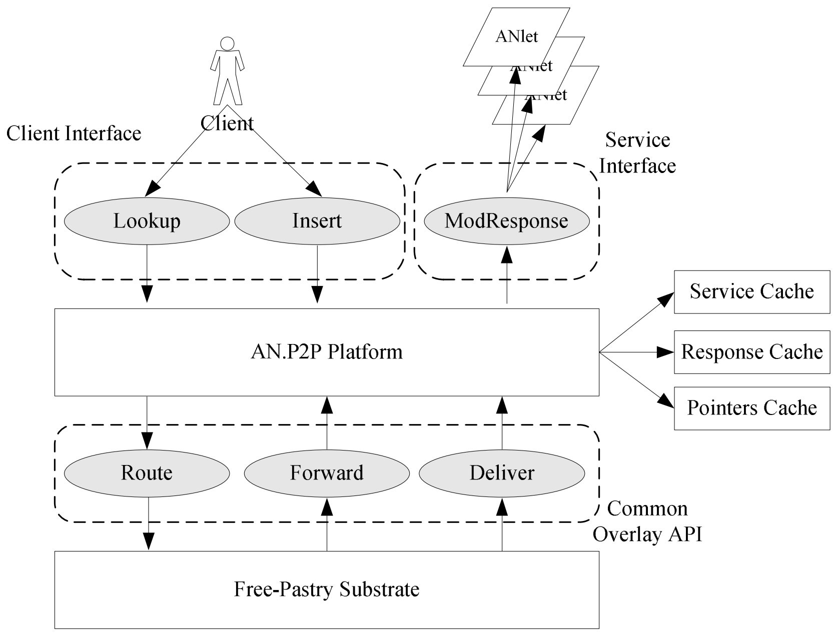


Figure 7.1 AN.P2P Prototype

The system components communicate with each other through the standard client interface, service interface, and P2P substrate interface. The usage of these standard interfaces facilitates adopting the AN.P2P platform onto different P2P substrates and supplying various content services through the AN.P2P platform.

The “client interface” enables the client software to interact with the AN.P2P platform, through two API functions: “insert” and “lookup”. The former function allows the content provider to publish the content service, and the latter function allows the client to initiate a content query into the Pastry network.

The AN.P2P platform interacts with the content services through the ANlet API, as defined in Chapter 4, whereas the HTTP request and response are replaced by the Pastry query and response messages.

The “substrate interface” enables the AN.P2P platform to communicate with the Pastry substrate through the “common overlay API” [DZD03]. The Pastry substrate exports a “route” function to the AN.P2P platform, which can send messages to the P2P network by calling this function. In the opposite direction, the AN.P2P platform exports two functions, “forward” and “deliver”. The Pastry substrate will pass the relayed messages to AN.P2P through calling the former function, and deliver the query to the AN.P2P platform for service execution through the latter function.

A basic query-response-replication process is shown in Figure 7.2, where node 1 is the requesting node, node *n* is the serving node, and node *n*-1 is the last intermediate node. When the client queries a content service, the requesting node sends a query into the network (step 1 ~ step 3). When an intermediate node receives this query, it will check whether it is the home node for the queried id. If not, the query will be passed to the AN.P2P platform of this node through the “forward” function (step 4). The AN.P2P platform checks whether a reusable service is stored in local service cache. If none, the platform notifies the Pastry substrate to forward the query to next hop node. The forwarding process is repeated until the query reaches a node that is the home node of the service or with the service replica (step 5~6). This serving node’s Pastry substrate calls the “deliver” function, asking the AN.P2P platform to execute content service (step 7~8) and send the generated response content to the requesting node (step 9~11). Meanwhile, the serving node replicates the content service to the last intermediate node (step 12~14).

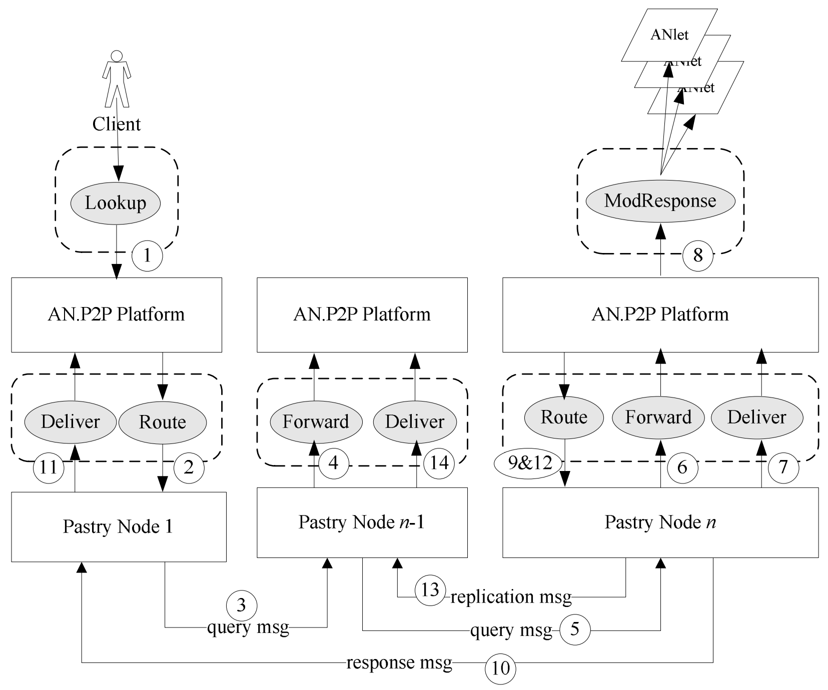


Figure 7.2 AN.P2P's Basic Query-Response-Replication Process

7.2 Selective Replication

In the basic AN.P2P process, the serving node always replicates the service to the last intermediate node. However, without judging the performance enhancement and the corresponding replication overhead, this straightforward method is not beneficial in all situations. Inappropriate replication could put the service to a node that rarely receives queries for this service, and it also wastes the replication node's storage space that otherwise would be more useful if storing higher utility services.

In the Pastry network, the effectiveness of service replication depends on three factors: (i) the numerical distance between the replication node's id and the content service's id; (ii) the physical distance between the serving node and the replication node; and (iii) the size and request rate of the service. Firstly, a node, whose id is numerically close to the id of the content service, is very likely to be involved into the query path to this service. Thus, the potential reuse rate of the service replica should reduce with the increasing numerical distance between the service's id and the replication node's id. Secondly, distributing service replicas to the physically remote nodes will be more beneficial than putting them to the adjacent nodes, because with the dispersed service replicas the requesting nodes are more likely to retrieve content from a nearby replica. Last yet importantly, replication algorithm should give priority to service with small size but high request rate.

Considering these factors, we propose the selective replication method. It requires the serving node to encapsulate the meta-information about the content service to be replicated into the replication message, and send it to the last intermediate node. The information includes the service's id, the components of the service, and the total size of these components. Upon receiving the replication message, the node will judge whether to replicate this service or how to replace the cached services if necessary. We proposed the selective GreedyDual-Size (GDS) replication algorithm and the selective LFU replication algorithm to perform the replication judgment.

7.2.1 Selective GDS Replication Algorithm

The selective GDS replication algorithm is based on the GreedyDual-Size (GDS) caching algorithm [CI97], which incorporates the caching locality with the data's retrieval cost and storage size concerns. The original GDS algorithm assigns the data to be stored an initial cost as in Eq.7.1. Since we need to consider the numerical and physical distance of replication, our selective GDS algorithm assigns the initial cost for a content service as in Eq.7.2.

$$Cost = 1/Sizeof(data) \qquad \text{(Eq.7.1)}$$

$$Cost = \frac{(2^b)^{shl(service\_id, local\_id)} \times dist(serving, local)}{Sizeof(service)} \qquad \text{(Eq.7.2)}$$

In the equation, $shl(service\_id, local\_id)$ denotes the length of the shared prefix between the service's id and the id of the replication node. According to the

Pastry algorithm, the likelihood that the node is involved in the query path to the service is proportional to the value of $(2^b)^{shl(service\_id, local\_id)}$, which reflects the proximity of the node and the content service in the id ring. Meanwhile, $dist(serving, local)$ denotes the physical distance between the serving node and the present node (the replication node).

Our algorithm uses an *isReplicate*() function, as shown in Figure 7.3, to determine whether the present node should replicate a content service. The algorithm first computes the initial GDS cost for the service (line 03~04). Then, it examines the free space in the service cache. If there is sufficient space, the node will replicate this service (line 05~07). However, if the cache does not have enough space, some stored services "may" be evicted. To make a decision, the algorithm continuously chooses the cached service with the least GDS cost, until the total size of these services plus the free cache space is larger than the size of the new service (line 08~14). Then the algorithm compares the cost of the new service to the maximum cost of the chosen services. If the stored service has larger cost (line 15~16), the new service is not beneficial enough and should not be replicated. However, if the new service has larger cost (line 17~18), it will be replicated and the chosen services will be evicted from the cache.

According to the GDS algorithm, when a cache replacement is performed, the cost value of all remaining services should be reduced by the maximum cost of the evicted services. In addition, when a service is accessed, its cost will be restored to the initial value. Through above adjustment, the services with high utility will be restored to the initial cost frequently, while the costs for the low utility services will decrease gradually.

```
(01) Boolean isReplicate ($Wf(Obj_1, ANlet_1, ANlet_2,\dots, ANlet_f)$)
(02) {
(03)     $Sizeof(Service) = Sizeof(Obj_1) + \sum_{i=1}^{f} Sizeof(ANlet_i)$
(04)     $Cost(Service) = \frac{(2^b)^{shl(service\_id, local\_id)} \times dist(serving, local)}{Sizeof(service)}$
(05)     AvailableSize = Cache.Free _Size ( );
(06)     if (Sizeof (Service) < AvailableSize)
(07)         return TRUE;
(08)     ReplaceSet = Ø;
(09)     MaxCost = 0;
(10)     do {
(11)         ReplaceService = Cache.Least_Cost_Service ( );
(12)         AvailableSize += Sizeof (ReplaceService);
(13)         MaxCost = Cache.Cost (ReplaceService);
(14)     } while (Sizeof (Service) > AvailableSize);
(15)     if (Cost (Service) < MaxCost)
(16)         return FALSE;
(17)     else
(18)         return TRUE;
(19) }
```

Figure 7.3 isReplicate ( ) for Selective GDS Replication

7.2.2 Selective LFU Replication Algorithm

The selective LFU replication algorithm is based on the LFU algorithm, and it assigns a cost value for each service as in Eq.7.3, where the *counter* records the times the service has been accessed since its replication, *Sizeof*(*service*) denotes the size of service, and $(2^b)^{shl(service\_id, local\_id)}$ denotes the numerical proximity of the replication node and the service, and $dist(serving, local)$ denotes the physical distance between the replication node and the serving node.

$$Cost = \frac{counter \times (2^b)^{shl(service\_id, local\_id)} \times dist(serving, local)}{Sizeof(service)} \qquad \text{(Eq.7.3)}$$

The algorithm also uses an *isReplicate*() function, as shown in Figure 7.4, to determine whether the node should replicate a service. The algorithm first computes the storage size and the initial cost for the new service (line 03~04), where the initial value of counter is set to one. If the service cache has sufficient free space, the node will replicate the service directly (line 05~07). However, if the cache does not have enough free space, the algorithm continuously chooses the cached service with the least cost until the total size of these services plus the free cache space is larger than the size of the new service (line 08~14). Then the algorithm compares the total cost of the chosen services with the cost of the new service. If the new service has larger cost, the node should replicate the new service and evict those chosen services (line 15~16). However, if the cost of the new service is smaller, it will not be replicated (line 17~18).

```
(01) Boolean isReplicate ($Wf(Obj_1, ANlet_1, ANlet_2,\ldots, ANlet_f)$)
(02) {
(03)     $Sizeof(Service) = Sizeof(Obj_1) + \sum_{i=1}^{f} Sizeof(ANlet_i)$
(04)     $Cost = \frac{counter \times (2^b)^{shl(service\_id, local\_id)} \times dist(serving, local)}{Sizeof(service)}$
(05)     AvailableSize = Cache.Free _Size( );
(06)     if(Sizeof (Service) < AvailableSize)
(07)        return TRUE;
(08)     ReplaceSet = Ø;
(09)     TotalCost = 0;
(10)     do {
(11)          ReplaceService = Cache.Least_Cost_Service( );
(12)          AvailableSize += Sizeof (ReplaceService);
(13)          TotalCost += Cache.Cost (ReplaceService);
(14)     } while (Sizeof (Service) > AvailableSize);
(15)     if( TotalCost < Cost (Service))
(16)        return TRUE;
(17)     else
(18)        return FALSE;
(19) }
```

Figure 7.4 isReplicate( ) for Selective LFU Replication

7.2.3 Simulation Results

Our simulation compares the selective GDS and LFU replication methods with the normal GDS and LFU replication methods in AN.P2P. The simulation results are as follow, where the result for the normal LRU replication is also given to ease readers to refer the results in Chapter 6.

