Large-Scale Simulator for Global Data Infrastructure Optimization

by

Sergio Herrero-López

M.S., Electrical Engineering and Computer Science
University of Navarra (2007)
B.S., Electrical Engineering and Computer Science
University of Navarra (2005)

Submitted to the Department of Civil and Environmental Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in the field of Information Technology

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2012

Author .....
Department of Civil and Environmental Engineering
December 12, 2011

Certified by .....
John R. Williams
Professor of Civil and Environmental Engineering
and Engineering Systems
Thesis Supervisor

Accepted by .....
Heidi Nepf
Chair, Departmental Committee for Graduate Students

Large-Scale Simulator for Global Data Infrastructure Optimization

by

Sergio Herrero-López

Submitted to the Department of Civil and Environmental Engineering
on December 12, 2011, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in the field of Information Technology

Abstract

Companies depend on information systems to control their operations. During the last decade, Information Technology (IT) infrastructures have grown in scale and complexity. Any large company runs many enterprise applications that serve data to thousands of users which, in turn, consume this information in different locations concurrently and collaboratively. The understanding by the enterprise of its own systems is often limited. No one person in the organization has a complete picture of the way in which applications share and move data files between data centers. In this dissertation an IT infrastructure simulator is developed to evaluate the performance, availability and reliability of large-scale computer systems. The goal is to provide data center operators with a tool to understand the consequences of infrastructure updates. These alterations can include the deployment of new network topologies, hardware configurations or software applications. The simulator was constructed using a multi-layered approach and was optimized for multicore scalability. The results produced by the simulator were validated against the real system of a Fortune 500 company. This work pioneers the simulation of large-scale IT infrastructures. It not only reproduces the behavior of data centers at a macroscopic scale, but allows operators to navigate down to the detail of individual elements, such as processors or network links. The combination of queueing networks representing hardware components with message sequences modeling enterprise software enabled reaching a scale and complexity not available in previous research in this area.

Thesis Supervisor: John R. Williams
Title: Professor of Civil and Environmental Engineering
and Engineering Systems

To my parents, and my beloved wife, Cristina

Acknowledgments

Professor John R. Williams for the opportunity, guidance and inspiration to pursue this career. I will always be grateful for his dedication, discussions, insights, generosity and the fact that his door was always open to me.

Dr. Abel Sanchez for his confidence in me, mentorship and help, beyond the boundaries of what it is required by MIT and making my degree an endless learning experience.

Professor Jerome Connor and Dr. George Kocur for their wise advice, invaluable guidance and assistance both as professors and thesis committee members.

The Basque Government, EPCglobal, Microsoft Research, SAP Labs, Dassault Systèmes and Ford Motor Company for providing funding and financial aid for my work throughout different stages of my research.

My Auto-ID Lab and IESL colleagues and friends, Jin Hock Ong, Fivos Constantinou, Indy Yu, Monica Sun, Tzu-Ching Horng, Christian Floerkemeier, Rahul Bhattacharyya, Edmund W. Schuster, Isaac Ehrenberg, David Holmes and Christopher Leonardi for being partners in different stages of this journey, engaging in endless stimulating conversations and making the experience enjoyable.

My friends Ruben Martín, Laura Herrero, Ana Elena Minatti, Naiara Rodriguez and Eduardo Granados because they are part of the best memories that I will take with me from this journey.

My parents and brother for their unconditional support, understanding and encouragement.

My wife because her patience and love have made this possible. She stood by my side for the good and bad moments, and for me, the merit of this thesis is as much hers as mine.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The relevance of Information Technology (IT) infrastructures in corporations has grown in consonance with the unstoppable phenomenon of globalization. Multinational corporations have expanded their presence across multiple continents to offer their services or products directly to every region while capturing local talent and resources. Nevertheless, these distributed corporations still operate as integral units thanks to the interconnectivity provided by global data infrastructures, which have been frequently compared to human nervous systems. In 1999, Gates predicted that future organizations would have their systems and processes united, forming a digital nervous system that would enable them to increase efficiency, growth and profits [27]. Similarly, Kephart and Chess also envisioned computer systems to behave as nervous systems, but they took it a step further by adding the notion of autonomy. Autonomic computer systems would be designed to self-configure, self-optimize, self-heal and self-protect their own infrastructure [50]. Mitchell also supported the vision of autonomic systems, but on a broader context, he envisioned that the integration of large-scale computer systems into cities would make civil infrastructures behave as nervous systems [67]. Aligned with this vision of digital nervous systems, today, the data centers in the IT infrastructure of an enterprise behave as body organs that exchange signals through request-response messages so as to coordinate actions such

as data visualization and manipulation.

These globally distributed organizations never sleep, having personnel visualizing, generating, manipulating and sharing information assets simultaneously throughout all time zones. This *modus operandi* was never exclusive of telecom or internet companies, in which the infrastructure itself represents the core business; but global collaboration also became key for a wide variety of other organizations. Today, banking, pharmaceutical or automobile industries cannot properly function without a platform and a set of tools that facilitate the creation, visualization and manipulation of information across remote locations. For these reasons, performance, reliability and availability these infrastructures are major concerns for these organizations, and their optimum operation at a low cost has become a key factor for differentiation against the competition.

Data sharing and collaboration capabilities have given global organizations the flexibility, agility and efficiency to operate without pause. However, these advantages have also lead to an unprecedented dependency on IT infrastructures. Kembel reported that each hour of downtime can be costly, from $200,000 per hour for an e-commerce service like Amazon.com to $6,000,000 per hour for a stock brokerage company [48]. Almost without exception, downtime is considered unaffordable and oftentimes the performance of the system and the availability of fresh information are sacrificed to keep the system operating. Unfortunately, a fully operational infrastructure cannot be left "as is" either, and three factors require making continuous adjustments to the system:

1. *Continuous Innovation*: Continuous integration of new features, state-of-the-art technologies or improved practices are necessary in order to maintain a competitive edge. For example, eBay deploys 300 features per quarter and adds 100,000 new lines of code biweekly [80].

2. *Continuous Cost Reduction*: As the infrastructure grows, continuously reducing the complexity of the infrastructure is a key mechanism to reduce costs. Akella et al. [2] propose out-of-the box solutions, component reuse, consolidation,

standardization and interface simplification as critical efforts to succeed in the goal of reducing the complexity of IT systems.

3. *Continuous Failure*: Hardware failure is unavoidable. Typical data centers are composed by thousands of commodity servers that will inevitably fail, and hence, IT infrastructures needs to be designed to continuously deal with the dynamics of failure. During a year, on a cluster of 2000 nodes, Google reported 20 rack failures, 1000 machine crashes, thousands of hard drive failures among a variety of other network, configuration and power related incidents [21].

Under these circumstances, two driving forces, the need for "change" and the need "not to change", collide. Consequently, decisions susceptible of affecting IT operations need to go through exhaustive reviewing processes across individuals, groups and divisions of the corporation so as to minimize the risk of stopping the natural flow of information. Often the implementation of non-critical features, cutting-edge technologies and latest software updates or protocols, is delayed, in order to preserve the stability of the system.

In this thesis, the construction of a Global Data Infrastructure Simulator, called GDISim, is presented, in order to evaluate the impact of 'what if" scenarios on the performance, availability and reliability of large-scale computer systems. The simulator takes as input the workload of each application, the resources allocated by individual client requests, the network topology of the organization, the hardware configuration deployed in each data center and details on background processes. Using this information, the queueing network models that the simulator is built upon produce estimates of the response time for user requests, along with measurements of the hardware allocation and network occupancy, so as to facilitate optimization goals for data center operators.

The information generated by the simulation platform can be used towards diverse optimization purposes as shown in Figure 1-1:

1. *Performance Estimation*: Enables the response time to be evaluated for a given workload, network topology, hardware configuration and software application.

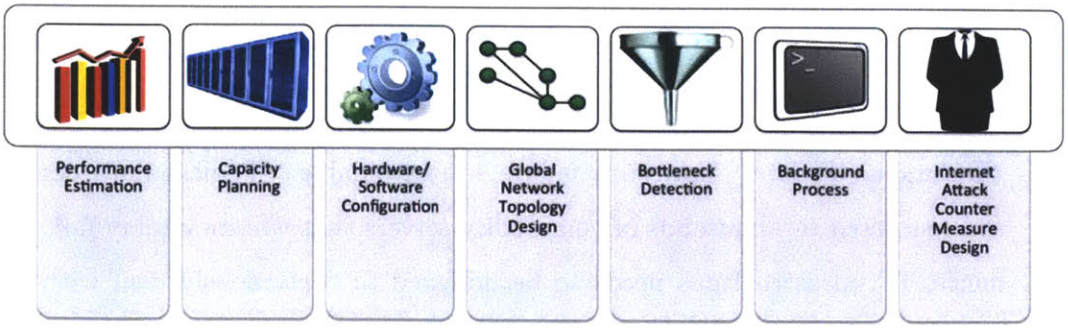

Figure 1-1: Potential applications for the Global Data Infrastructure Simulator.

2. *Capacity Planning*: Enables the data center operator to determine the resources required to meet Service Level Agreements (SLA) for each distributed application running on the infrastructure.

3. *Hardware/Software Configuration*: Enables both hardware and software parameters to be calibrated to achieve optimal performance and utilization of resources.

4. *Network Administration*: Allows the topology of the global network to be designed to cope with the expected traffic while maximizing its utilization.

5. *Bottleneck Detection*: Enables potential infrastructure bottlenecks to be identified and prevented.

6. *Background Job Optimization*: Facilitates the scheduling and effectiveness of jobs such as synchronization, replication or indexing without degrading user response times.

7. *Internet Attack Protection*: Allows the evaluation of the effects of denial-of-service attacks and facilitates the design of counter measures to fight them.

Chapter 2 reviews previous research on computer system modeling. The variety of mechanisms to reproduce computer system behavior are covered and the contributions provided by GDISim and differences to previous work are emphasized.

Chapter 3 presents the principles and models that GDISim is built upon. Special attention is given to the queueing network models utilized to represent hardware

components and the message cascade representation used to describe software applications.

Chapter 4 contains detailed information on the implementation of the simulation platform. The asynchronous messaging mechanisms and coordination primitives utilized to parallelize the calculations and boost the performance of the simulator are covered.

Chapter 5 validates the models introduced in Chapter 3 by profiling the performance of a downscaled version of a real data infrastructure used by a Fortune 500 company and comparing it with results obtained by simulating the same system. The accuracy results are analyzed and compared to other simulators.

Chapter 6 contains a case study that demonstrates the applicability of GDISim on a data center consolidation problem. The daily operation of a data center infrastructure of a global collaborative design company running Computer Aided Design, Visualization and Product Data Management software applications is modeled and simulated.

Chapter 7 takes the case study in Chapter 6 a step further by proposing a different mechanism to run background processes that maximizes their effectiveness. The results obtained by simulating the infrastructure with this new mechanism are reported and analyzed.

Chapter 8 summarizes the conclusions and lessons learned from this work, while Chapter 9 presents three different research directions that could be pursued to improve GDISim.

Chapters 2 and 3 utilize a three $(A/B/C)$ or six $(A/B/C/K/N-D)$ factor notation known as *Kendall's notation*. The details of this standard system utilized to classify queueing models are gathered in Appendix A.

Chapter 2

Related Work

2.1 Introduction

The existence of computer systems has always been accompanied by the demand to evaluate their performance, not only motivated by the need to control their cost, but also by the requirement to understand their functionality, reliability, security and availability characteristics. Furthermore, the relevance of evaluation techniques has grown in consonance with the complexity of computer systems making performance study a key component of the design, development, configuration and calibration of any computer system infrastructure.

Earliest developments on the evaluation of the performance of computer systems go back to the mid-1960s, when time sharing systems were first modeled using queuing models. Since them, research initiatives addressing the development of better evaluation techniques fall into three areas as shown in Figure 2-1: *Analytic Models*, *System Profiling* and *System Simulation*. Complex evaluation techniques may combine concepts that belong to one or more of these areas.

This chapter utilizes a three ($A/B/C$) or six ($A/B/C/K/N-D$) factor notation known as *Kendall's notation*. The details of this standard system utilized to classify queueing models are gathered in Appendix A.

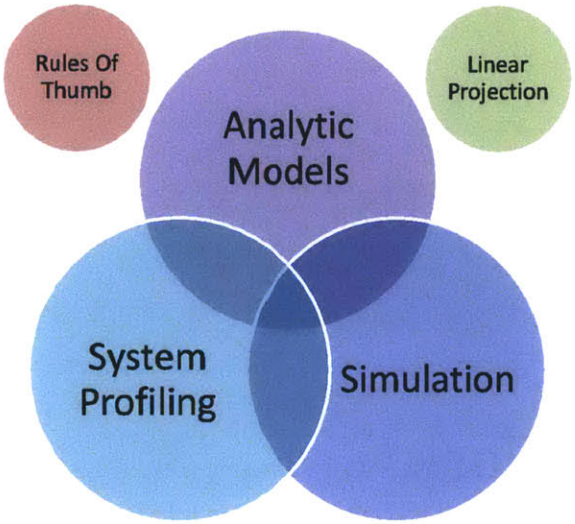

Figure 2-1: Computer system evaluation method techniques.

2.2 Analytic Models

Rules of Thumb have been popular tools for the estimation of performance and capacity in day-to-day operations of computer systems. Heuristics such as Moore's Law [78] and Gilder's Law [31] were obtained by observation and have accurately predicted the yearly growth in the number of transistors on an integrated circuit and the increase on bandwidth availability of communication systems respectively. These types of rules and other *Linear Projection* techniques based on extrapolation have been frequently considered for the estimation of storage, processing and networking costs [37][36]. Nevertheless, these should be carefully utilized since they apply linear model assumptions to systems known to be inherently nonlinear. Under these circumstances, a cost-effective but yet sophisticated technique to understand computer systems is the construction of *Analytic Models*, which describe the behavior of a system through mathematical closed form solutions. The historical evolution of analytic models is explained as follows:

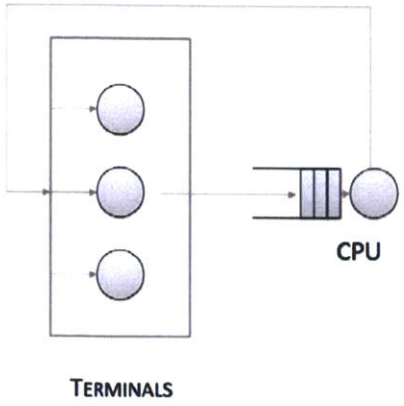

Figure 2-2: Time sharing system modeling a single CPU.

2.2.1 Pre-1975 Developments

Traditionally, complex computer systems have been analytically represented using queueing theory. Initially, in the mid-1960s, the earliest published works analyzed queueing models of time sharing systems [3] [51]. These models were single server queues with Poisson arrivals, in which the only computer resource modeled was the CPU. This is described in Figure 2-2. The purpose of these models was to study the performance of different processor scheduling algorithms. This research lead to the creation of the Processor Sharing (PS) queueing discipline, in which all jobs received simultaneous service by the CPU with a rate inversely proportional to the number of jobs [52].

Next, research in the field advanced to contemplate multiple resources as part of a single model. This was the case for multiprogramming computer systems [29] [15], in which multiple programs were allowed to simultaneously contend for resources. These systems were modeled as closed queueing networks: a single server queue representing the CPU and a single server queue for each I/O device modeled. This is illustrated in Figure 2-3.

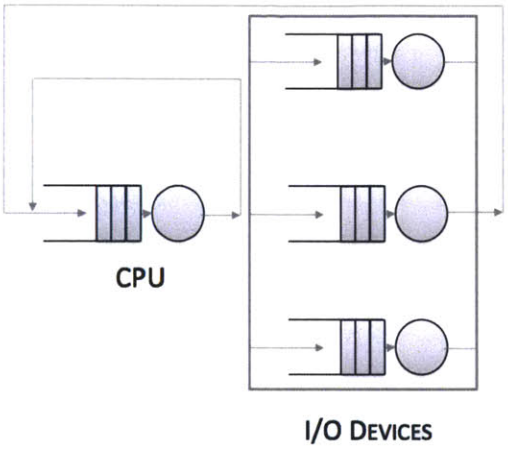

Figure 2-3: Central server model of a multiprogramming system including a CPU and multiple I/O devices.

2.2.2 1975-1990 Developments

This period focused on the representation of additional features, such as memory management and I/O subsystems, aiming to enrich the existing models of computer systems.

Brown et al. [14] first, and Bard [8] later, provided models to evaluate the effects of finite memory size and workload memory requirements in queueing network models. Figure 2-4 illustrates these enhanced models, which included an additional memory queue to represent the contention for memory access and memory partitions to represent memory allocation and release.

During this period of time, modeling the time spent by a job waiting for and receiving I/O service became a priority. Queueing models of I/O subsystems started considering the mechanical nature of disk drives and became the basis for subsequent modeling work. Wilhelm et al. [89] modeled several moving head disk units attached to a single I/O channel and differentiated between seek, latency and transfer times. Figure 2-5 describes this model, in which multiple seek operations can be carried out simultaneously, while seek command request and transfer operations depend on the availability of the channel.

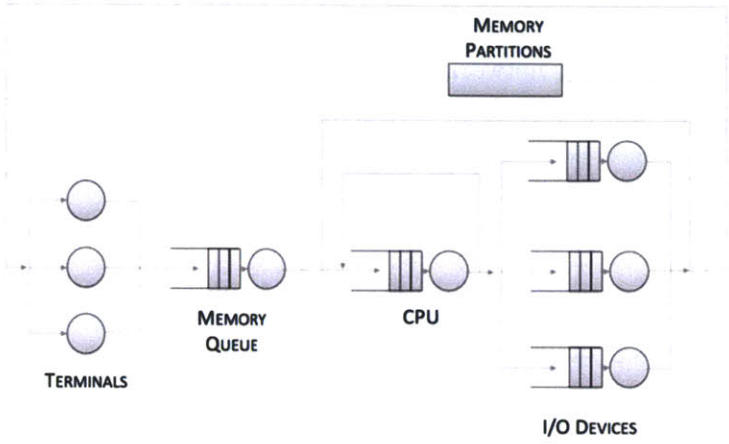

Figure 2-4: Memory management model (memory queue and partitions) in a time shared multiprogramming system.