Figure 7.5 shows the replication costs for all methods. The selective replication methods achieve much less replication cost than the normal replication methods. This is because the selective methods only populate services to nodes with sufficient service reuse rate, whereas the normal replication methods replicate service for all queries. The three normal methods cause comparable replication costs. The costs firstly increase and then drop with the increase of cache size, and the reason for such trend is as follows. When the cache size is smaller than 10MB (the largest size of original images), some large services cannot be replicated due to the size limitation of cache. With the increase of cache size, more services can be replicated and the system replication cost increases accordingly. When the cache size exceeds 10MB, all services can be replicated and the cache size is no longer the determinant for the replication cost. With large caches being applied, the service penetration in the network should increase correspondingly, which makes the final replications are more likely to happen between nearby nodes. It results in the measured replication cost to decrease. In contrast, the replication costs for the selective methods enhance slowly with the increase of cache size, because the selective replication keeps the replication cost low and it naturally increases with the increasing cache size as more services will be replicated.

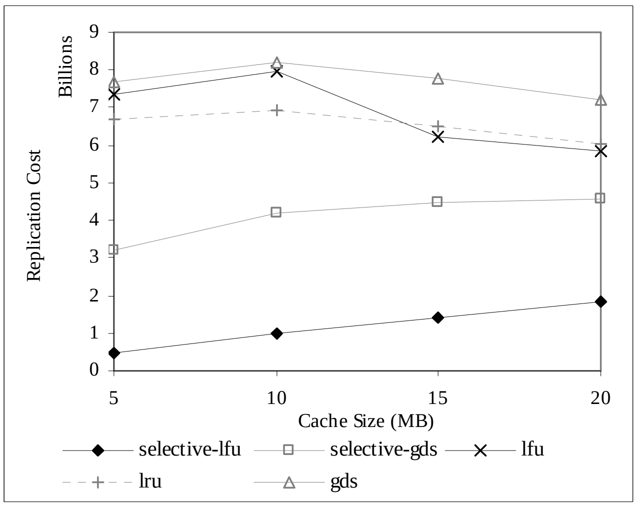


Figure 7.5 Replication Costs against Different Cache Sizes

Figure 7.6 shows the replication costs against the number of queries issued, where the cache sizes of all nodes are 20MB and the replication cost is measured for every 4,000 queries. At the beginning, all methods cause the similar replication cost, because most caches are empty and ready to be filled. After 50,000 queries, the replication costs for the normal methods stabilize to the constant values, which are the final replication thrash traffic for these methods. In contrast, the replication costs for the selective methods reduce continuously in the simulation period. This is because the selective methods can replicate services to high utility nodes and it makes better service distribution in the network. Thus, the replication thrash traffic is much lower than that of the normal methods, and it reduces with more queries are issued.

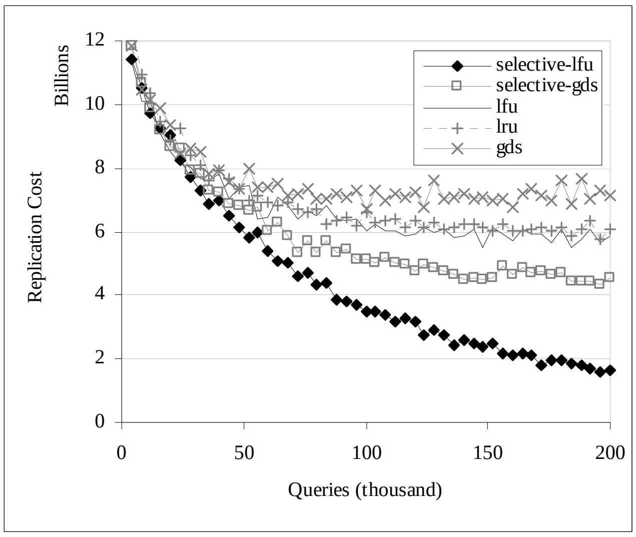


Figure 7.6 Replication Costs against the Number of Queries

We also measured the service replica hit ratio and the content retrieval distance for different methods, as shown in Figure 7.7 and Figure 7.8. The results show the two selective methods improve the overall performance more significantly than the normal methods. It is because the selective replication methods can replicate services to the selected nodes with high reuse rate, so that the resulting service distribution in the network is more beneficial to efficient content delivery. The two figures also show that the selective LFU method outperforms the selective GDS method.

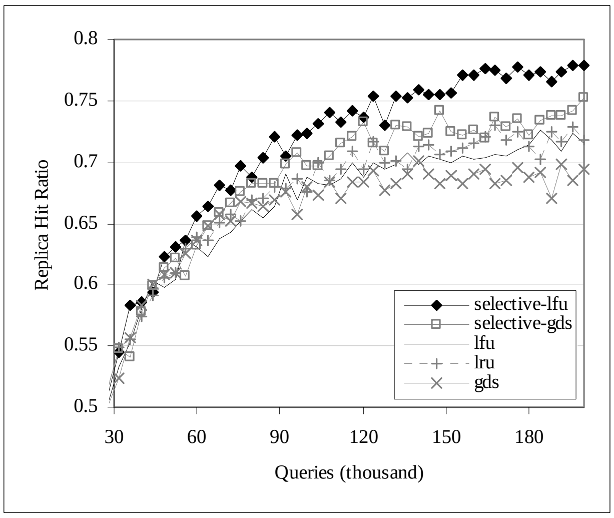


Figure 7.7 Replica Hit Ratio

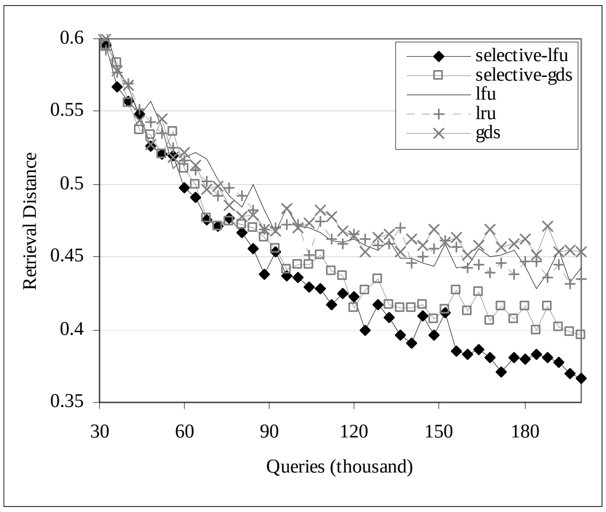


Figure 7.8 Retrieval Distance

In summary, the simulation results indicate that the selective replication algorithms have two advantages: (i) they allow the system to avoid some unbeneficial service replications so that the overall replication cost can be reduced; (ii) they make all services to be replicated to the nodes with high reuse rate, so that the resulting content delivery performance can be improved. In addition, the results also show that the selective LFU method outperforms the selective GDS method by achieving less replication cost, higher replica hit ratio and lower retrieval distance. It suggests that through directly counting the query frequency of service, the selective LFU method is more effective to estimate the service utility than the selective GDS method.

7.3 Partial Service Replication

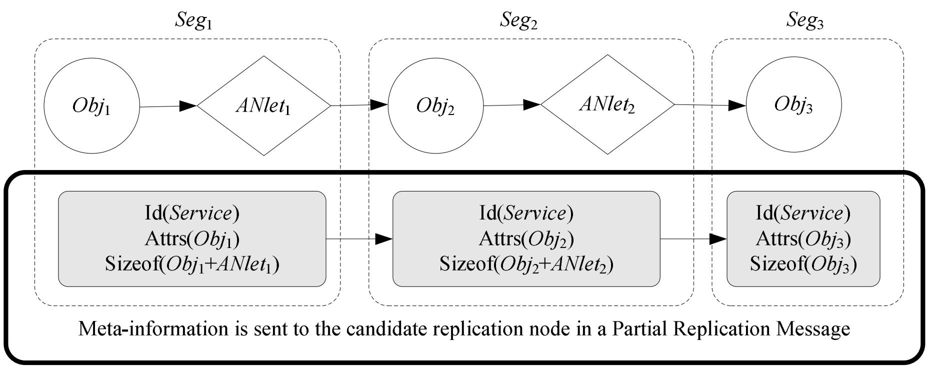
7.3.1 Partial Replication Algorithm

The previous replication methods treat content service as an atomic unit and replicate it as a whole. As explained, the AN.P2P system organizes each content service as a workflow. Different sub-sets of service workflow, which starts from a particular version of content object and passes through the subsequent workflow segments, usually have different reusability and replication overhead at different network nodes. If we can replicate the partial content services according to their specific reusability and replication overhead, the AN.P2P system will be able to achieve cost-effective service replication and be more adaptive to the changing environment.

Due to the changing characters of the P2P network, it is infeasible to find a well-defined solution to resolve the optimal partial service replication in the whole network. Even for a comparatively stable network, the huge numbers of nodes and content services also make this optimization problem intractable. Therefore, we propose a partial service replication algorithm that derives the replication strategy in the best-effort manner. The algorithm is performed independently at any node, and allow it to replicate as many workflow segments as possible given they are beneficial enough.

The algorithm requires the serving node, the sender of replication message, to put the meta-information about the service workflow into the replication message. The meta-information is specified as a sequence of blocks, each refers to a segment in the workflow. The blocks are listed in the same order as the corresponding segments in the workflow. Each block specifies the service id, the attributes of the segment's input content object, the size of the input object, and the id and size for the segment's application.

In the AN.P2P system, a serving node always intends to replicate a content service when it replies a query. A natural way to compose the replication message is to include the meta-information for the service workflow used and the content objects generated for the just query. An example is given in Figure 7.9, where we suppose a node stores a service that starts from Obj_1 and contains two applications: $ANlet_1$ and $ANlet_2$. (*Note* †: *as the service can be partially replicated, the workflow we discuss may not be the full original service workflow*) When the node serves a query, it executes this workflow to generate an intermediate object *Obj*$_2$ and a final response object *Obj*$_3$. In this situation, the replication message should include the meta-information for *Obj*$_1$, *Obj*$_2$, *Obj*$_3$, *ANlet*$_1$ and *ANlet*$_2$. If we express an ANlet with its input object as a segment, the resulting replication message is shown as in the bold square in the figure, and the shaded squares denote the meta-information blocks for segments *Seg*$_1$, *Seg*$_2$, and *Seg*$_3$ respectively.



Seg_1
Seg_2
Seg_3
Obj_1
$ANlet_1$
Obj_2
$ANlet_2$
Obj_3
Id(*Service*) Attrs(Obj_1) Sizeof(Obj_1+$ANlet_1$)
Id(*Service*) Attrs(Obj_2) Sizeof(Obj_2+$ANlet_2$)
Id(*Service*) Attrs(Obj_3) Sizeof(Obj_3)
Meta-information is sent to the candidate replication node in a Partial Replication Message

Figure 7.9 Partial Replication Message

The replication node has four possibilities to replicate the service above:

(i) Replicate *Seg*$_1$→*Seg*$_2$→*Seg*$_3$: the replication node needs to fetch *Obj*$_1$ and applications *ANlet*$_1$ and *ANlet*$_2$ from the sender of the replication message.

(ii) Replicate *Seg*$_2$→*Seg*$_3$: the replication node needs to fetch the intermediate object *Obj*$_2$ and the second application *ANlet*$_2$.

(iii)Replicate *Seg*$_3$: the replication node only fetches the response object *Obj*$_3$.

(iv)None: the replication node does not replicate anything.

Our partial replication algorithm is shown in Figure 7.10, which directs the replication node to decide how many workflow segments to be replicated. The input to the algorithm is the meta-information for the workflow to be replicated. We skip lines 03~04 first. The algorithm starts from examining the entire workflow, *Wf*(*Obj*$_1$, *ANlet*$_1$, *Obj*$_2$, *ANlet*$_2$,…, *Obj*$_{f\text{-}1}$,*Anlet*$_{f\text{-}1}$,*Obj*$_f$). The partial replication algorithm relies on the selective replication algorithm to judge whether to replicate a specific workflow.

Thus, the “isReplicate” function (line 08~09) is provided by the selective replication algorithm used, such as Figure 7.3 or Figure 7.4. If the workflow is eligible to be replicated, the algorithm returns the present workflow (line 10~11) to the local node, which will fetch the corresponding workflow components from the network and store them locally. However, if the current workflow is not eligible for replication, the algorithm will remove the first segment (i.e. Obj_1 and $ANlet_1$) and form a partitioned workflow $Wf(Obj_2, ANlet_2, \dots, Obj_{f\text{-}1}, Anlet_{f\text{-}1}, Obj_f)$ (line 12~13). Then, the algorithm reevaluates the new workflow. Such iteration will continue until an eligible partial workflow is found or no more segments left in the workflow.

```
(01) Partial-Replication-Judgment ($Wf(Obj_1, ANlet_1, Obj_2, ANlet_2, \dots, Obj_{f-1}, Anlet_{f-1}, Obj_f)$)
(02) {
(03)    if ( Cache.isRedundantWorkflow (Wf) == TRUE)
(04)       return FALSE;
(05)    idx = 1;
(06)    while(idx <= f)
(07)    {
(08)       Candidate_Wf = $Wf(Obj_{idx}, ANlet_{idx}, \dots, Obj_{f-1}, Anlet_{f-1}, Obj_f)$;
(09)       result = isReplicate (Candidate_Wf);
(10)       if(result)
(11)          return Candidate_Wf;
(12)       else
(13)          idx = idx + 1;
(14)    }
(15)    return NULL;
(16) }
```

Figure 7.10 Partial Replication Judgment Algorithm

7.3.2 Partial Service Caching Scheme

The partial replication causes different versions of the same content service to be stored in the service cache, and it is important to identify and manage these content services. As each content service starts from a unique input object, we can identify a content service through identifying its input object.