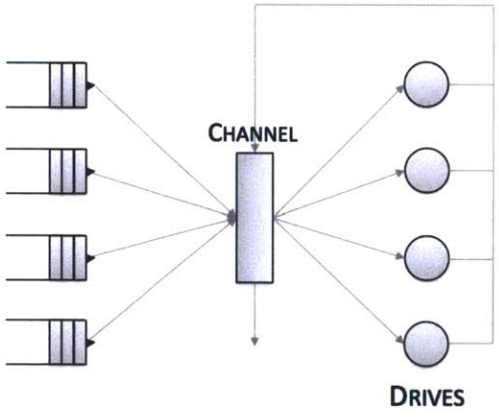

Figure 2-5: General queueing model for multiple disk drives.

2.2.3 Post-1990 Developments

During the last two decades, computer systems have grown exponentially in complexity and scale, and hence, analytical models have tried to keep up describing their behavior. Today, queueing network models are used to represent systems with different granularities and scales. The granularity varies from single isolated hardware components, to their aggregation to form servers and supercomputers. The scale varies from a single component to arrays of them, or combinations of arrays of identical components. The most relevant initiatives are presented as follows:

Low-level hardware components are frequently represented by simple queuing configurations. Multi-core CPUs have been represented by $M/M/c$ queues [79] [64] and disk arrays have been modeled using fork-join $M/M/c$ queues [59] [86].

These low-level components are used as building blocks for the construction of models for higher-level entities, such as computer servers. The goal behind the modeling of servers is to facilitate the assignation of hardware resources towards the fulfillment of predefined Service Level Agreements (SLA). Doyle et al. [25] interconnect low-level models for server memory, CPU and storage I/O to create a tool that simplifies utility resource management given a series of service quality targets.

A different but yet popular approach is to represent each server by a single queue. This initiative has been popular when constructing analytic models for server tiers. The application server tier for an e-commerce system is described as a collection of $M/G/1$-PS queueing systems by Villela et al [87]. This queuing system is used to construct an objective function that describes the cost of minimizing service misses that threaten to break SLAs established with clients. Similarly, Ranjan et al. [72] present a Java application tier formed by n servers using a $G/G/n$ queue.

Recently, analytic models for multi-tier configurations running in data centers have been explored. Urgaonkar et al. [85] describe a multi-tier model in which each tier is represented by a $M/M/1$ queue and the queues are interconnected on a Markov Chain. This model allows capturing effects such as session based workloads, caching between tiers and load balancing between replicas. The Markov Chain is illustrated

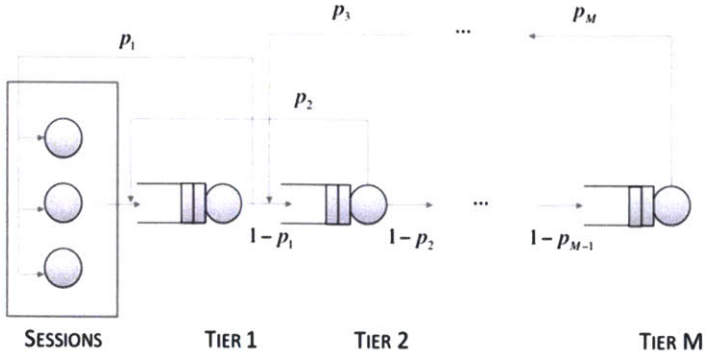

Figure 2-6: Queuing network model for multi-tier configuration in a data center using a Markov Chain.

in Figure 2-6, where p_i with $i = 1 \ldots M$ represents the probability of moving from queue i to $i-1$ and $(1-p_i)$ with $i = 1 \ldots M$ the probability of moving from queue i to $i+1$. Bi et al. [11] also model a virtualized multi-tier data center, but considering each virtual machine in each tier as a $M/M/1$ queue, preceded by a serving system represented by a $M/M/c$ queue.

Finally, it is necessary to mention approaches to analyze data transfer latency using queuing networks. Ramachandran et al. [71] propose a queueing network model to estimate overall file transfer time in peer-to-peer network topologies. The model, illustrated in Figure 2-7 uses $M/G/1/K\text{-}\ PS$ queues to represent peers and $G/G/1$ queues for routers. Similarly, Simitci [81] explains the modeling of bulk data transfers using a closed queuing network model.

2.3 System Profiling

Another popular technique to obtain performance insights on information infrastructures is *Profiling*. Profiling tools analyze the behavior of one or more programs and use the information collected during their execution. Typically, the goal of the analysis is to optimize sections of the program by increasing their speed and reducing resource requirements. Similar to *Benchmarking* techniques, *Profiling* is a sampling-based method that measures a collection of indicators within machines in a data

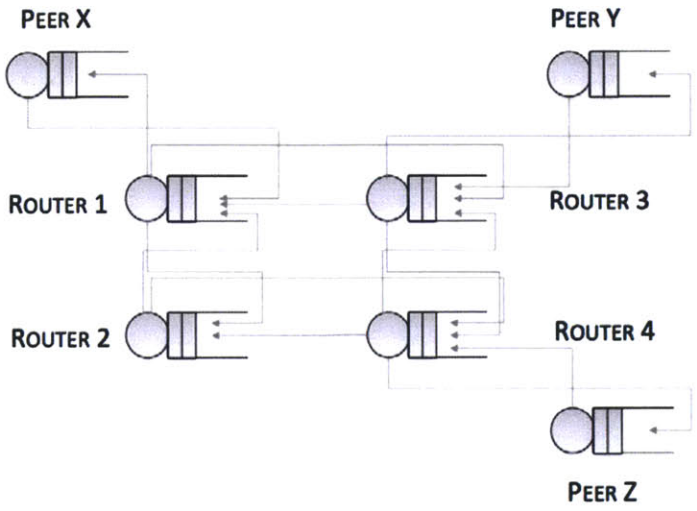

Figure 2-7: Queuing network model for a peer-to-peer network topology modeling peers and routers.

center. Nevertheless, *Benchmarking* is as simple as comparing the execution of the same piece of code across multiple types of hardware and configurations within an isolated environment. *Profiling*, on the other hand, is an intrusive procedure that collects fine-grained measurements such as stack traces, hardware events, lock contention profiles, heap profiles, and kernel events- during the normal activity of the data center. For these reasons, in *Profiling* it is critical to maintain the overhead and distortion to acceptable levels not to hinder the normal operation of the data center.

2.3.1 Single Execution - Single Program - Single Machine

Traditionally, *Profiling* focused on the analysis of a single execution of a single program on a single machine. This was the case of Intel *VTune* [43], a commercial performance analyzer for Intel-manufactured x86 and x64 machines and its open-source counterpart *gprof* [35]. These profilers provided mechanisms to decompose a function into call graphs and measure the time spent in subroutines. They carry out both, time-based sampling so as to find hotspots and event-based sampling to detect cache misses and performance problems. Recently, Intel provided *Parallel Amplifier* [44], an additional call graph analysis tool that accounts for concurrency, locks &

waits.

2.3.2 Continuous Profiling

As computer systems evolved, they were required to be "always-on" and thus, profiling was adjusted to be carried out continuously.

The *Morph* system, by Zhang et al. [91], proposed a solution that combined operating system and compiler. Morph collected profiles with low overheads ($\leq 0.3\%$), and provided a binary rewriting mechanism to optimize programs to their host architectures.

Similarly, Digital Continuous Profiling Infrastructure (DCPI), by Anderson et al. [5], provides a data collection subsystem that generates more detailed execution profiles, and as opposed to Morph, focuses on the presentation of the collected data to data center operators. DCPI, illustrated in Figure 2-8, is composed by three modules: 1) A *Kernel Driver* that services the performance counter interrupts; 2) A *Daemon Process* that extracts samples from the driver, associates these with an image of the executable and writes the data to the profile database; and 3) a *Loader* that identifies and loads executable images.

OProfile [60] takes these principles a step further. Its open source nature and acceptance have made it stable over a large number of different Linux systems. Analogous to DCPI, *Oprofile* consists of a kernel driver and a daemon process for collecting hardware and software interrupt handlers, kernel modules, shared libraries and applications, with a low overhead.

2.3.3 Infrastructure Profiling

Cloud computing infrastructures present additional challenges surfaced by heterogeneous applications, unpredictable workloads and diverse machine configurations making infrastructure profiling a daunting task. Nevertheless, it has been proven that profiling an IT infrastructure at scale can be successfully performed, as shown by Google-Wide Profiling (GWP) [73]. Unfortunately, its high cost puts it out of

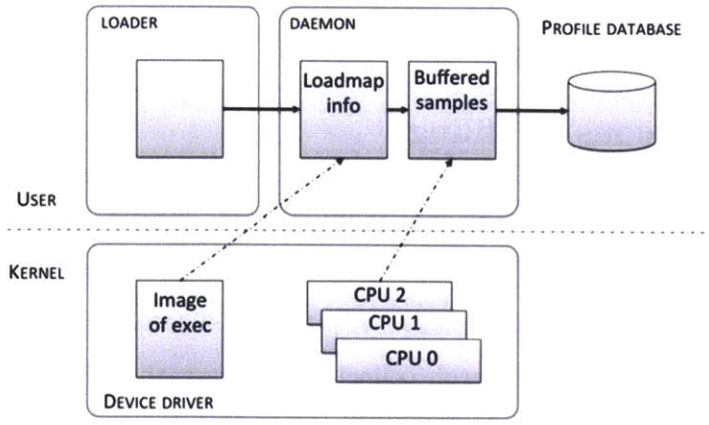

Figure 2-8: Digital Continuous Profiling Infrastructure (DCPI) collection system from Anderson et al..

reach for many corporations. GWP, illustrated in Figure 2-9, is an *OProfile* based continuous profiling solution that scales to thousands of nodes across multiple data centers. GWP provides fine-grained information - speed of routines and code sections, performance difference across versions, lock contention, memory hogs, cycle per instruction (CPI) information across platforms - for internet-scale infrastructures, and hence, requires an additional infrastructure with tens or hundreds of nodes to analyze the collected profiles. Today, only a handful of companies can justify the cost of this deployment, and it is out of reach for many non-Internet organizations.