Our method is based the metadata approach proposed in Chapter 3. We assign each service a “service metadata”, which includes the service’s id and the attributes for the input object to this service. Each attribute composes of the attribute’s name and the attribute's value. The metadata is created and manipulated by the workflow applications, and is carried by the partial replication message. For the example image service in Chapter 6, the metadata for the original service only contains the service's id. For the partial service starting from the second ANlet while the first ANlet already being executed to resize the original image, the metadata contains the service id and the dimensions for the resized image, e.g. {id, (width, 800), (height, 600)}.

Using the service metadata, the service cache can find a reusable service replica by matching the replica's metadata and the query. The matching service replica should have the same id as the queried id, and all the common metadata attributes of the cached service and the query parameters should match with each other.

To cover more queries, the service cache does not store redundant services, one of which can be generated from the other one. Our partial-service checking criterion deems service A to be a partial service of service B if the metadata for B is subsumed by the metadata for A, and at the same time, the workflow segments of A is subsumed by the workflow segments of B. For the example in Figure 7.9, for service $Obj_1 \rightarrow ANlet_1 \rightarrow ANlet_2$, the attribute is {id, attr(Obj_1)} and the workflow is {Seg_1, Seg_2}; for service $Obj_2 \rightarrow ANlet_2$ the attribute is {id, attr(Obj_1), attr(Obj_2)} and the workflow is {Seg_2}. We can infer the latter service is a partial service of the former, as $\{id, attr(Obj_1), attr(Obj_2)\} \supset \{id, attr(Obj_1)\}$ and $\{Seg_2\} \subset \{Seg_1, Seg_2\}$.

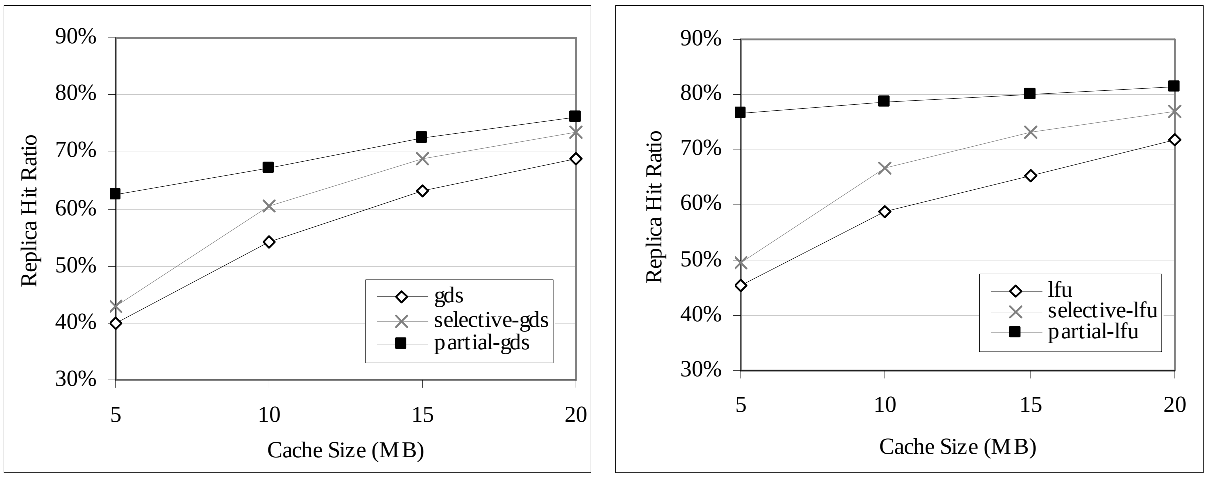
When the service cache determines whether to store a new service, it checks if the new service is a partial service of one service that is already cached. If so, the new service should not be replicated. This is why the replication algorithm in Figure 7.10 performs a pre-checking in lines 03~04. In addition, after the cache determines to store a new service, it will evict the cached services that are the partial service of the new service.

7.3.3 Simulation Results

Our simulation compares the partial replication methods with their corresponding selective and normal replication methods. The results are as follow.

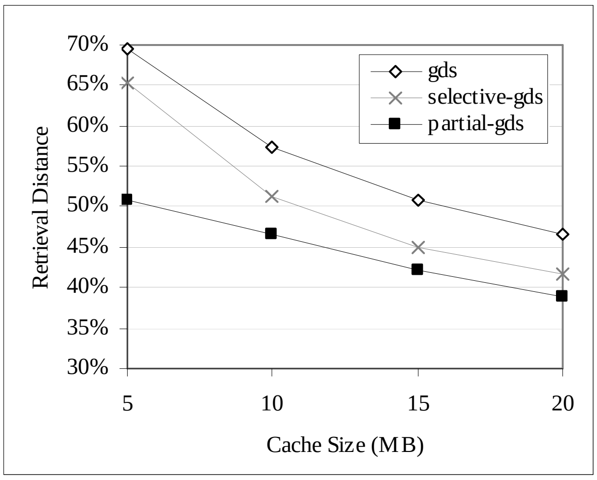
Figure 7.11 and Figure 7.12 show the service replica hit ratio and the content retrieval distance, and we present the results for the GDS-based and the LFU-based methods in separate figures. The results show that, in each category, the partial replication method outperforms the selective method that in turn outperforms the normal method. It indicates the partial replication methods can further improve the system performance through fine-grained service replication.

Moreover, the advantage of the partial replication methods is particularly significant when the cache size is small. This is because when the small cache is applied, the selective method and normal method cannot replicate many large services, whereas these services can be partially replicated using the partial replication method. This advantage also implies the usage of partial replication method can reduce the system's requirement for large sized cache but achieve comparable performance with the selective and normal replication methods with large cache used.

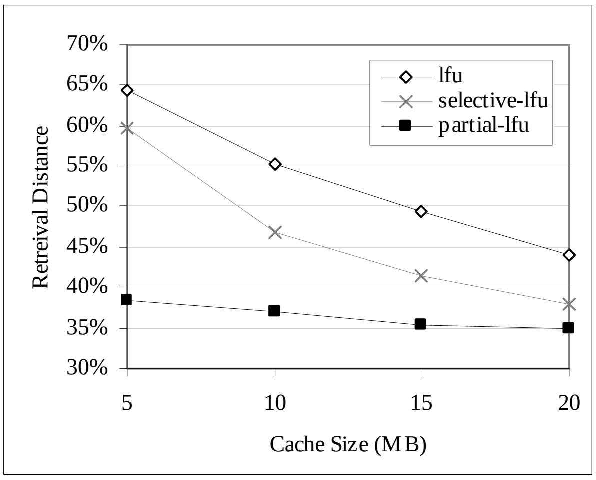


(a) GDS-based Replication Methods (b) LFU-based Replication Methods

Figure 7.11 Replica Hit Ratio



(a) GDS-based Replication Methods



(b) LFU-based Replication Methods

Figure 7.12 Retrieval Distance

The results also show that the partial LFU method outperforms the partial GDS method. This further supports our previous argument that the LFU algorithm is more effective than the GDS algorithm for service replication in the Pastry network. More results about the LFU-based replication methods are shown in Figure 7.13 and Figure 7.14, where the node's cache size is set to 5MB. At the beginning of simulation all methods achieve similar performance, because most queries are served by the home nodes in this period. After 50,000 queries are issued, both the selective method and the normal method reach the maximum performance, whereas the system performance of the partial replication method continues to improve. This is because, in the rear phase of simulation, the partial replication methods can still replicate many partial services in the network, so that the system performance can continue to improve.

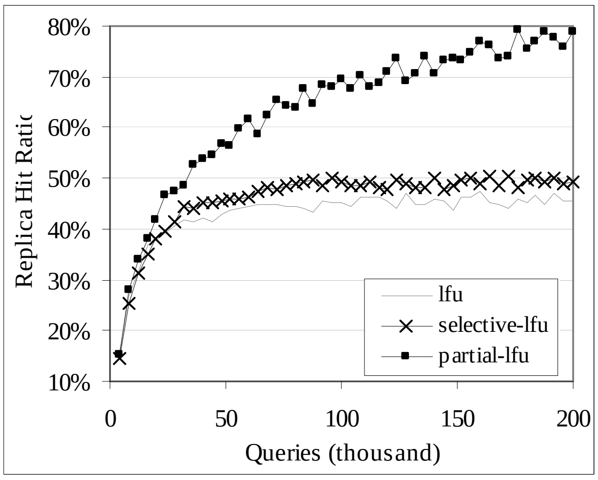


Figure 7.13 Replica Hit Ratio

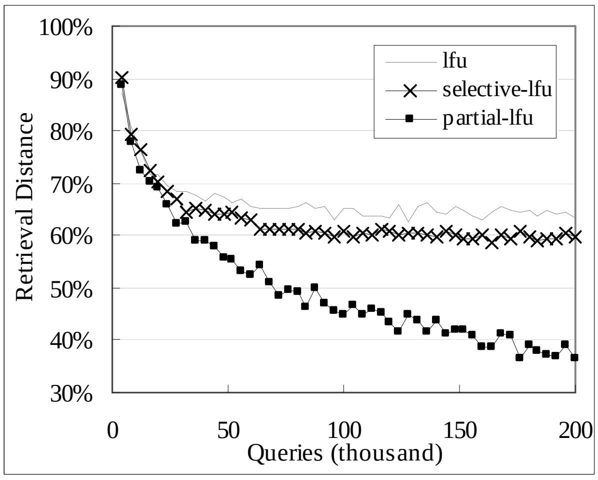


Figure 7.14 Retrieval Distance

Figure 7.15 shows the replication costs of the LFU-based replication methods. The partial and selective methods cause significantly lower replication cost than the normal method, because both methods replicate to high utility nodes and cause less thrash replication traffic. In specific, the replication cost for the partial method is slightly higher than that of the selective method, because it can partially replicate some services, which are unbeneficial as a whole and rejected to be replicated by the selective method.

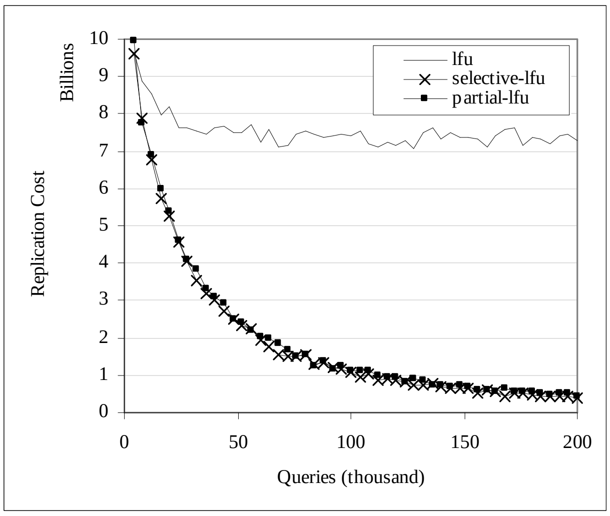


Figure 7.15 Replication Costs

Finally, Figure 7.16 presents the computation loads for the two partial replication methods and the other replication methods (including the normal and selective replication methods). In our simulation environment, the actual system computation load cannot be got directly. Therefore, we count the times the ANlet applications are executed by all nodes to estimate the system computation load. Since we count the average computation load for every 4,000 queries and our service workflow contains 2 ANlets, the figure shows that the measured computation load for the normal and selective replication methods is strictly 8,000. It means there is no computation saving for using these methods. In comparison, the two partial replication methods achieve less computation load because some queries can be served by the partial service that starts from the intermediate object. Furthermore, the computation loads of the partial replication methods reduce continuously. It means that, with more queries being issued, more numbers of partial services are populated, so that the overall computation load is reduced accordingly.

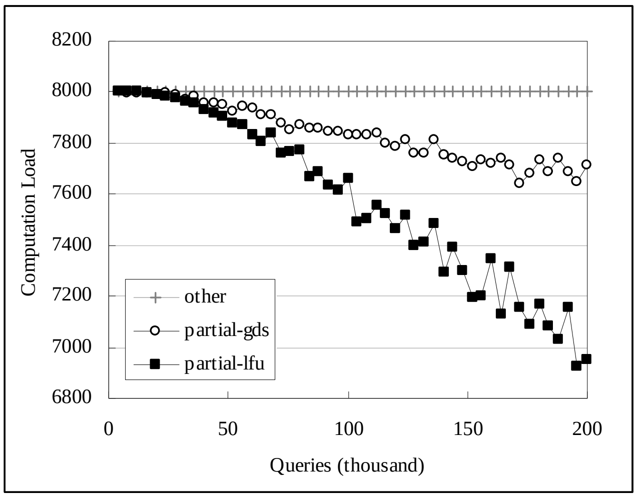


Figure 7.16 Computation Loads

In summary, the simulation results show that the partial replication methods are effective to improve the overall performance through flexible service replication. They introduce slightly higher replication cost than the selective replication method, but the significant improvement on the content retrieval distance, the replica hit ratio, and the computation load. The partial replication methods exhibit particularly good performance when small service cache is applied, and it suggests these methods can reduce the system's requirement on the cache size.

7.4 Pointers Cache

This section discusses another important issue in the system: how to store and reuse the response objects. This can avoid some repeated service executions and reduce the computation load in the system. A straightforward method is to use the "reverse cache". This method requires each node maintain a reverse cache to store the execution results of local services. When this node receives a new query for the same content object, it can reuse the cached object rather than executing the service again. However, the drawback of this method is that the objects stored in one reverse cache can only be reused by the local node. This limits the overall object reuse rate in the network.