Although its is out of the scope of this research, it is necessary to mention the special attention that energy profiling has received with the advent of cloud computing infrastructures. The latest techniques use profiling and prediction to characterize the power needs of hosted workloads. Govidan et al. [34] use a combination of statistical multiplexing techniques to improve the utilization of power in a data center. Ge et al. [30] designed a tool called PowerPack that isolated and measured the power consumption of multicore, disks, memory and I/O and correlated the data with the application functions. Kansal et al. [45] take energy profiling a step further and study the tools and techniques needed to design applications taking into account energy profiling and performance scaling.

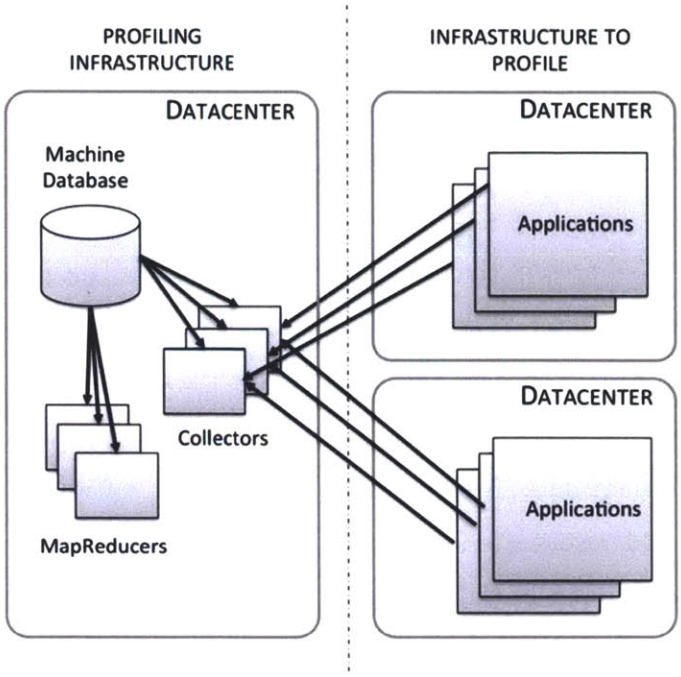

Figure 2-9: Google-Wide Profiling (GWP) infrastructure and the infrastructure to be profiled by it.

2.4 System Simulation

Simulation has become a popular technique for representing complex computer systems. The level of detail and complexity that can be achieved by combining simpler models exceeds the capabilities that analytic models in queueing theory can reach in practice. For smaller infrastructures, there might not be sharp cost distinction between constructing a simulator and profiling methods. Nevertheless, as the infrastructure to model grows in complexity, the simulator results in a more cost-effective solution. In this section, queueing network based and non-queueing network based simulation platforms are introduced.

2.4.1 Queueing Network-based Simulators

Traditionally, analytic models using queueing networks to represent computer systems abstract entire servers or data center tiers into a single queue. In practice, analytic

models are not capable of representing data center tiers as arrays of servers composed by interconnected queues that correspond to every hardware component. In contrast, simulation platforms enable implementing arbitrarily complex networks of queues that reproduce data center behavior with the greatest level of detail and on a flexible manner.

Kounev et al. [55] present the implementation of a closed queueing network model for a multi-tier data center. They model CPUs in application and database servers as Processor Sharing (PS) queues and the access to the disk subsystem is represented by a First Come First Served (FCFS) queue. The workload loaded in the simulator corresponds to a distributed supply chain management software application. The processing cost and resource allocation of each user action in the application was profiled and used as input of the simulator.

Steward et al. [83] follow a similar approach, but combine queueing network models with linear models, so as to represent multi-tier data centers and predict their response times and resource utilization.

Multi-tier Data Center Simulator (MDCSim), by Lim et al. [61], takes these approaches a step further. In addition to model all the components in a server -CPU, I/O and NIC- as $M/M/1$ queues, the simulator focuses on capturing the particular idiosyncrasy of each different type of tier -web, application or database-. A diagram showing the MDCSim model used for the simulation of a data center is shown in Figure 2-10. This effort pays particular attention to the modeling of the interconnections between servers, facilitating the comparison between technologies such as Infiniband or 10 Gigabit Ethernet.

2.4.2 Non-Queueing Network-based Simulators

Not all computer system simulation platforms have been designed upon the interconnection of queueing models. In this section, alternative approaches not using queueing network theory are presented.

Rosenblum et al. [76] [75] present SimOS a computer system simulator capable of modeling computer hardware and analyzing the impact that OS processes have in

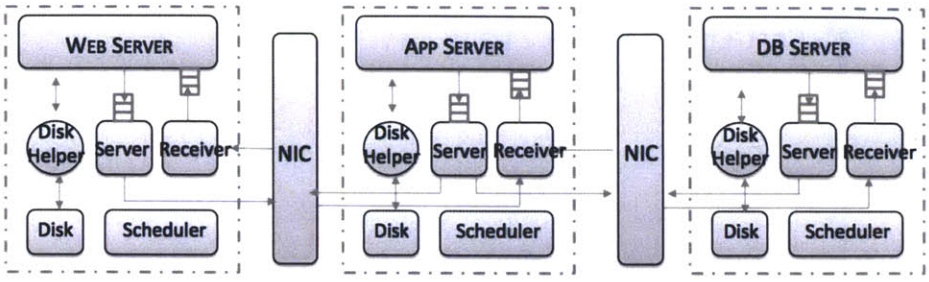

Figure 2-10: The structure of the Multi-tier Data Center Simulator (MDCSim) illustrating web, application and database server tiers.

conjunction with multiple concurrent application workloads. SimOS is decomposed into hardware components - e.g. processors, memory management units, disks, ethernet and cache-, and is constructed to accept various specifications and levels of detail for each hardware component. Therefore, this simulator allows operators navigating into two dimensions: 1) Workload: They can focus on specific workload parts of interest. 2) Detail: Once the workload segment of interest has been selected, detailed models can be used to understand the system behavior exhaustively. SimOS does not use mathematical models to represent the behavior of each component, it reproduces in software the functional behavior of the hardware component instead.

Similarly, Austin et al. [6] provide an open source infrastructure for computer architecture modeling called *SimpleScalar*. Like SimOS, the models utilized to construct the simulator are not based in mathematical solutions but on the reproduction in software of functional behavior of the hardware. As opposed to SimOS in which the primary goal was to facilitate capacity planning given specific application workloads, the authors of *SimpleScalar* offer this tool to facilitate the work of the computer architecture research community.

2.5 Contributions & Differences

GDISim is designed to reproduce the behavior of globally distributed data centers in which clients in different time zones visualize, manipulate and transfer data concurrently using a variety of software applications. At the core, it is a simulation platform,

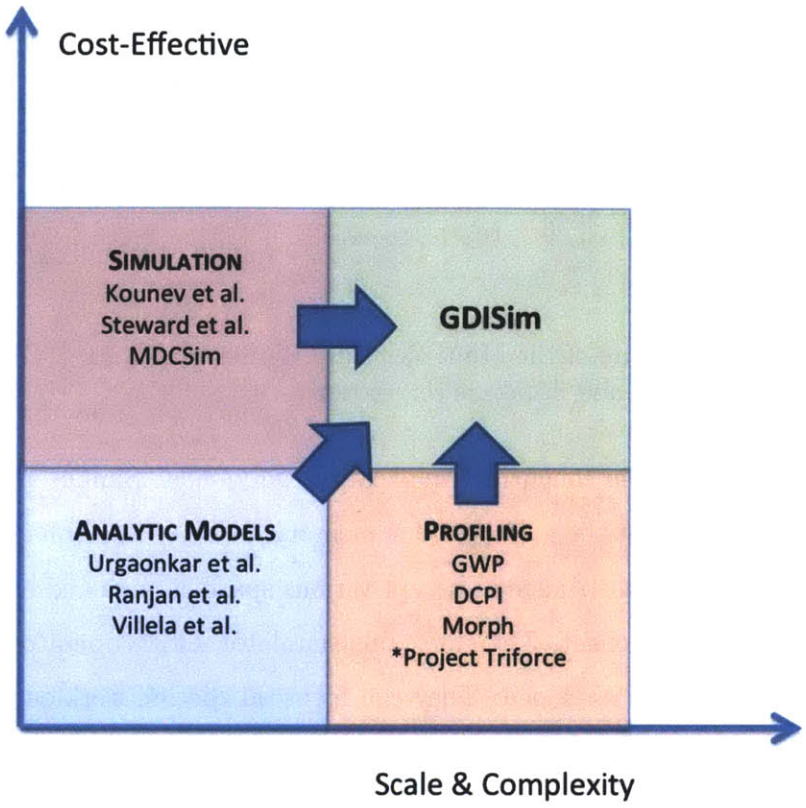

Figure 2-11: Quadrant system illustrating the location of GDISim with respect to similar work in the field of computer system evaluation. * In this work, Project Triforce [56] is classified as profiling system instead of a simulator.

but is constructed by implementing and combining analytical models described in Section 2.2, and feeding them with the data measured by profiling mechanisms explained in Section 2.3. GDISim can reproduce the behavior of systems with the scale and detail that profiling techniques deliver, while keeping the cost orders of magnitude lower. Figure 2-11 illustrates the location of GDISim in a quadrant system with respect to similar work in the field of computer system evaluation. In this section, the contributions that this work has made to computer system evaluation techniques are emphasized along with its differences with previous research.

2.5.1 Contributions

- *Global Data Center Infrastructure:* The scale of the information managed by globalized businesses and the flexibility provided by virtualization technologies, have enabled grouping all the IT systems across an enterprise into a single concept of IT infrastructure. As opposed to other analytic models and simulations preceding this work which only considered a single piece of hardware, a single computer or a single datacenter, this work presents the first platform for the simulation of an entire global infrastructure composed by multiple multi-tier data centers connected through networks, and serving clients across continents.

  MDCSim [61] is the computer system simulator that shares the largest number of characteristics with GDISim. Both efforts focus on the simulation of multi-tier data centers using queueing network models as a foundation and seek to provide estimates that facilitate the efficient and cost-effective design of these systems. However, GDISim differentiates from MDCSim in the following aspects: 1) *Queueing Models*: MDCSim models all the components of a server as $M/M/1 - FCFS$ queues. Even though it can produce satisfactory estimations of the overall latency and throughput of a data center, MDCSim does not include models to predict CPU or bandwidth utilization. The models that GDISim is built upon produce computational and network utilization estimates that can be used towards capacity planning of the infrastructure. 2) *Multi-Data center*: MDCSim was constructed to simulate a single data center loaded with the RUBiS benchmark [17]. While the same work could be taken further to simulate the operation of an infrastructure composed by multiple data centers, the authors did not consider this direction and did not provide a software application model to represent distributed enterprise software nor background processes. On the contrary, GDISim is conceived as an infrastructure simulator and utilizes a messaging cascade structure that reproduces arbitrarily complex interactions between data centers and tiers.

- *Application Diversity:* GDISim takes the modeling of software operations a step

further and presents a message passing mechanism capable of reproducing arbitrarily complex interactions between data centers components with high levels of detail. The consolidation of different IT systems into a single IT infrastructure has pushed multiple distributed software applications to run concurrently on the same resources using standard interfaces and platforms. As opposed to previous research [55] [83] [61] which only considered one or few independent applications, this work proposes a general software model designed to represent any distributed application and provides the capability of intertwining multiple workloads. Each application is modeled as a series of client operations, which in turn are decomposed into sequences of messages that convey encoded resource allocation information. These messages flow concurrently through the infrastructure altering the state of the components they pass through.

The messaging approach utilized by GDISim to simulate distributed software programs has similarities with previous research initiatives for the simulation of parallel computation. Dickens et al. [23] implemented a simulator tool called LAPSE for the performance prediction of massively parallel code. LAPSE supports parallelized direct execution and simulation of parallel message passing applications. The tool was validated against four scientific and engineering applications. Similarly, MAYA [1] is a simulation platform for evaluating the performance of parallel programs on parallel architectures that is also build upon message passing mechanisms on shared memory systems. However, none of this approaches designed a message passing mechanism that represents arbitrarily complex interactions in multi-data center infrastructures.

- *Background Jobs:* Background jobs are complex data manipulation and transfer tasks periodically scheduled by daemon processes. As opposed to other simulation initiatives, GDISim contemplates the execution of client workloads simultaneously with background processes. Popular background jobs are *Synchronization*, *Replication* and *Indexing*. Distributed data infrastructures need *synchronization* processes to guarantee that the latest versions of the data are

locally available to remote clients, and hence, avoid costly on-demand synchronization between data centers. *Replication* guarantees that each file has replicas in different locations (different server and/or data center) and it can be integrated into the synchronization process or executed independently. Similarly, as data volumes grow, search functionality is desired, which involves the execution of *indexing* processes that analyze each file and its relationships. Typically, background processes need more resources than individual user actions requiring copy and movement of large volumes of files and utilizing computational resources to process the information and generate metadata. If the execution of these and any other background jobs is not scheduled properly, overly frequent jobs can degrade client experience in the form of increased response times, whereas infrequent jobs lead to serving stale files or searching through outdated indexes.

The optimization of background file movement looking to guarantee the immediate availability of data files in the location of interest has been the focus of attention a variety of research initiatives in grid computing. Ko et al. [53] explore *worker-centric* scheduling strategies to distribute tasks among a group of worker nodes exploiting the locality of files and looking to minimize transfers in data-intensive jobs. The authors of this work simulate different task distribution strategies to compare their effects in network utilization and computation time. Similarly, Meyer et al. [65] utilize *Spatial Clustering* to derive a task workflow based on spatial relationships of files. This strategy increases data reuse and reduces file transfers by grouping together tasks with significant overlaps in input data.

While GDISim also focuses on the simulation of worker-centric strategies that take advantage of data locality, it differentiates from previous research in three aspects: 1) *Background Jobs & Client Workloads*: Previous research focused on the simulation of scheduling mechanisms to optimize file movement of standalone data-intensive jobs in computer grids. However, GDISim can simulate

jobs that aim to optimize transfers of files in the background, running concurrently with a population of thousands of users visualizing and manipulating the same files. 2) *Data Locality*: Previous efforts focused on exploiting data locality explicitly given by the spatial characteristics of the problem, while GDISim simulates infrastructures in which data locality is determined by non-deterministic user access patterns. 3) *Change Propagation*: Research in grid computing focused on the consumption of read-only data, while GDISim simulates the propagation of changes in one data center to the others.

- *Simulator Validation:* GDISim pioneers the simulation of an interconnected data center infrastructure using queueing networks and validated with data collected from a real infrastructure. The accuracy of the models utilized to construct the simulator was validated against the global IT infrastructure of a Fortune 500 company running three distributed software applications: Computer Aided Design (CAD), Visualization (VIS) and Product Data Management (PDM). This infrastructure was comprised by multiple data centers, with thousands of clients visualizing and manipulating files, while synchronization, replication and indexing jobs were carried out simultaneously in the background.

  Facebook reported the use of a simulator, called Project Triforce [56], to evaluate the impact of the partitioning of the social networking site from two to three geographically distributed data centers. The company describes the isolation and reconfiguration of thousands of servers in the active production cluster in one of the two original data centers to make it look as the future third data center. While this efforts seeks to provide the same functionality as GDISim, Facebook relies on reproducing the behavior of the future configuration using real hardware and real software. This effort is closer to an emulation than to a simulation, and even though it may produce satisfactory estimations, its overall cost will be closer to profiling techniques than the simulators explained in this chapter.

  Calheiros et al. [16] describe the construction of a simulation framework, called

CloudSim, for the evaluation of virtualized cloud computing infrastructures. This work focuses on reproducing the effects of scheduling policies on virtual machines executing tasks in multicore hardware. The contributions provided by CloudSim and GDISim are complementary.

- *Platform Scalability:* GDISim is the first data center simulation platform that pays special attention to multicore scalability so as to reduce execution time. Reproducing the daily activity on tens of interconnected data centers with thousands of clients, numerous applications and multiple background jobs, can lead to the elapsed simulation time being greater than the simulated time. As important as getting accurate predictions from the simulator is getting them in a timely manner. Under these circumstances, GDISim was designed and implemented to run multithreaded using highly efficient coordination of asynchronous messages.

Early work on simulation of parallel computer systems were constructed to be executed sequentially on the host machine. Examples of this research include SimpleScalar [6] and Parallel SimOS [58]. However, over the last decade, a number of parallel simulators that target parallel architectures have been developed. Examples of multithreaded simulators of multicore systems are provided by Miller et al. [66] and Almer et al. [4]. Both approaches utilize Dynamic Binary Translation (DBT) and report significant speedups against their serial counterparts.

While most multithreaded simulators for the evaluation of computer systems focus on predicting the behavior of multiprocessor chips, GDISim targets the parallel simulation of entire data center infrastructures, which may or may not include servers with shared-memory multiprocessors.

2.5.2 Differences

Simulation vs Analytic Models

In the past, the principal weakness of simulation of computer systems was its relative expense compared to analytical models. Nevertheless, as the complexity of computer systems increased, their representation through analytical modeling soared too and oftentimes became intractable. In the meantime, simulation costs remained steady and modern technologies facilitated the implementation and debugging of complex computer programs.

The principal strength of simulation is its flexibility, as opposed to analytic models which are rigid. Simulators can reproduce the behavior of an arbitrary complex queueing network, and integrate specific behaviors with arbitrary level of detail. A modular design of the simulation framework should allow computer systems modules to be added, removed or reutilized easily, broadening the applicability of the simulator and allowing it to adjust following the evolution of the real system itself.