We propose the "pointers cache" method that enables to reuse the response objects from distributed nodes. The method requires each node to maintain a response cache to store the retrieved response objects. Asking the requesting nodes to store the response objects is because: (i) these nodes are the natural places to store the response objects; (ii) letting the content consumer to share some of its resource is helpful to efficient resource utilization in the P2P network.

Now, the problem is how to reuse these objects cached on the distributed nodes. Because the requesting nodes are randomly distributed in the id ring, storing the response objects on them will result in the random distribution of objects within the id ring. According to the Pastry routing protocol, such random object distribution can only achieve very limited replica reuse. To solve this problem, our method lets all nodes also maintain a "pointers cache", which stores a set of pointers, each tells which node is probably caching what object. When a requesting node retrieves a response object, it stores the object in the response cache and sends an ACK message into the network. The ACK message carries the same id as the retrieved response object. The message also includes the attributes about the retrieved object and the address of this requesting node. As the ACK message has the same id as the corresponding query message that fetched the object, it will pass through the same intermediate nodes as the corresponding query message and finally reach the same serving node that generated the response object. When these nodes receive the ACK message, each of them will create a pointer in the local pointers cache. The pointer information includes the id and the attributes of the referred content object, and the address of the ACK message sender.

When a node receives a query message, it should operate according to the flowchart in Figure 7.17. It first checks if the local response cache stores the queried object. If the query hits the response cache, the object is sent back to the requesting node directly. Otherwise, the node searches in the pointers cache to see whether other nodes are probably caching the objects. If the query hits the pointers cache, it is forwarded to that pointed node. However, if no pointer is useful, the node will continue the normal AN.P2P process – it serves the query using the local service replica or routes the query to another node.

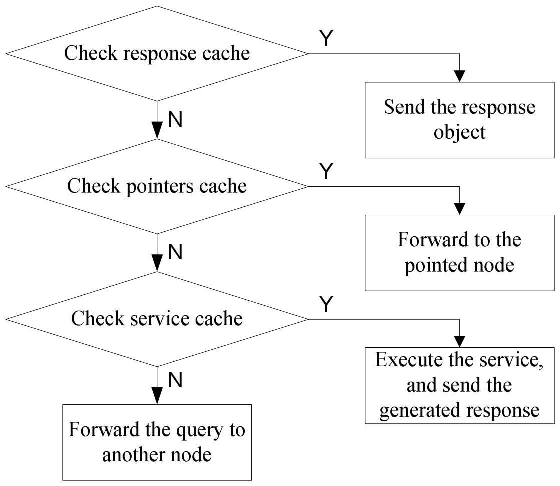

Check response cache
Y
Send the response object
N
Check pointers cache
Y
Forward to the pointed node
N
Check service cache
Y
Execute the service, and send the generated response
N
Forward the query to another node

Figure 7.17 Operations Flowchart

As each node can only provide limited space to store the response objects, the formerly retrieved objects could be replaced by the recently retrieved objects. Meanwhile, any node may leave the network due to various reasons. These two aspects cause the pointers in the pointers cache to become stale. Hence, before a node forwards a query to the pointed node, it needs to send a probe message to the pointed node, asking whether this node is stilling storing the claimed object. If getting the positive response, the node will forward the query to the pointed node. However, if the pointed node is no longer active or returns a negative reply, the node will remove the stale pointer and try other pointers.

An important issue about the pointers cache method is how this method would affect the final content retrieval distance. We argue that, even though this method may introduce an additional query hop, it will not enhance the content retrieval distance significantly and actually can reduce it in many situations.

Figure 7.18 shows a Pastry network, where node p_1 sends a query that is forwarded to nodes p_2 and p_3. We suppose p_3 has a service replica and it also keeps pointers to nodes p_4, p_5 and p_6 for the queried content. If p_3 receives the query, it will forward the query to any of p_4, p_5 and p_6, rather than executing the local service. Let r denote the distance between p_2 and p_3. In the Pastry network, the expected distance traveled by a message during each successive routing step is exponentially increasing with base 2^b. Thus, the nodes pointed by p_3 should reside in a circle of radium $r' = \lim_{i=1}^{\infty}(r + \frac{1}{2^b}r + \frac{1}{(2^b)^2}r + \cdots + \frac{1}{(2^b)^i}r) = \frac{2^b}{2^b - 1}r$ and centered at p_3. Given the default value of b is 4, we have $r' = \frac{16}{15}r$. Thus, in the worse case the maximum distance between the pointed node and p_2 is $r + r' = \frac{31}{15}r$. Hence, we can conclude the usage of pointers would not increase the content retrieval distance greatly. In actual implementation, we can choose a node that is the nearest to the requesting node (e.g. node p_6 in the figure), by using the distance measurement function of Pastry. This can significantly reduce the retrieval distance.

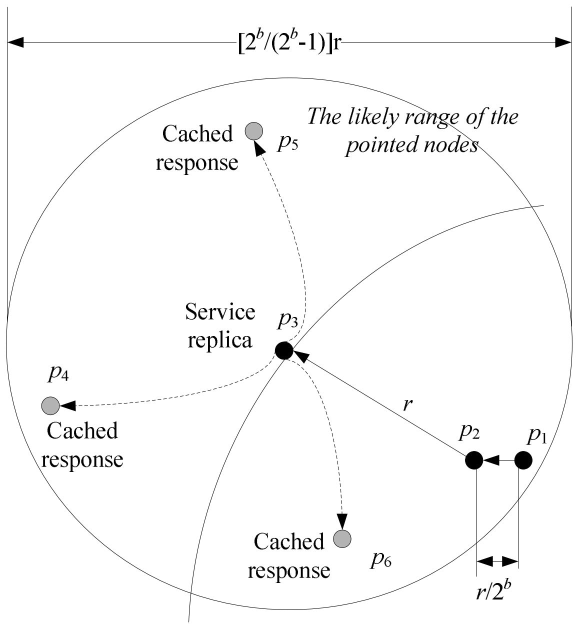


Figure 7.18 Range of the Pointed Nodes (1)

Moreover, as the ACK message are populated to all nodes in the path, it makes some query messages to be forwarded to the pointed node before it reaches a node with the service replica. An example is shown in Figure 7.19, where only p_7 stores the service replica and p_3 keeps pointers for the queried content. In this case, p_3 can directly determine a node to reply p_1, and the distance between p_3 with the pointed node is $r' = \frac{2^b}{2^b - 1} r$. However, without using the pointers, the query would be routed to p_7, and the distance between p_3 and p_7 is $r'' = 2^b r$. Given b = 4, we have $r' = \frac{16}{15} r$ and $r'' = 16r$. As $r' << r''$, it indicates that the content retrieval distance can be greatly reduced.

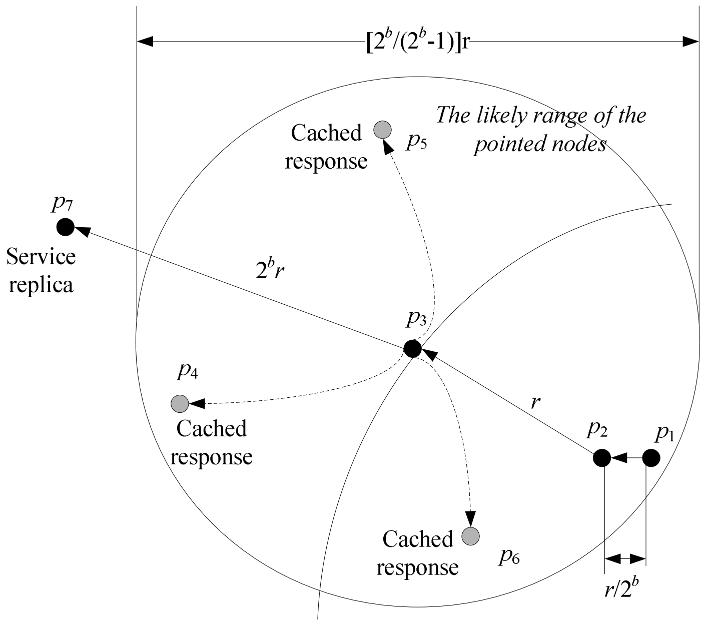


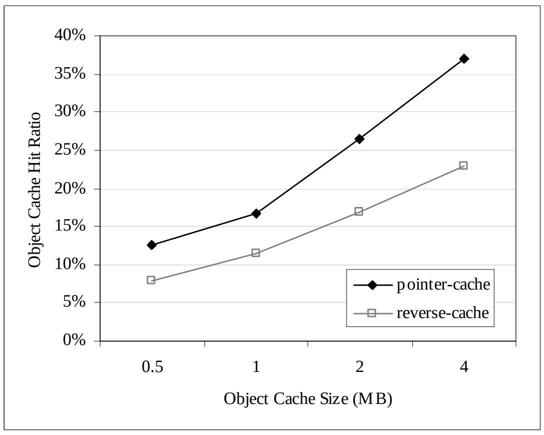
Figure 7.19 Range of the Pointed Nodes (2)

7.4.1 Simulation Results

Our simulation compares the pointers cache method and the reverse cache method. For precise comparison, we used the selective LFU method to replicate services, so that all computation load reduction is due to the usage of the compared two methods. The simulation results are as follow.

For simplicity, we call the reverse cache and the response cache as the "object cache". The hit ratio for object cache denotes the percentages of queries served by the stored response objects instead of executing the content services. Figure 7.20 shows the object cache hit ratio for the two methods, and it indicates the pointers cache method achieves higher hit ratio than the reverse cache method. This is because, in the pointers cache method, the propagated pointers make the objects cached at a node to be reused by other nodes, whereas the reverse cache method can only reuse objects locally.

Figure 7.21 shows the system computation load. The results indicate both methods can continuously reduce the computation load with the increase of object cache size. The pointers cache method outperforms the reverse cache method, because this method achieves higher object reuse than the reverse cache method.



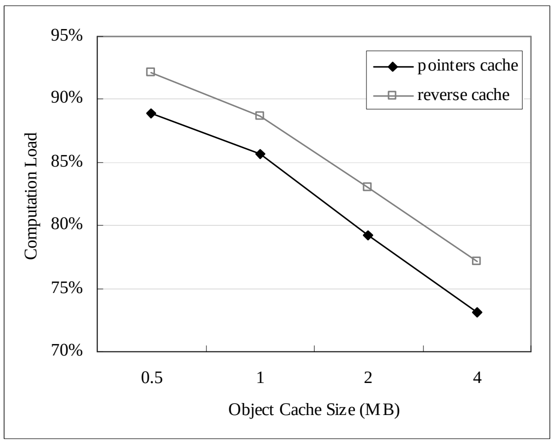
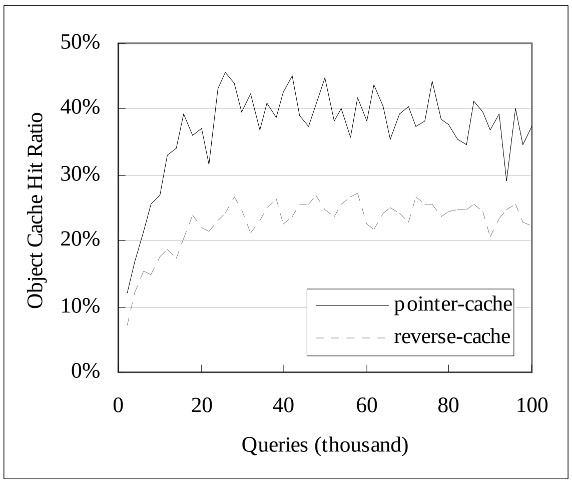


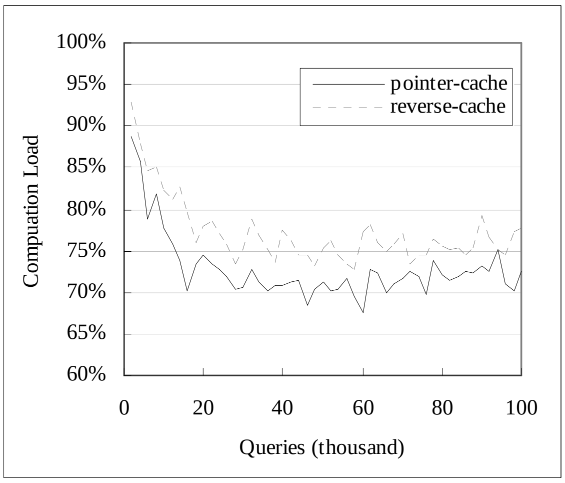
Figure 7.20 Object Cache Hit Ratio

Figure 7.21 Computation Load

Figure 7.22 (a) and (b) respectively present the object cache hit ratio and the system computation load against the number of queries. At the beginning, both methods achieve similar performance. With more queries being issued, the pointers cache method outperforms the reverse cache method. This is because after some amounts of queries are issued, the system adopting the pointers cache method has propagated sufficient numbers of pointers in the network. Therefore, the reuse rate of the response objects is improved greatly. In addition, comparing Figure 7.22 (b) and Figure 7.16, we see that the computation load reduction due to using the partial replication methods is much lower than using the pointers cache method. This suggests even if the partial replication method is applied, the proposed pointers cache method can still contribute to the major reduction of computation load.



(a) Object Cache Hit Ratio



(b) Computation Load

Figure 7.22 System Performance against the Number of Queries

We also measured the system's content delivery efficiency, such as the average query hops and the content retrieval distance. The results are as follow.