Finally, it is necessary to point out that simulators can have a large number of input parameters. Often these parameters are expensive to evaluate. In this work, we propose to obtain the majority of the input parameters through small-scale profiling of the infrastructure in a laboratory.

Simulation vs Profiling

Profiling data center behavior requires running processes that measure vast amounts of highly detailed information. As the infrastructure size increases, the profiling overhead can degrade system performance and leave collected data unexploited, unless additional resources are allocated, which increases the overall profiling cost. The simulation platform is a simpler yet powerful non-intrusive tool capable not only of reproducing system behavior on a high level, but also predicting the impact of "what if" scenarios for a lower cost.

While data center operators can focus on profiling and evaluating different parts of an infrastructure independently, the complexity of the infrastructure makes it com-

plicated for a single data center operator to collect the dynamics of the workload, hardware, software and network required for simulation experiments of the entire system. In fact, typically individual operators do not have detailed knowledge about the system beyond their area of responsibility. Even though this can be a limiting factor, our experience indicates that joint collaboration of multiple data center operators to produce the collection of inputs for the simulation platform can be beneficial. This collaborative effort gives operators a common view of the infrastructure, which upon reception of the predicted results by the simulator can be positively used towards finding consensus on system modifications.

2.6 Summary

In this chapter previous research on evaluating complex computer systems has been compiled. For the last four decades, the models and mechanisms to evaluate the computer systems have grown in complexity in consonance with the creation and evolution of internet-scale infrastructures. Previous work aimed to model or profile individual hardware components of the system, and oftentimes simplifications were made to represent servers or multi-tier data centers. Today, internet corporations can carry out company-wide infrastructure profiling, but unfortunately, this approach remains unreachable for the majority of organizations.

The work presented in this research defines a cost-effective simulation tool for the evaluation of global infrastructures. The simulator makes use of previous research on analytic models and utilizes profiling techniques to generate inputs for the simulator. No other simulation or analytical model represents a global information infrastructure with the level of detail of GDISim.

Chapter 3

Global Data Infrastructure Simulator

3.1 Introduction

In this chapter the foundations on which the Global Data Infrastructure Simulator (GDISim) is built upon are explained. This chapter aims to provide a conceptual vision of the simulator without any implementation details. First, the methodology followed to reproduce the impact that thousands of concurrent clients have on a distributed computer system is explained. Second, the Multi-Agent System (MAS) principles followed by the simulator are described and previous work on simulation of computer systems using MAS is succinctly reviewed. Third, the models utilized to reproduce the behavior and status of the hardware components that compose the infrastructure are analyzed in detail. Finally, the messaging model constructed to represent distributed software applications running across multiple data centers is presented.

3.2 Methodology

Each software application running on the infrastructure is decomposed into a list of client initiated actions denoted as *Operations*. Classical operations in software

applications running at internet-scale are login, file search or file open and classical background jobs are file synchronization, replication or file indexing. Background jobs running in the infrastructure are also represented by operations, but these are initiated by daemon processes instead. The complexity of each operation can range from a simple round trip from the client to an application server, to large sequences round trips between the client and multiple tiers transmitting large quantities of information. The execution of a single isolated operation of each type in the infrastructure yields the *canonical* cost for that type of operation.

As introduced in Chapter 2, queueing networks are a popular technique to model computer system hardware. In this research, a global IT infrastructure is decomposed into a large scale network of lower-level hardware elements denoted as *Components*. Each of these components is modeled as a queue or network of queues. Their interconnection reproduces the behavior of higher level entities such as servers or clients, and consequently, the infrastructure.

The simulator launches thousands of operations of different types and uses their canonical costs to estimate the cumulative effect caused by client workloads and background jobs running concurrently and competing for the global infrastructure resources. The predictions are obtained through the study of the interactions between these operations and the queueing network models representing the components that form the IT infrastructure.

3.2.1 Simulator Inputs & Outputs

The simulator takes a collection of parameters corresponding to the software and the hardware specifications as input, and reproduces the performance of software applications along with the utilization of hardware components as output. Alterations of the input parameter space enable discovering the outcome of hypothetical scenarios or approaching optimization goals. Next, the input parameters as well as output results are enumerated and explained, these are also illustrated in Figure 3-1.

Input Parameters of the Simulator

- *Software Applications:* For each different software application hosted by the infrastructure, the hourly client workload in each data center along with the messaging structure and canonical cost of the operations composing the application must be provided.

- *Background Jobs:* Background jobs are either scheduled periodically or triggered by events. The simulator requires this information along with the messaging structure and canonical cost of the operations representing the background job.

- *Data Centers:* The input for each data center definition is comprised of three elements: specifications, configuration and connectivity. The specification contemplates the number of tiers, servers per tier and hardware available in each server. The configuration considers load balancing policies and effects such as caching between tiers. The connectivity takes into account the bandwidth and latency of the links interconnecting the tiers inside the data center.

- *Global Topology:* The global topology represents the connectivity links between data centers distributed across continents including latency and bandwidth information, along with secondary links in case of failure.

Output Results of the Simulator

- *Software Applications:* The simulator produces estimates of the response time for each operation type and software application at each location of the infrastructure. Saturation of resources leads to degradation of the user experience which is manifested by increased response times.

- *Background Jobs:* Analogous to the software applications, the simulator returns estimates of each background job duration. Additionally, the performance of the background job can be evaluated based on a metric specific to the job itself.

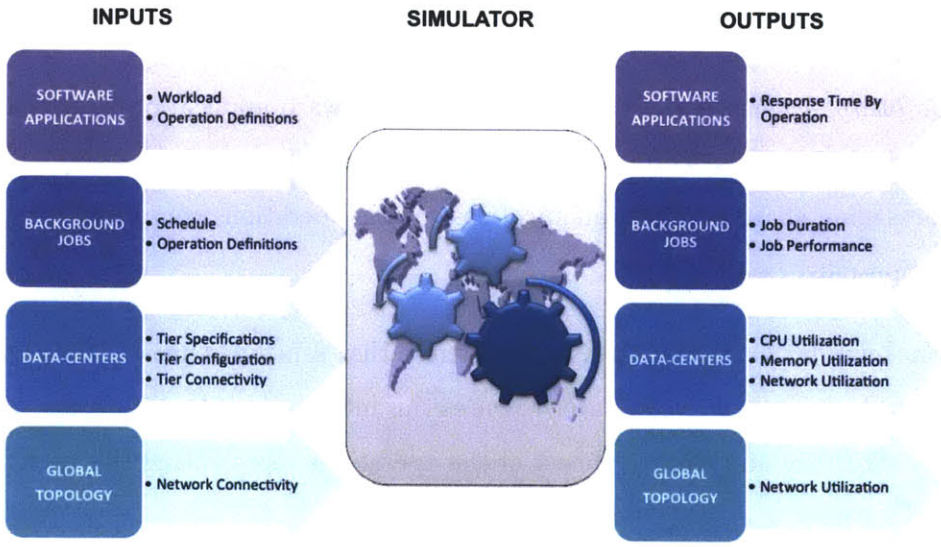

Figure 3-1: Input parameters taken by GDISim and Output estimations produced by the models in the simulator.

For example, file synchronization performance can be evaluated by measuring the longest time interval in which a data center can keep a stale copy. Similarly, indexing performance can be evaluated by measuring the longest time interval in which a file is unsearchable.

- *Data Centers:* The simulator returns computational, memory and bandwidth utilization measurements taken from every machine and tier in the data center.

- *Global Topology:* The bandwidth utilization estimate of the links connecting data centers in different continents is reproduced by the simulator.

3.3 Multi-Agent Systems

The Global Data Infrastructure Simulator is constructed following the principles of a Multi-Agent System (MAS). A MAS is a system comprised of multiple intelligent agents that have interactions with each other. Traditionally these systems are used to solve or simulate complex problems out of reach for monolithic systems.

In the context of computer systems, MAS are also referred as *Software Agents*,

in which a piece of software acts for a user, program or device in a relationship of agency [13]. This relationship is understood as the agreement to act on one's behalf. The main characteristics of agents were captured by Woolridge and Woolridge [90]:

- *Autonomy:* Agents are located in some environment and are capable of autonomous actions within the environment in order to meet their design objectives.

- *Partial Control:* Agents do not have complete control over the environment. At best, they can exercise partial control by influencing the environment through actions. Nevertheless, from the point of view of an agent, the same action performed twice in apparently identical circumstances can lead to different effects.

- *Failure:* Agent actions may not achieve the desired effect, therefore, the environment must be prepared to deal with failure.

MAS based simulations have the following advantages: 1) Flexibility: By definition MAS are modular. Hence, agents can be added, removed or reused and their behavior modified without critical changes on the simulation platform. 2) Scalability: The autonomous nature of the agents enables different degrees of parallelization that can be exploited using multithreaded platforms.

Additionally, MAS can also include recursive agents, also known as *Holons.* A Holon (Greek: holos "whole") is something that is simultaneously a whole and a part [54]. In the MAS context, Holons are agents built from other agents in which their behavior at a given level is a partial consequence of the behavior of the agents composing them. A hierarchy of nested holons is called a *holarchy* and an agent based system based on holons is called a Holonic Multi-Agent System (HMAS) [74]. The Global Data Infrastructure Simulator follows HMAS principles.