Figure 7.23 presents the average query hops. As explained, the pointers cache method may introduce an additional hop to forward the query to the pointed node. This fact is shown in the figure that, at the beginning of simulation, the number of query hops for the pointers cache method is slightly higher than that of the reverse cache method. However, after 20,000 queries are issued, the number of query hops for the pointers cache method becomes lower than that of the reverse cache method. This is because, in the rear phase of simulation, large numbers of pointers are propagated, so that many queries can be served before they reach a service replica. At the result, the pointers cache method gets less query hops than the reverse cache method.

Figure 7.24 shows the similar trend on the content retrieval distance. At the beginning of simulation, the retrieval distance for the pointers cache method is slightly larger than that of the reverse cache method. This shows that the pointers cache method would not prolong the retrieval distance significantly. After 20,000 queries are issued, the retrieval distance for the pointers cache method becomes less than that of the reverse cache method, because many queries can be served from the nearby object replicas before they reach the service replicas.

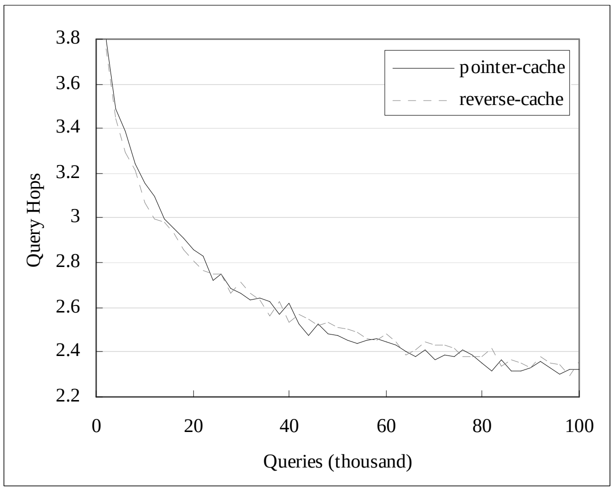


Figure 7.23 Query Hops

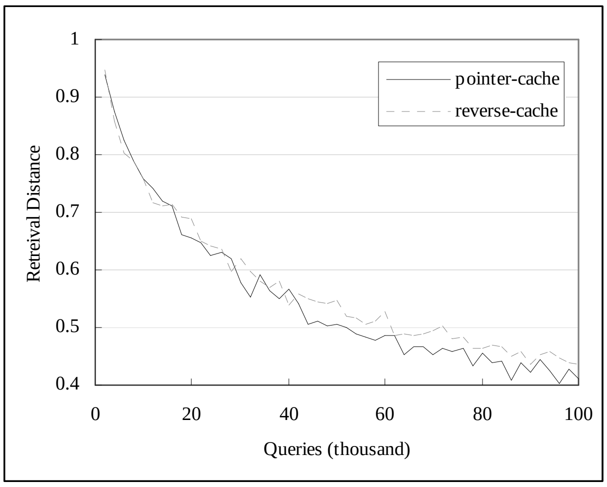


Figure 7.24 Retrieval Distance

The following part of simulation looks into the pointers cache method for more details. When a query is received the node searches the reusable pointers in its pointers cache. There are three search possibilities: cache miss, cache stale, and cache hit. If the pointers cache does not have the matching pointer, we call this situation as “cache miss”. However, if the node finds the matching pointers, this situation is called “cache found”. But, the found pointers could be useful or useless, depending on the probing results. The found pointer may be stale and we call this situation “cache stale”. If the found pointer carries the up-to-date information, this situation is called “cache hit”. For the pointers cache method, the ratio of cache-stale and cache-hit reflects the quality of the stored pointers. Figure 7.25 displays the measured percentages of cache-stale and cache-hit in the cache-found situation. With larger response cache being applied, the pointers cache’s hit ratio will improve, and accordingly the cache-stale ratio will drop. It is because with large response cache, more response objects can be stored, so that the pointers are more likely to point to the valid object replicas. Thus, enhancing the size of response cache can improve the validity of the stored pointers.

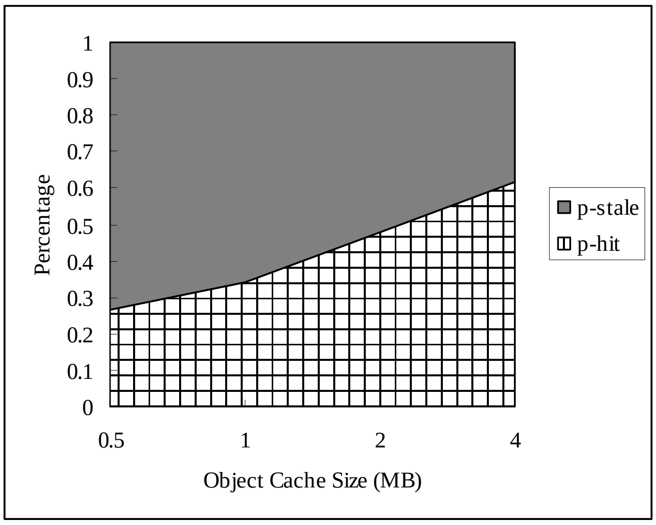


Figure 7.25 Percentages of Cache-Stale and Cache-Hit

Figure 7.26 presents the number of probe messages needed to find a valid pointer, where curve “probe-average” denotes the number of probe messages averaged over the total number of queries, while curve “probe-found” denotes the number of probe messages averaged over the number of the cache found queries. The values of “probe-found” are around 1.1, and it means when a node finds the matching pointers it needs in average 1.1 probe messages. This low value is because our pointers cache always probes from the newest pointer, and such node is much likely still caching the announced object. Thus, in many situations, we can find the valid pointer in the first probe. Meanwhile, the values of “probe-average” are around 0.6, and it means for every query the system only needs to fire around 0.6 probe messages. The results indicate the pointers cache method causes negligible numbers of probe messages in the network.

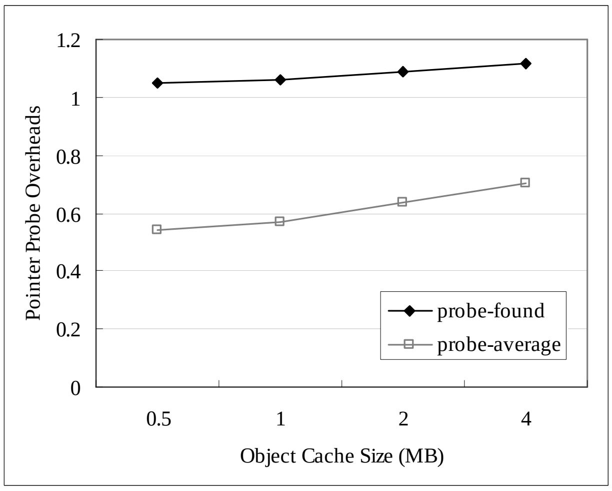
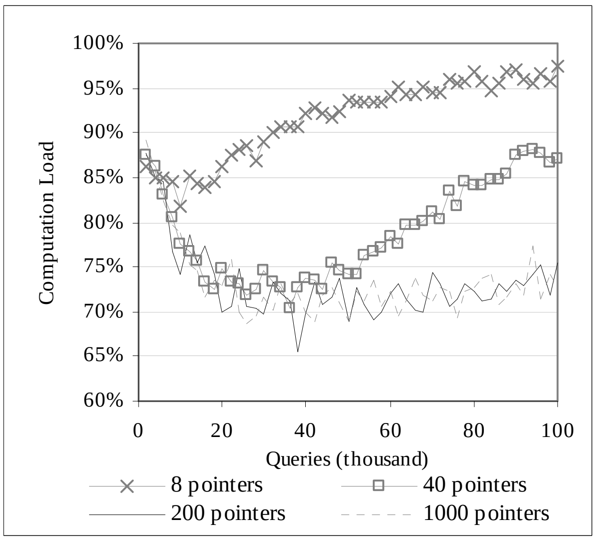
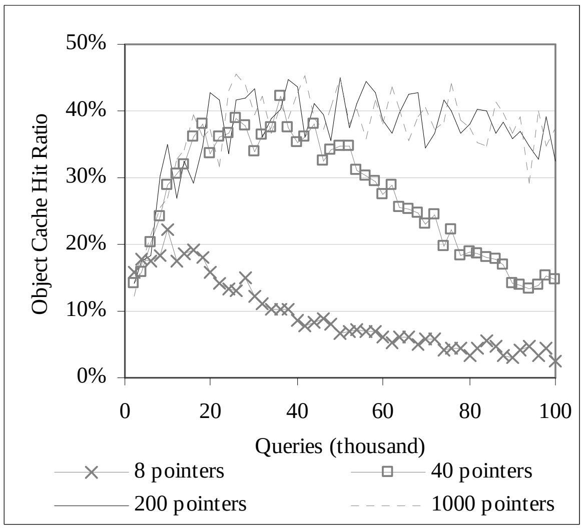


Figure 7.26 Pointer Probe Overheads

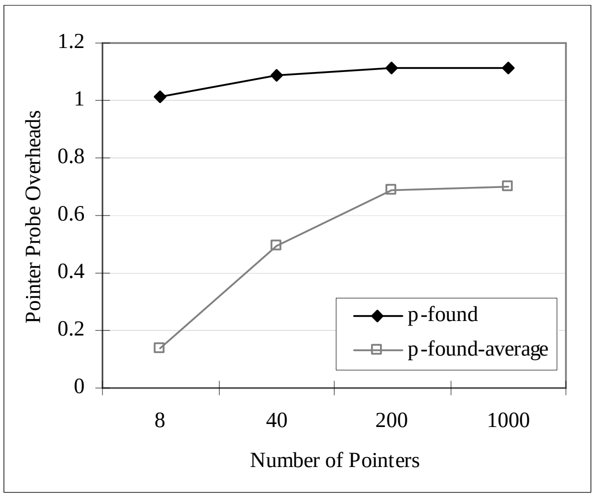
Finally, Figure 7.27 shows the system performance where we varied the number of pointers kept by each node from 8 to 1000. Figures (a) and (b) show that keeping more pointers is helpful to reduce the computation load and improve the hit ratio of the response cache. When the number of pointers is less than 100, the system perform improves at the beginning but drops quickly with more queries being issued. This is because, at the beginning of simulation, both the pointers caches and the response caches are not filled yet. Thus, the first few announced pointers are all pointing to the valid object replicas, so that the system computation load reduces sharply due to the usage of these valid pointers. However, with more queried being issued, more response caches are saturated and the old response objects are replaced by the recently downloaded objects. In this situation, if the capacity of the pointers cache is small, the node cannot get sufficient number of pointers to track the object replicas, and those few pointers kept are likely pointing to the staled replicas. As the result, the cache hit ratio drops and the computation load increases accordingly. However, if each node can maintain larger numbers of pointers, the system can efficiently track the object replicas and be more tolerant to the staled replicas. Since the pointer is a piece of soft-state information, maintaining over 1,000 pointers should not cause much storage overhead on the node. Figure (c) shows that keeping more pointers would not cause the number of probe messages to increase considerably. Hence, the results show keeping more pointers on each node is a feasible way to improve the system performance.



(a) Computation Load



(b) Object Cache Hit Ratio



(c) Probe Overhead

Figure 7.27 Pointers Cache Performance for Different Pointer Numbers

In summary, the simulation results show the pointers cache method is effective to reuse the response objects cached by the distributed nodes. This method is more efficient than the reverse cache method because it makes use of the shared storage space on different nodes whereas the reverse cache only uses the local storage space. Furthermore, the results indicate that (i) maintaining large numbers of pointers can achieve higher system performance without enhancing the probing overhead considerably, and (ii) applying large response cache can improve the validity of the stored pointers.

7.5 Summary

This chapter explains the detailed structure for our AN.P2P system. Three mechanisms are proposed to achieve high system performance. Firstly, we propose to use the Pastry routing information to perform selective service replication. The simulation results indicate such selective replication method can reduce the replication cost significantly and increase the system performance, such as the replica-hit ratio and the content retrieval distance. Secondly, we propose to replicate partial service according to the service's utility and the relevant replication overhead. The simulation results show such fine-grained replication strategy can further improve the system performance and make the system perform particularly well with small caches applied. Finally, we propose the pointers cache method to avoid repeated service execution. The simulation results show our method can leverage the shared storage space on different nodes and reduce the overall computation load considerably, but generating negligible overhead.

CHAPTER 8

CONCLUSIONS

8.1 Contributions

This thesis reports our study on the Application Networking framework for pervasive content delivery. The general framework is firstly proposed, followed by in-depth study on two concrete systems, App.Net and AN.P2P, that apply the framework into the Web and P2P content delivery contexts. The main contributions of our research work are summarized as follow.

(1) Application Networking Framework

Observing the common drawback of existing content delivery systems, we propose the Application Networking framework to integrate the content delivery and content transformation processes as the unified service delivery process. Due to this change, our framework can achieve some advantages that are not provided by the conventional frameworks. These include preserving the content semantics, improving the system scalability on adopting new applications, and achieving efficient content delivery through fine-grained service deployment.

Our framework uses workflow to organize the content transformation operations as a well-defined service. The workflow is a loosely coupled structure, which enables the framework to deliver an intermediate response object with the partial service. It generalizes the traditional content delivery methodologies that are based on either the original content or the final content presentation. More importantly, this flexibility allows the system to achieve efficient content delivery by sending different responses in the most beneficial ways.

Our framework also proposes a metadata-based method to enable the general purposed as well as service-oriented content reuse, which is not provided by the conventional type-oriented reuse methods.