3.3.1 Related Work on MAS for Computer System Simulation

In the context of computer system simulation, MAS have been used for a variety of purposes. In this section some of these application areas are briefly described.

Niazi and Hussain [68] propose the utilization of agent-based tools for modeling and simulation of peer-to-peer, ad hoc and other complex networks. They evaluate how agent-based tools can help the understanding of networks involving the interaction between humans and the environment.

Karnouskos and Tariq [46] explore the simulation of heterogeneous web-service enabled (SOA-ready) devices using a MAS. Their simulator facilitates the test of aspects such as communication overheads and performance.

Gorodetski and Kotenko [33] utilize MAS for the simulation of attacks against computer networks and facilitate the design and testing of intrusion detection and learning systems. The authors of this work emphasize the advantages provided by MAS to simulate coordinated distributed attacks of different types.

Huang et al. [42] utilize an agent system for the parallel simulation of a High Performance Computer (HPC). Their goal is to provide a scalable tool that facilitates the architecture design of high performance systems.

Gaud et al. [28] implemented a multi-agent platform called JANUS designed to support to holonic multi-agent systems. This tool is part of a larger effort aiming to facilitate the simulation of complex systems and increase their modularity and reusability.

3.3.2 Holons, Agents and Messages

GDISim is composed by two building blocks: *Hardware Components* and *Software Operations*. In the context of an HMAS based simulator, hardware components are represented by holons (or agents) and software operations define the communication between them through messages. Figure 3-2 illustrates the holons and agents composing the HMAS simulator.

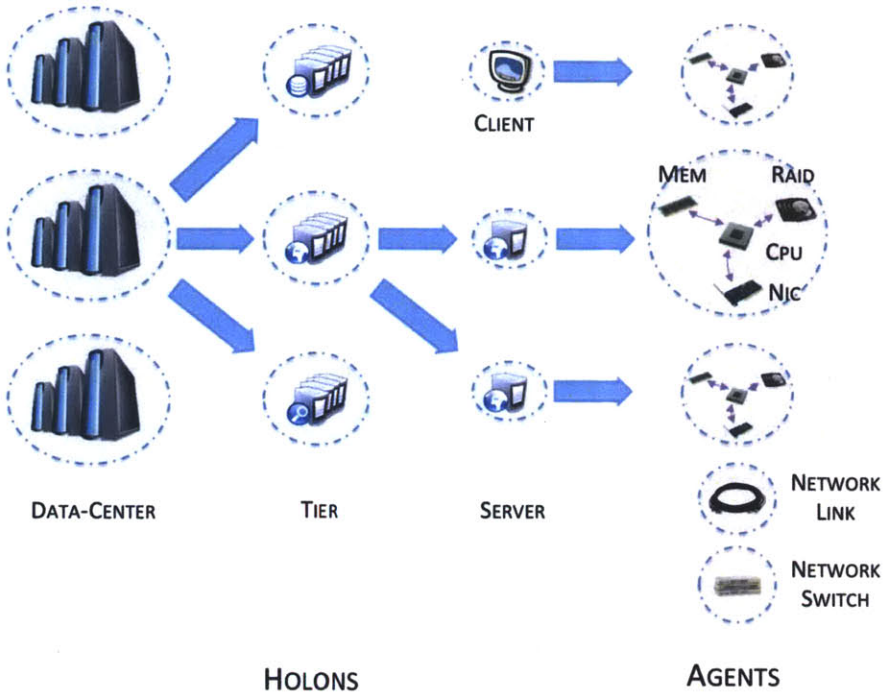

Figure 3-2: Holons and agents composing the Global Data Infrastructure Simulator.

Lowest level *Hardware Components* such as *memory*, *CPU*, *disk arrays*,*network cards*, *switches* and *links* are represented by agents of different types. Each agent has an internal state and a predefined behavior. The internal state of an agent is influenced by incoming messages from other agents, and based on this state it produces messages addressed to other agents. Agents of the same type, can be assigned different parameters, which determine their specific behavior. These parameters are established during the creation of the instance of an agent. For example, two agents of the CPU type can have different behaviors, since one may have been specified as a dual-core processor and the other as a quad-core processor each with different clock frequencies.

Agents representing low-level hardware components are encapsulated into a *server* or *client* holon. The state of the server or client holons is the composition of the internal states of the agents they encapsulate and their behavior follows the combination of their individual behaviors. The *tier* holon is an array of identical server holons

interconnected by network switch and network link agents. Tier holons can be of different types and cover different responsibilities, e. g. application tier, database tier or file server tier, based on the specifications of the holons and agents that form them. Similarly, the *data center* holon interconnects multiple tiers using network switch and network link agents. The interconnection of data centers defines the global infrastructure to be simulated.

Software applications are modeled as collections of sequences of messages called *Software Operations*. Each message within an operation, specifies the details of a relationship between two holons. The message is characterized by an array of hardware agnostic parameters, $\bar{R}$, that encapsulates information on the computational (R_p), network (R_t), memory (R_m) and disk cost (R_d) of the relationship. Each of these relationships between holons is decomposed into a series of interactions with the agents composing them at both ends. An interaction with an agent utilizes one or more parameters of $\bar{R}$ to alter its internal state. Changes in the state may trigger interactions addressed to other agents in the holon. The duration of each agent interaction is registered and the cumulative time for all the interactions yields the time elapsed to process the message. Similarly, the cumulative time for all the messages in the sequence produces the total elapsed time for the operation.

Messages follow the $m_{A \rightarrow B}^{X \rightarrow Y}$ notation, where $A \rightarrow B$ specifies the holons involved in the transaction, A for the origin and B for the destination holon. $X \rightarrow Y$ indicate the data centers in which these holons are located, $A \in X$ and $B \in Y$. Figure 3-3 illustrates the sequence of messages of a standard *Login* operation. This operation is decomposed into only two messages, an outbound message from a client located in Europe, C^{EU}, to an application server in North America, S_{app}^{NA}, and the inbound message, from S_{app}^{NA} to C^{EU}. Each message contains a different parameter array $\bar{R}$.

3.4 Data Center Hardware Modeling

This section presents the models utilized to represent data center hardware. Lowest level hardware components (agents) are modeled using queues or networks of queues.

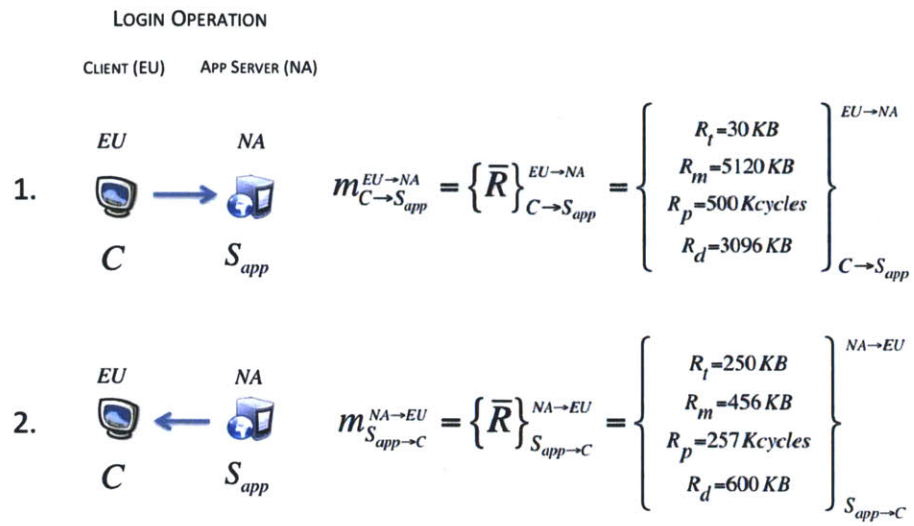

Figure 3-3: Messages and parameters composing the Login operation.

The interconnection of multiple low level components yields higher level entities such as servers (holon). Similarly, interconnected servers create tiers (holon) and interconnected tiers create data centers (holon).

3.4.1 Queueing Networks

Networks of queues are systems containing an arbitrary, but finite, number m of queues. Customers travel through the network and are served at the nodes. The state of a network can be described by a vector, where k_i is the number of customers at queue i.

The complexity of the majority of real-world systems cannot be handled by the mathematical models in classical queueing theory. An alternative means of analysis is the use of simulation of these queueing networks. Next, the queueing network models for computer hardware components used in the simulator are explained.

3.4.2 Queuing Network Models for Hardware Components

In this section, the queueing network model for each low-level hardware component is presented and explained. These queueing networks models are the basic building

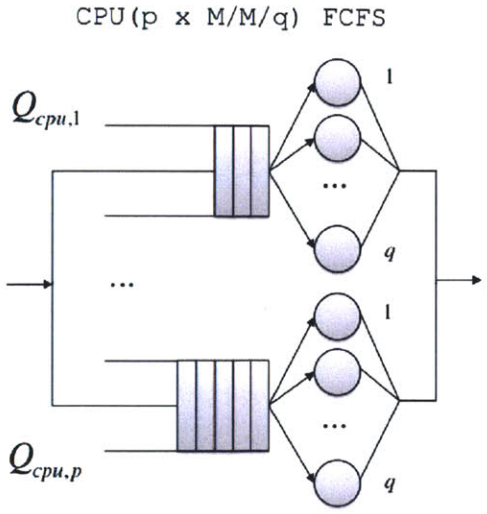

Figure 3-4: Queueing network model for a multi-socket multi-core CPU (p- socket, q-core).

blocks of the data center hardware model, and their implementation is the pillar of the simulation platform.