(2) App.Net System

Our research proposes the App.Net system, which applies the Application Networking framework in the Web context. The system extends the HTTP 1.1 protocol and provides new caching mechanism to enable caching and reusing different versions of response on the Web proxy. We also propose performance models for both dynamic and static content services. Based on them, the service placement algorithm is provided to obtain the optimal service placement with the minimum server-to-proxy transmission cost.

The efficacy of our system is verified through a series of simulations. The first group of simulations divides the existing Web applications into several categories, and chooses a representative from each category to test. The results show our system can gain considerable performance improvement for a wide range of applications. The second group of simulations measures the effectiveness of our placement algorithms based on a sample workflow. The results show the App.Net system can resolve the optimal service placement, and it makes our system to outperform the conventional server-based and proxy-based solutions.

(3) AN.P2P System

We also extend our framework to the P2P network, and the research is explored in two phases: the general AN.P2P methodology and a concrete AN.P2P system.

The AN.P2P methodology explains how to apply service replication in the Pastry network and shows the advantage of this method over the traditional system that replicates the response objects. A mathematical model is built for the Pastry network, and the quantitative analysis indicates, in the heterogeneous query environment, service replication is more beneficial than the object replication in term of reducing the overall query hops number. More advantages of our methodology are explored in the simulation, such as reducing the retrieval distance, smoothing the replication traffic, and being tolerant to the node failure.

The second phase of research proposes a detailed AN.P2P system, which uses dedicated mechanisms to implement cost effective service replication and content reuse: (i) the selective replication method can replicate services to nodes with high reuse rate; (ii) the partial replication method enables fine-grained service replication according to the utility and delivery overhead of different service segments; and (iii) the pointers cache method enables efficient content reuse among distributed nodes, but incurring negligible overhead. The simulation results show that the usage of these mechanisms further pushes up the system performance and makes AN.P2P an efficient P2P platform for pervasive content delivery.

8.2 Future Work

This research focuses on presenting the efficacy and efficiency of the Application Networking framework. However, there are many other topics to be further explored, such as: (i) the metadata and specification languages, (ii) the system security, (iii) further performance study, and (iv) the content grid architecture.

(1) Metadata and Specification Languages

Metadata and the relevant specification languages are the essential approaches to regulate the content reuse, service delivery, and content transformation operations in the framework. Our current study leverages some existing specification languages. However, there would be more work to be done to provide the full-featured specifications.

The p-language [BR03], used for describing the configuration rules, should be extended to fit more requirements of the framework. Current p-language provides the basic vocabulary to represent the system entities and their relevant properties. This vocabulary should be extended to a wider range, so that the framework can identify more aspects about the content's properties, the client's requirements, and the other network and system conditions. Moreover, the rule language should be able to define multi-choice operations, each with a specific priority, so that the processing nodes can choose an appropriate option from them. In addition, the ruling language should provide utilities to detect and handle the conflicts and errors in the rules.

It is worthy to provide the well-defined metadata to describe the content's properties and the client's requirements. The metadata vocabulary should be extensible so that new properties and requirements can be added. In addition, it is important to construct a mapping mechanism between the client's requirements and the content properties, so that the system can perform correct content reuse and transformation.

It is also worthy to provide the metadata for the application. It should describe the application's functionality and the relevant features. The functionality metadata helps the node to perform correct application searching and reuse. In addition, other relevant specifications are wanted to describe the application's requirements to the system, such as the peak execution load and the storage size. Such information facilitates the network node to manage the system resource efficiently.

Finally, but importantly, it is worthy to improve the workflow structure to include some complex logics, such as looping. Present workflow intends to put such complicated logics inside of the application implementation, whereas moving them into the workflow would make the content manipulation and service placement more flexible.

(2) System Security

In order to achieve a secure framework, we need to study the security issues related to the content integrity, the authentication for the execution nodes and the applications, and the protection of client privacy.

It is worthy to protect the content integrity when the content is transformed by the intermediate nodes. Although the workflow specifies the wanted operations, the framework needs to guarantee that all network nodes do follow those directives so that the integrity of the final content can be maintained. Moreover, it is necessary to establish the authentication between the origin site and the intermediate nodes. Based on such authenticity, the framework can make sure the content services are received from or delivered to the trusted parties. In addition, it is important to consider the security issues related to the dynamic application execution, such as how many local resources can be accessed and which third-party nodes can be contacted by the downloaded applications. Finally, it is also worthy to protect the client's privacy especially when a downloaded application is to access the client's personal profile. Our future framework should include these security solutions.

(3) Further Performance Study

The present App.Net and AN.P2P studies treat the network transmission cost as the major performance metric. However, other factors like the computation load are also important. It would be significant to consider the computation factor in the future study. Relevant load balancing and off-loading mechanisms should be provided. Furthermore, the system should be able to trade off between the network transmission cost and the system computation load for better quality of service and user experience.

It is also worthy to strengthen the service placement mechanism by enabling to deliver the service to a cluster of nodes that work collaboratively. In specific, the performance study would address how to cluster the network nodes dynamically, how to distribute the service segments within the cluster, how to propagate the request or P2P search to the correct cluster, and how to maintain the robust service that is tolerable to node failure in the cluster.

(4) Content Grid Infrastructure

We are exploring a new infrastructure to provide the unified Application Networking mechanism over dissimilar content delivery systems, including Web proxy, CDN and P2P. The new infrastructure is named as the Content Grid (CG), which is aimed to leverage the respective advantages of the composing sub-systems to perform content delivery and service placement.

There are several key challenges in the CG system. First, we need to define a set of full-featured system protocols to implement the unified content query, request distribution and service placement mechanisms over the dissimilar sub-systems. Secondly, a service subscription protocol is also needed, allowing the network nodes to join the grid and use its shared resource. Thirdly, we need to find out how to construct the grid architecture to make it compatible with the contemporary Service Oriented Architecture (SOA). In addition, the load balancing and service placement algorithms are to be built in the system. We have done some studies on the content grid, and the preliminary results are reported in our papers at SCC’06 and SKG’06. It is of our great research interest to strengthen this study in the future.

Reference

[AAB03] L. Alimal, S. El-Ansary, P. Brand,S. Haridi, “DKS(N, k, f): a family of low communication, scalable and fault-tolerant infrastructures for P2P applications”, Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003.

[Ajax] Asynchronous JavaScript and XML (Ajax), [online], http://en.wikipedia.org/wiki/Ajax_(programming)

[Aka] Akami, [online], http://www.akami.com.

[AR98] A. Arrarwal, M. Rabinovich, “Performance of Dynamic Replication Schemes for an Internet Hosting Service”, Technical Report, AT&T Labs, October 1998.

[AS04] M. Amnefelt, J. Svenningsson, “Keso - A Scalable, Reliable and Secure Read/write Peer-to-peer File System”, Master Thesis, May 2004.

[B02] V. Berstis, “Fundamentals of Grid Computing”, http://www.globus.org.

[Bam] Bamboo-DHT Website, http://bamboo-dht.org/

[BBC02] A. Barbir, E. Burger, R. Chen, “OPES Use Cases and Deployment Scenarios”, http://www.ietf.org/internet-drafts/draft-ietf-opes-scenarios-01.txt, Aug. 2002.

[BBG00] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman, D. Zukowski, “Challenges: An Application Model for Pervasive Computing”, Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking, 2000.

[BCF99] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web Caching and Zipf-like Distributions: Evidence and Implications, INFOCOM,1999.

[BCH04] A. Barbir, R. Chen, M. Hofmann, etc., “An Architecture for Open Pluggable Edge Services”, www.ietf.org/ internet-drafts/draft-ietf-opes-architecture-04.txt.

[BCZ97] S. Bhattacharjee, K. Calvert and E. Zegura, “An Architecture for Active Networking”, Proceedings of High Performance Networking (HPN'97), White Plains, NY, April 1997.

[BH98] P. Bhagwat, R. Han, R. etc. “Dynamic Adaptation in an Image Transcoding Proxy for Mobile Web Browsing”, Proceedings of IEEE Personal Communication, 1998.

[BH01] A. Beck, M. Hofmann, “IRML: A Rule Specification Language for Intermediary Services”, http://standards.nortelnetworks.com/opes/non-wg-doc/draft-beck-opes-irml-02.txt, 2001.

[BitC] “Bit Comet”, http://www.bitcomet.com/

[BitS] “Bit Spirit”, http://www.167bt.com/intl/

[BJA98] H. Bharadvaj, A. Joshi, S. Auephanwiriyahul, “An Active Transcoding Proxy to Support Mobile Web Access”, Proceedings of IEEE Symposium on Reliable Distributed Systems”, 1998.

[BK01] S F Bush, A B Kulkarni. Active Networks and Active Network Management – A Proactive Management Framework. Kluwer Academic /Plenum Publishers. 2002.

[BO00] G. Barish, K. Obraczka, World Wide Web Caching: Trends and Techniques, IEEE Comm Magazine. Vol.38 Iss.5 May 2000.

[BR03] A. Beck, A. Rousskov, “P: Message Processing Language”, http://tools.ietf.org/html/draft-ietf-opes-rules-p-02, 2003.

[Bpel] “Business Process Execution Language for Web Services version 1.1”, http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[BPH04] A. Butt, S. Patro, Y. C. Hu, “On the Equivalence of Forward and Reverse Query Caching in Peer-to-Peer Overlay Networks”, Proceedings of the 9th International Workshop on Web Content Caching and Distribution, 2004.

[BPT03] F. Bagci, J. Petzold, W. Trumler, T. Ungerer, “Ubiquitous Mobile Agent System in a P2P Network”, Proceedings of the UbiSys Workshop at the Fifth Annual Conference on Ubiquitous Computing, 2003.

[BS01] S. Buchholz, A. Schill, “Web Caching in a Pervasive Computing World”, Proceedings of the 7th ACM/IEEE International Conference on Mobile Computing and Networking, 2001.

[BT] Bit Torrent, [online], http://bittorrent.com.

[C99] P. Chen, “On the Study of Watermarking Application in WWW – Modeling, Performance Analysis and Applications of Digital Image

Watermarking Systems", Master Thesis of Dept of EE, National Tsing Hua Unv., 1999.

[Carp97] "Cache Array Routing Protocol (CARP) and Microsoft Proxy Server 2.0", http:// www.microsoft.com/technet/archive/proxy/prxcarp.mspx

[CB96] M. E. Crovella, A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes", Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems, 1996.

[CC02] C-H. Chi, Y. Cao, "Pervasive Web Content Delivery with Efficient Data Reuse", Proceedings of 7th International Workshop on Web Content Caching and Distribution, 2002.

[Ccpp] Composite Capabilities/Preference Profiles, http://www.w3.org/Mobile/CCPP/.

[CDK03] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron, A. Singh, "SplitStream: High-bandwidth Multicast in a Cooperative Environment", Proceedings of Symposium of Operating Systems Principles, New York, October 2003.

[CP05] C-H. Chi, H. N. Palit, "Modulation for Scalable Multimedia Content Delivery", Proceedings of the 6th International Conference on Web-Age Information Management, 2005.

[CI97] P. Cao, S. Irani, "Cost-Aware WWW Proxy Caching Algorithms", Proceedings of USENIX Symposium on Internet Technologies and Systems, 1997.

[CIW00] J. Challenger, A. Iyengar, K. Witting, "A Publishing System for Efficiently Creating Dynamic Web Content", Proceedings of INFOCOM, 2000.

[CLR90] T. H. Cormen, C. E. Leiserson, R. L. Rivest, "Introduction to Algorithms", MIT Press, 1990.

[CMI99] S. Chandrasekaran, S. Madden, M. Ionescu, "Ninja Paths: An Architecture for Composing Services Over Wide Area Networks", http://ninja.cs.berkeley.edu/dist/papers/path.ps.gz.

[CMZ03] Y. Chen, W. Ma, H. Zhang, "Detecting Web Page Structure for Adaptive Viewing on Small Form Factor Devices", Proceedings of 12th International World Wide Web Conference, May 2003.

[CN03] Y. Cui, K. Nahrsdedt, “Layered Peer-to-Peer Streaming”, Proceedings of the 13th ACM International Workshop on Network and Operating Systems Supports for Digital Audio and Video (NOSSDAV’03), 2003.

[Com] Component Object Model Techniques, http://www.microsoft.com/com/default.mspx

[Corba] OMG’s CORBA website, http://www.corba.org/.

[CRB03] Y. Chawathe, S. Ratnasamy, L. Breslau, S. Shenker, "Making Gnutella-like P2P Systems Scalable", In Proceedings of the ACM SIGCOMM, August 2003.

[CRK05] F. Chen, T. Repantis, V. Kalogeraki, “Coordinated Media Streaming and Transcoding in Peer-to-Peer Systems”, Proceedings of 19th International Parallel and Distributed Processing Symposium, 2005.

[CY03] R. Y. Chen, B. Yeager, “Java Mobile Agents on Project JXTA Peer-to-Peer Platform”, Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS’03), 2003.

[CYH00] V. Cardellini, P. S. Yu, Y. Huang, “Collaborative Proxy System for Distributed Web Content Transcoding”, Proceedings of the 9th International Conference on Information and Knowledge Management. p.520-527, Nov, 2000.

[CS02] E. Cohen, S. Shenker, “Replication Strategies in Unstructured Peer-to-peer Networks”, Proceedings of SIGCOMM Conference, 2002.

[CZB98] P. Cao, J. Zhang, K. Beach, “Active Cache: Caching Dynamic Contents on the Web”, Proceedings of IFIP International Conf. on Distributed Systems Platforms and Open Distributed Processing, Sept. 1998.

[CZS03] J. Chen, B. Zhou, J. Shi, H-J. Zhang, F. Qiu, “Function-based Object Model towards Website Adaptation”, Proceedings of 10th International WWW Conference, 2001.

[H06] P. Howard, “SOA and Information Services”, Bloor Research Report. 2006.

[DB04] G. Ding, B. Bhargava, “Peer-to-Peer File-Sharing over Mobile Ad Hoc Networks”, Proceedings of 2nd IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004.

[DCG01] R. P. Doyle, J. S. Chase, S. Gadde, A. M. Vahdat, “The Trickle-Down Effect: Web Caching and Server Request Distribution”, Proceedings of 6th

International Workshop on Web Caching and Content Distribution, USA, 2001.

[DDT02] A. Datta, K. Dutta, H. Thomas, etc., “A Proxy-based Approach for Dynamic Content Acceleration on the WWW”, Proceedings Fourth IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, 2002.

[DHR97] F. Douglis, A. Haro, M. Rabinovich, “HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching”, Proceedings of USENIX Symposium on Internetworking Technologies and Systems, 1997.

[Dig] Digital Island, http://www.digitalisland.co.nz/

[Dik04] M. Dikaiakos, “Intermediary Infrastructures for the WWW”, Proceedings of Computer Networks: The International Journal of Computer and Telecommunications Networking, Volume 45, Issue 4, July, 2004.

[DKK01] F. Dabek, M. F. Kaashoek, D. Karger, “Wide-area Cooperative Storage with CFS”, Proceedings of Symposium of Operation System Principles, 2001.

[Dom] W3C Document Object Model, http://www.w3.org/dom.

[Dou04] B. Dournaee, “Introduction to ebXML”, http://dev2dev.bea.com/pub/a/2004/12/ebXML.html

[DZD03] F. Dabek, B. Zhao, P. Druschel, I. Stoica, “Towards a Common API for Structured Peer-to-Peer Overlays”, Proceedings of 2nd International Workshop on Peer-to-Peer Systems, 2003.

[EC03] J. Elson, A. Cerpa, “Internet Content Adaptation Protocol (ICAP)”, http://www.i-cap.org/spec/rfc3507.txt, 2003.

[Erl05] T. Erl, “Service-Oriented Architecture: Concepts, Technology, and Design”, Prentice Hall PTR, Aug 2005.

[Esi] Edge Side Includes, http://www.esi.org.

[FB96] A. Fox, E. A. Brewser, “Reducing WWW Latency and Bandwidth Requirements by Real-time Distillations”, Proceeding of 5th International WWW Conference, May, 1996.

[FCA00] L. Fan, P. Cao, J. Almeida, A. Z. Broder, “Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol”, Proceedings of IEEE/ACM Transactions on Networking, Vol. 8, No. 3, June 2000.

[Free] The Free Network Project – Wiring the Internet, [online], http://freenet.sourceforge.net/

[FG98] M. Fry, A. Ghosh, “Application Level Active Networking”, Proceedings of Computer Networks, 1998.

[FGC98] A. Fox, S. D. Gribble, Y. Chawathe, “Adapting to Network and Client Variation Using Active Proxies: Lesson and Perspectives”, Proceedings in a special issue of IEEE Personal Communication on Adaptation, 1998.

[FKN02] I. Foster, C. Kesselman, J. M. Nick, S. Tuecke, “The Physiology of the Grid - an Open Grid Services Architecture for Distributed System Integration”, http://www.globus.org/research/papers/ogsa.pdf.

[FSA01] X. Fu, W-S Shi, M. Allen, V. Karamcheti, “CANS: Composable, Adaptive Network Services Infrastructure”, Proceedings of USENIX on Internet, 2001.

[FSK01] X. Fu, W-S. Shi, V. Karamcheti, “Automatic Deployment of Transcoding Components for Ubiquitous, Network-Aware Access to Internet Services”, Tech Report, 2001.

[GDH01] R. Grimm, J. Bavis, B. Hendrickson, E. Lemar, A. Beth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, D. Wetherall, “System Directions for Pervasive Computing”, Proceedings of the 8th Workshop on Hot Topics in Operating Systems, Germany, May 2001.

[GDN03] L. Gao, M. Dahlin, A. Nayate, J. Zheng, “Application Specific Data Replication for Edge Services”, Proceedings of International WWW Conference, 2003.

[GDS03] K. P. Gummad, R.J. Dunn, S. Saroiu, “Measurement, Modeling, and Analysis of a Peer-to-peer File-Sharing Workload”, Proceedings of SOSP'03, 2003.

[GFTP] GridFTP, http://www.globus.org/grid_software/data/gridftp.php.

[GKB05] Shen Tat Goh, P. Kalnis, S. Bakiras, Kian-Lee Tan, “Real Datasets for File-sharing Peer-to-peer Systems”, Proceedings of the International Conference on Database Systems for Advanced Applications, 2005.

[Glo] Globus Toolkit, www.globus.org.

[Gnu] Gnutella, [online], www.gnutella.com.

[GNY04] X. Gu, K. Nahrstedt, B. Yu, “SpiderNet: In Integrated Peer-to-Peer Service Composition Framework”, Proceedings of the IEEE International Symposium on High Performance Distributed Computing, 2004.

[GR05] C. Gkantsidis, P. Rodriguez, “Network Coding for Large Scale Content Distribution”, Proceedings of IEEE/INFOCOM 2005, Miami. March 2005.

[GSB04] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, P. Kelenher, “Adaptive Replication in Peer-to-Peer Systems”, Proceedings of 24th Inter. Conf. on Distributed Computing System, 2004.

[GWeb] “Gnutella Web Caching System”, [online], http://www.gnucleus.com/gwebcache/newgwc.html.

[HGV02] T. Horozov, A. Grama, V. Vasudevan, S. Landis, “MOBY - A Mobile Peer-to-Peer Service and Data Network”, Proceedings of International Conference on Parallel Processing, 2002.

[HIR01] K. Henricksen, J. Indulska, A. Rakotonirainy, “Infrastructure for Pervasive Computing: Challenges”, Proceedings of Workshop on Pervasive Computing and Information Logistics, 2001.

[HKO99] M. Hori, G, Kondoh, K. Ono, S. Hirose, S. Singhal, “Annotation-Based Web Content Transcoding”, Proceedings of 9th International World Wide Web Conference, 1999.

[HM04] R. Hanrahan, R. Merrick, “Authoring Techniques for Device Independence”, http://www.w3.org.

[Http] HTTP/1.1, http://www.w3.org/protoc- ols/rfc2616/.

[Hul02] S. Hull, “Content Delivery Networks : Web Switching For Security, Availability, and Speed”, Berkeley, Calif. ; London : Osborne/McGraw-Hill, 2002.

[HW04] R. Hsiao, S-D Wang. “Jelly: A Dynamic Hierarchical P2P Overlay Network with Load Balance and Locality”, Proceedings of the 24th International Conference on Distributed Computing Systems Workshops (ICDCSW'04), 2004.

[IAUS03] T. Iwata, T. Abe, K. Ueda, H. Sunaga, “A DRM System Suitable for P2P Content Delivery and the Study on its Implementation”, Proceeding of the 9th Asia-Pacific Conference on Communication, Vol.2, 21-24, pp.806-811, 2003.

[Icap] ICAP, http://www.i-cap.org/home.html.

[IC97] A. Iyengar, J. Challenger, "Improving Web Server Performance by Caching Dynamic Data", Proceedings of the USENIX Symposium on Internet Technologies and Systems, 1997.

[IHA02] A. Ivan, V. Karamcheti, "Partitionable Services: A Framework for Seamless Adapting Distributed Application to Heterogeneous Environments", Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing, 2002.

[IRD02] S. Iyer, A. Rowstron, P. Drusche, "Squirrel: A Decentralized Peer-to-peer Web Cache", Proceedings of the 21st ACM Symposium on Principles of Distributed Computing, July 2002.

[Jigsaw] Jigsaw - W3C's Server, http://www.w3.org/Jigsaw/

[JJK01] S. Jamin, C. Jin, A. R. Kurc, D. Raz, Y. Shavitt, "Constrained Mirror Placement on the Internet", Proceedings of IEEE INFOCOM, 2001.

[JLH01] X. Jia, D. Li, X. Hu, D. Du, "Optimal Placement of Web Proxies for Replicated Web Servers in the Internet", Proceedings of the Computer Journal, Vol. 44, No. 5, 2001.

[Kaz] Kazaa, www.kazaa.com.

[KC04] W. Ku, C-H. Chi, "Survey on the technological aspects of Digital Rights Management", Proceeding of the 7th Information Security Conference, 2004.

[KLM02] B. Knutsson, H. Lu, J. Mogul, "Architecture and pragmatics of Server-directed Transcoding", Proceedings of 7th International workshop on Web content caching and distribution, 2002.

[KLM03] B. Knutsson, H. Lu, J. Mogul, "Architecture and Performance of Server-directed Transcoding", Proceedings of ACM Transaction on Internet Tech., Vol.3 Iss.4, 2003, pp. 392-424.

[KRR01] J. Kangasharju, J. Roberts, K. W. Ross, Object Replication Strategies in Content Distribution Networks, Proceedings of the Workshop of Web Content Caching and Distribution, 2001.

[KRS00] P. Krishnan, D. Raz, Y. Shavitt, "The Cache Location Problem", Proceedings of IEEE/ACM Transaction on Networking, 2000.

[KRT02] J. Kangasharju, K. W. Ross, D. A. Turner, "Adaptive Content Management in Structured P2P Communities", Proceedings of 21st ACM Symposium on Principles of Distributed Computing, 2002.

[KTor] KTorrent, [online], http://ktorrent.org/

[KW02] B. Krishnamurthy, C. E. Wills, “Improving Web Performance by Client Characterization Driven Server Adaptation”, Proceedings of the World Wide Web Conference, 2002.

[KWV05] N. Kotilainen, M. Weber, M. Vapa, J. Vuori, “Mobile Chedar - A Peer-to-Peer Middleware for Mobile Devices”, Proceedings of the 4th Annual IEEE International Conference on Pervasive Computing and Communications, 2005.

[KWZ03] B. Krishnamurthy, C. E. Wills, Y. Zhang, K. Vishwanath, “Design, Implementation, and Evaluation of a Client Characterization Driven Web Server”, Proceedings of the World Wide Web Conference, 2003.

[LCC02] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, “Search and Replication in Unstructured Peer-to-peer Networks”, Proceeding of the 16th Int. Conf. on Supercomputing, 2002.

[LGI99] B. Li, M. J. Golin, G. F. Italiano, X. Deng, K. Sohraby, “On the Optimal Placement of Web Proxies in the Internet”, Proceedings of IEEE INFOCOM, 1999.

[Ley01] F. Leymann, “Web Services Flow Language (WSFL 1.0), http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[LHP04] W-S Li, W-P. Hsiung, K. Hino, K. S. Candan, D. Agrawal, “Challenges and Practices in Deploying Web Acceleration Solutions for Distributed Enterprise Systems”, Proceedings of International WWW conference, 2004.

[LHS04] B. T. Loo, R. Huebsch, I. Stoica, J. M. Hellerstein, “The Case for a Hybrid P2P Search Infrastructure”, Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS’04), 2004.

[LM01] Q. Li, B. Moon, “Distributed Cooperative Apache Web Server”, Proceedings of 10th International World Wide Web Conference, 2001.

[LYR02] J. Li, M. Yarvis, P. Reiher, “Securing Distributed Adaptation”, Proceeding of Computer Networks, Vol. 38, pp. 347-371, 2002.

[MC00] M. Metter, R. Colomb, “WAP Enabling Existing HTML Application”, Proceedings of the 1st Australasian User Interface Conference, 2000.

[MDS03] V. Mastoli, V. Desai, W-S. Shi, “SEE: A Service Execution Environment for Edge Services”, Proceedings of the 3rd IEEE Workshop on Internet Applications, p. 61, 2003.

[MF99] G. McGraw, E.W. Felten, “Securing Java: Getting Down to Business with Mobile code”, Wiley, 1999.

[Mime] MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies, http://www.faqs.org/rfcs/rfc1521.html.

[MSB01] W. Ma, B. Shen, J. Brassil, “Content Services Network: The Architecture and Protocols”, Proceedings of 6th International Workshop on Web Caching and Content Distribution, June 2001.

[MSR01] A. Maheshwari, A. Sharma, K. Ramamritham, P. Shenoy, “TranSquid: Transcoding and Caching Proxy for Heterogenous E-Commerce Environments”, Proceedings of 12th IEEE Workshop on Research Issues in Data Engineering, 2001.

[Nap] Napster, http://www.napster.com.

[O01] H. K. Orman, “Data Integrity for Mildly Active Content”, http://doi.ieeecomputersociety.org/10.1109/AMS.2001.993722

[Ocp] OPES Callout Protocol, http://www.ietf.org/rfc/rfc4037.txt.

[Opes] Open Pluggable Edge Services, http:// www.ietf-opes.org.

[Pastry] Free-Pastry platform, http://research.microsoft.com/~antr/Pastry/

[PC05] H. N. Palit, C-H. Chi, “Modulation for Scalable Multimedia Content Delivery”, Proceedings of the 6th International Conference for Advances in Web-Age Information Management, China, 2005.

[PCL06] H. N. Palit, C-H. Chi, L. Liu, “Proxy-Based Pervasive Multimedia Content Delivery”, Proceedings of 30th Annual International Computer Software and Applications Conference, USA, 2006.

[PCP03] V. S. Pai, A. L. Cox, V. S. Pai, W. Zwaenepoel, “A Flexible and Efficient Application Programming Interface (API) for a Customization Proxy Cache”, Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems, 2003.

[PD99] C. Parris, B. Dennis, “Transformation Proxy Support for Thin-Clients”, http:// citeseer.ist.psu.edu/58739.html

[PG04] J. Patel, I. Gupta, “Overhaul: Extending HTTP to Combat Flash Crowds”, 9th International Workshop on Web Content Caching and Distribution, Beijing, 2004.

[PH97] D. Povey and J. Harrison, “A Distributed Internet Cache”, Proceedings of the 20th Australasian Computer Science Conference, February 1997.

[PS01] G. Pierre, M. v. Steen, “Globule: a Platform for Self-Replicating Web Documents”, Proceedings of the 6th International Conference on Protocols for Multimedia Systems, Oct. 2001.

[PS02] M. Papadopouli, H. Schulzrinne, “Design and Implementation of a Peer-to-Peer Data Dissemination and Prefetching Tool for Mobile Users”, Proceedings of the 1st New York Metro Area Networking Workshop, 2002.

[PS03] G. Pierre, M. v. Steen, “Design and Implementation of a User-Centered Content Distribution Network”, Proceedings of the 3rd IEEE Workshop on Internet Applications, 2003.

[PSB02] T. Phan, G. Zorpas, R. Bagrodia, “An Extensible and Scalable Content Adaptation Pipeline Architecture to Support Heterogeneous Clients”, Proceedings of the 22nd International Conference on Distributed Computing Systems, 2002.

[PST02] G. Pierre, M. v. Steen, A. S. Tanenbaum, “Dynamically Selecting Optimal Distribution Strategies for Web Documents”, Proceedings of the IEEE Transactions on Computers, Vol.51 (6): 637-751, June, 2002.

[QPV01] L. Qiu, V. N. Padmanabhan, G. M. Voelker, “On the Placement of Web Server Replicas”, Proceedings of the IEEE INFOCOM, 2001.

[Rab98] M. Rabinovich, “Issues in Web Content Replication”, http://citeseer.ist.psu.edu/rabinovich98issues.html.

[RA99] M. Rabinovich, A. Aggarwal, “RaDaR: A Scalable Architecture for A Global Web Hosting Service”, Proceedings of Computer Networks, Amsterdam, Netherlands, 1999.

[Rau99] M. S. Raunak, “A Survey of Cooperative Caching”, http://citeseer.ist.psu.edu/raunak99survey.html

[RD01] A. Rowstron, P. Druschel, “Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems”, In Proceedings of the

18th IFIP/ACM International Conference of Distributed Systems Platforms, Nov. 2001.

[RD01b] A. Rowstron, P. Druschel, “Pastry: Scalable, Decentralized Object Location and Routing for Large-scale Peer-to-peer Systems”, Proceedings of the 18th IFIP/ACM International Conference of Distributed Systems Platforms, Nov. 2001.

[RFH01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, “A Scalable Content-Addressable Network”, Proceedings of ACM SIGCOMM Conference, 2001

[RG98] A. D. Rubin, E.E. Greer, “Mobile Code Security”, Proceedings of IEEE Internet Computing, Nov. 1998.

[RGK05] 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”, Proceedings of ACM SIGCOMM 2005, August 2005.

[Rou05] A. Rousskov, “Open Pluggable Edge Services (OPES) Callout Protocol (OCP) Core”, http://www.ietf.org/rfc/rfc4037.txt

[RRR98] M. Rabinovich, I. Rabinovich, R. Rajaraman, “Dynamic Replication on the Internet Work Project No. 3116-17-7006”, AT&T Labs Research Report, 1998.

[RS98] A. Rousskov, D. Wessels, “Cache Digests”, Computer Networks and ISDN Systems, 1998

[RSB01] P. Rodriguez, C. Spanner, E. W. Biersack, “Analysis of Web Caching Architectures: Hierarchical and Distributed Caching”, Proceedings of IEEE/ACM Transactions on Networking, August 2001.

[RXA03] M. Rabinovich, Z. Xiao, A. Aggarwal, “Computing on the Edge: A Platform for Replicating Internet Applications”, Proceedings of 8th International Workshop on Web Caching and Content Distribution, Sept. 2003.

[RXD02] M. Rabinovich, Z. Xiao, F. Douglis, “Moving Edge-Side Includes to the Real Edge - the Clients”, http://www.research.att.com/~misha/otherPubs/csi.pdf.

[RV03] P. Reynolds, A. Vahdat. “Efficient Peer-to-peer Keyword Searching”, Proceedings of Middleware Conference. 2003.

[SAP05] S. Sivasubramanian, G. Alonso, G. Pierre, Maarten van Steen, “GlobeDB: Autonomic Data Replication for Web Applications”, Proceedings of the International World Wide Web Conference, 2005.

[SCK03] W-S. Shi, E. Collins, V. Karamcheti, “Modeling Object Characteristics of Dynamic Web Content”, Proceedings of Journal of Parallel and Distributed Computing (JPDC), special issue on scalable Internet services and architecture, Vol. 63, No. 10, pages 963-980, Oct.2003.

[SJS00] B. Schwartz, A.W. Jackson, W.T. Strayer, W. Zhou, R.D. Rockwell, and C. Partridge. "Smart Packets: Applying Active Networks to Network Management". ACM Transactions on Computer Systems, 18(1):67--88, 2000.

[SK01] W-S. Shi, V. Karamcheti, “CONCA: An Architecture for Consistent Nomadic Content Access”, Proceedings of Workshop on Cache, Coherence, and Consistency, 2001.

[SMK01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications”, Proceedings of the ACM SIGCOMM Conference, 2001.

[Soap] “Simple Object Access Protocol. version 1.2”, http://www.w3.org/TR/soap/.

[Soap-a] “SOAP Messages with Attachments”, http://www.w3.org/TR/SOAP-attachments.

[SPS04] S. Sivasubramanian, G. Pierre, M. v. Steen, “Replicating Web Applications On-Demand”, Proceedings of Services Computing, 2004 IEEE International Conference on (SCC'04) pp. 227-236, 2004.

[Squ] Squid Web Proxy Cache, http://www.squid-cache.org/.

[Smi] Smith, et. al. “SwitchWare: Towards a 21st Century Network Infrastructure”, http://www.cis.upenn.edu/~gunter/wip/ieee.ps.Z.

[Soap] Simple Object Access Protocol. version 1.2, http://www.w3.org/TR/soap/

[Sona] Cisco Service-Oriented Network Architecture, www.cisco.com/application/pdf/en/us/guest/netsol/ns477/c643/cdccont_0900aecd8039b324.pdf

[SP02] J. Steinberg, J. Pasquale, “A Web Middleware Architecture for Dynamic Customization of Content for Wireless Clients”, Proceedings of 11th International World Wide Web conference, 2002.

[SSM03] W-S. Shi, K. Shah, Y Mao, V. Chaudhary, “Tuxedo: A Peer-to-Peer Caching System”, Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications, June 2003.

[STR02] A. Singh, A. Trivedi, K. Ramamritham, P. Shenoy, “PTC: Proxies that Transcode and Cache in Heterogeneous Web Client Environments”, Proceedings of 3rd International Conference on Web Information System Engineering, Singapore, 2002.

[Tha01] S. Thatte, “XLANG Web Services for Business Process Design”, http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

[Tpc] Transaction Processing Performance Council, http://www.tpc.org.

[TW96] D. L. Tennenhouse, D. J. Wetherall, “Towards an Active Network Architecture”, http://tns-www.lcs.mit.edu/publications/ccr96.html

[TWJ01] M. Tsimelzon, B. Weihl, L. Jacobs, “ESI Language Specification 1.0”, www.esi.org, 2001.

[Wal01] A. Walker, “Proxylet Local Execution Environment Java Binding V0.1”, http://www.ietf-opes.org/documents/draft-walker-opes-proxylet-java-binding-01.txt, Aug. 2001.

[W3C] World Wide Web Consortium, “http://www.w3.org”.

[Wap] Wireless Application Protocol, http://www.wapforum.org/what/technical.htm.

[Wap00] Wireless Application Protocol - Wireless Markup Language Specification, www.openmobilealliance.org/release_program/docs/Browsing/V2_1-20050614-C/WAP-191-WML-20000219-a.pdf, 2000.

[WC97] D. Wessels and K Claffy, "ICP and the squid web cache." http://citeseer.nj.nec.com/wessels97icp.html

[WGT98] D. J. Wetherall, J. Guttag, D. L. Tennenhouse, “ANTS: A Tool kit for building and dynamically deploying network protocols”, Proceedings of IEEE OPENARCH'98, 1998.

[WHB03] M. Waldvogel, P. Hurley, D. Bauer, “Dynamic Replica Management in Distributed Hash Tables”, IBM Research Report, July 2003.

[Ws] Web Services, http://www.w3.org/2002/ws/.

[Wsa] Web Service Addressing (WS-Addressing), http://www.w3.org/Submission/ws-addressing/

[Wsdd] Web Service Dynamic Discovery (WS-Discovery), http://schemas.xmlsoap.org/ws/2005/04/discovery/

[Wsdl] “Web Service Description Language. Version 1.1”, http://www.w3.org/TR/wsdl

[Wsel] “Web Service Endpoint Language”, http://www.service-architecture.com/web-services/articles/web_services_endpoint_language_wsel.html

[Wsp] “Web Service Policy Framework”, http://www-128.ibm.com/developerworks/webservices/library/specification/ws-polfram/

[WVS99] A. Wolman, G. M. Voelker, N. Sharma, etc. “On the Scale and Performance of Cooperative Web Proxy Caching”, Proceedings of 17th Symposium on Operating Systems Principles, 1999.

[Uddi] “Universal Description, Discovery and Integration Specification”, http://www.uddi.org/specification.html

[Uapr] “User Agent Profiles”, http://w3development.de/rdf/uaprof_repository/.

[Uiml] “User Interface Markup Language”, www.uiml.org.

[Utor] µTorrent, [online], http://www.utorrent.com/

[Ver02] D. C. Verma. “Content distribution networks: an engineering approach”, New York : J. Wiley, 2002.

[Xslt] “XSL Transformations (XSLT) Version 1.0. W3C Recommendation”, http://www.w3.org/TR/xslt.

[YCZ03] C. Yuan, Y. Chen, Z. Zhang, “Evaluation of Edge Caching/Offloading for Dynamic Content Delivery”, Proceedings of International World Wide Web conference, 2003.

[YH01] L. Yang, M. Hofmann, “OPES Architecture for Rule Processing and Service Execution”, http://www.ietf-opes.org/documents/draft-yang-opes-rule-processing-service-execution-00.txt, Aug. 2001.

[YHZ03] C. Yuan, Z. Hua, Z. Zhang, “Proxy+: Simple Proxy Augment for Dynamic Content Processing”, Proceedings of Web Content Caching and Distribution, 2003.

[YS96] Yemini, da Silva, “Towards Programmable Networks”, Proceedings of IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, 1996.

[ZFJ97] L. Zhang, S. Floyd, and V. Jacobson. “Adaptive Web Caching”, Proceedings of the NLANR Web Cache Workshop, June 1997. http://citeseer.nj.nec.com/zhang97adaptive.html

[ZWL02] L. Zhuo, C-L. Wang, F. C. M. Lau, “Load Balancing in Distributed Web Server Systems with Partial Document Replication”, Proceedings of the International Conference on Parallel Processing, 2002.

Index

ACDN	Application Content Distribution Network
AJAX	Asynchronous JavaScript and XML
APP	Application
BPEL	Business Process Execution Language
BT	Bit Torrent
CAN	Content Addressable Network
CANS	Composable Adaptive Network Services
CC/PP	Composite Capabilities/Preferences Profile
CDN	Content Distribution Network
CR	Configuration Rule
CSI	Client Side Inclusion
CSN	Content Service Network
DHT	Distributed Hash Table
DIA	Device Independent Authoring Service
ESI	Edge Side Inclusion
HTTP	Hyper Text Transfer Protocol
LFU	Least Frequently Used
ICAP	Internet Content Adaptation Protocol
IRML	Intermediary Rule Markup Language
LISO	Large Input Small Output
LISO+D	LISO application with additional Data
LRU	Least Recently Used
OBJ	Content Object
OCP	OPES Callout Protocol
OPES	Open Pluggable Edge Service
P2P	Peer to Peer
PAST	Pastry-based distributed Storage system
PC	Personal Computer
PDA	Personal Digital Assistant
PTC	Proxy for Transcoding and Caching

RaDaR	Replicator and Distributor and Redirector based CDN System
SDT	Server-Directed Transcoding
SECP	Service Enabled Caching Proxy
SEG	Segment
SILO	Small Input Large Output
SILO+D	SILO application with additional Data
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SONA	Service Oriented Network Architecture
TSK	Task
Transquid	Transcoding Squid
TTL	Time To Live
UAProf	User Agent Profile
URI	Uniform Resource Identifier
URL	Uniform Resource Location
WF	Workflow
WS-BPEL	Web Service for Business Process Execution Language
WSFL	Web Service Flow Language
WWW	World Wide Web
XML	Extensible Markup Language
XSLT	XML Transformation