Central Processing Unit (CPU)

Multi-socket multi-core CPUs are modeled using p x $M/M/q$ FCFS queues, $Q_{cpu,i}$ with $i = 1 \ldots p$ [79]. This is illustrated in Figure 3-4. Each task enqueued in the CPU model contains the number of cycles to be consumed in the queue-server. Upon consumption the task is released.

The technical specifications of the CPU are utilized to specify the parameters of the CPU model. p is determined by the number of sockets, q is established by the number of cores and the service rate of the queue-servers (number of cycles consumed per second) is given by the frequency of each core in GHz. Hyper-threading effects can be included by increasing the number of cores by a factor based on the speedup measured empirically.

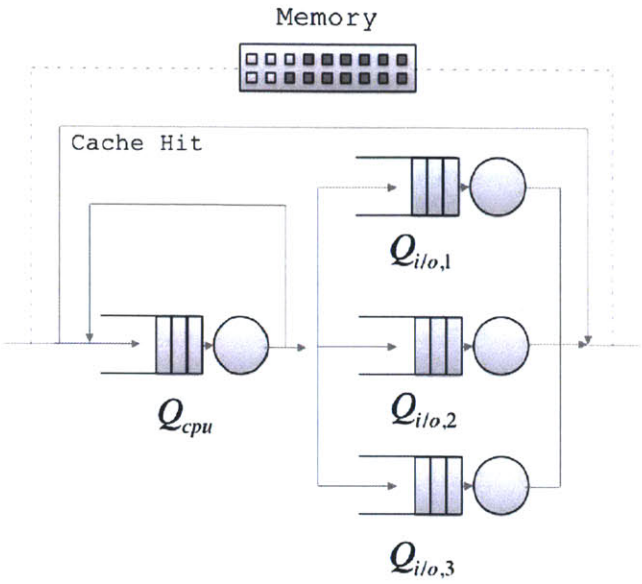

Figure 3-5: Queueing network model including the memory caching and occupancy effects.

Memory

The memory is the only component not modeled as a queue, and addresses two different effects (Figure 3-5): *Memory Caching* and *Memory Occupancy* [14]. A cache hit is modeled by bypassing the subsequent queues without requiring any processing in them. The occupancy is represented by the allocation of an established amount of memory for the duration of the processing in the CPU and I/O queues.

The specifications of the memory component are the size in GB and the cache hit rate, which should be obtained based on empirical results.

Network Interface Card (NIC) and Network Switch

Network Interface Cards and Network Switches are modeled using $M/M/1 - FCFS$ queues, denominated Q_{nic} and Q_{sw} respectively [71]. Both are illustrated in Figure 3-6 (left and center). Tasks are enqueued in the network card or network switch models and convey the number of bits to be processed by the queue-server.

The service rate (number of bits processed per second) is determined by the speed

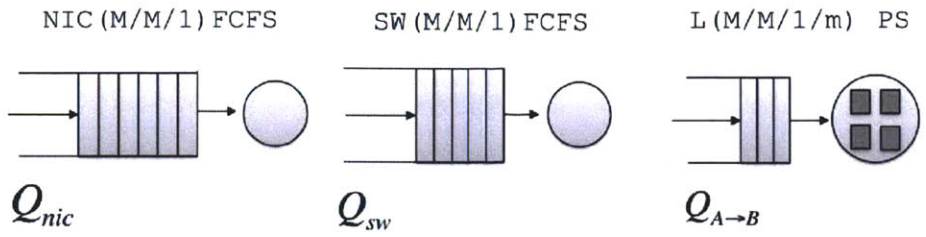

Figure 3-6: Queueing network models for the Network Inteface Card (Left), Network Switch (Center) and Network Link (Right).

in Mbps of the NIC or Network Switch, typically the NIC is an order of magnitude slower than the network switch.

Network Links

Network links are modeled using $M/M/1 - PS_k$ queues, denominated $Q_{A \to B}$ [71]. This is illustrated in Figure 3-6 (right). Each task enqueued in the network link model contains the number of bits to be processed by the queue-server. As opposed to $FCFS$ queues, PS_k queues can process up to a maximum of k tasks simultaneously. k is given by the number of simultaneous connections allowed for the link.

The service rate (number of bits processed per second) is determined by the bandwidth in Mbps of the link and is distributed uniformly among the number of tasks simultaneously being processed. The latency in milliseconds is a constant value that depends on the link characteristics added to the processing time of each task.

Redundant Array of Identical Disks (RAID)

Each disk is modeled as a sequence of two queues: Q_{dcc} representing the disk controller cache and Q_{hdd} representing the disk drive. A cache hit in Q_{dcc} can be modeled by bypassing Q_{hdd}. A RAID with n disks is modeled using an n fork-join structure of Q_{dcc}-Q_{hdd} queues preceded by a disk array controller cache Q_{dacc} [86]. A cache hit in Q_{dacc} is modeled by bypassing the fork-join structure. This is illustrated in Figure 3-7

The service rate of Q_{dacc} is given by the speed of the disk array controller in Gbps

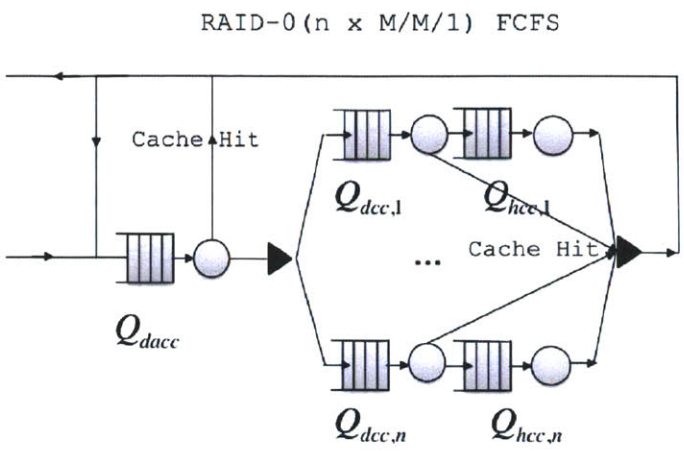

Figure 3-7: Queueing network model for a Redundant Array of Identical Disks (RAID).

and the cache hit rate is a tunable parameter that must be set based on empirical measurements. Similarly, the service rate of $Q_{dcc,i}$, with $i = 1 \ldots n$ is given by the speed of the disk controller in Gbps and a tuneable cache hit rate. The service rate of Q_{hdd} is given by the disk drive speed in MB/s.

Storage Area Network (SAN)

Analogous to RAIDs, Storage Area Networks (SAN) are also modeled using an n fork-join structure of Q_{dcc}-Q_{hdd} queues. As opposed to RAID models in which the fork-join queue was preceded only by a Q_{dacc}, in SANs the fork-join structure is preceded by three queues: a fiber channel switch Q_{fc-sw}, a disk array controller cache Q_{dacc} and a fiber channel arbitrated loop Q_{fc-al}. A cache hit in Q_{dacc} can be modeled by bypassing Q_{fc-al} and the fork-join structure. This is illustrated in Figure 3-8

The parameters of the fork-join structure and Q_{dacc} are set analogous to the RAID model. The service rate of Q_{fc-sw} is given by the fiber channel switch speed in Gbps. Similarly, the service rate of Q_{fc-al} is given by the speed of the fiber channel arbitrated loop in Gbps.

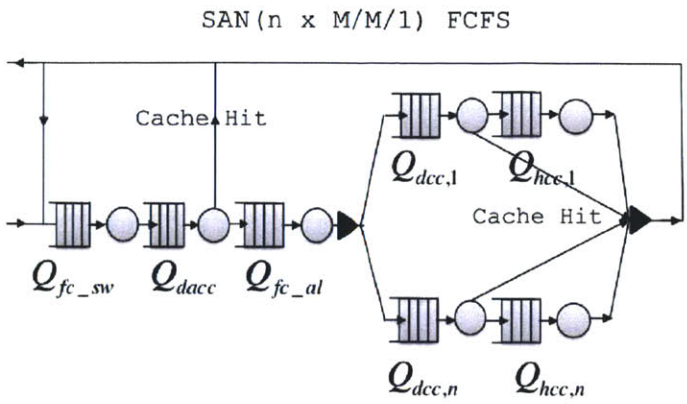

Figure 3-8: Queueing network models for a Storage Area Network (SAN).

3.4.3 Data Center Model using Queuing Networks

The interconnection of the low-level hardware components introduced in the previous sections produces the *Data Center Model*. The data center can be formed by an arbitrary number of tiers, with each tier containing an arbitrary number of servers. Typically, servers have different hardware specifications. Figure 3-9 illustrates an example of a small data center for explanation purposes. The derivation of systems of higher complexity is trivial following this example.

This data center is located in North America and contains two tiers, application tier T_{app} and database tier T_{db}, connected through a switch Q_{sw}. The data center is accessed by local clients in North America C_i^{NA} with $i = 1 \ldots 10$ through link $L^{NA \rightarrow NA}$ and Europe C_j^{EU} with $j = 1 \ldots 10$ through link $L^{EU \rightarrow NA}$.

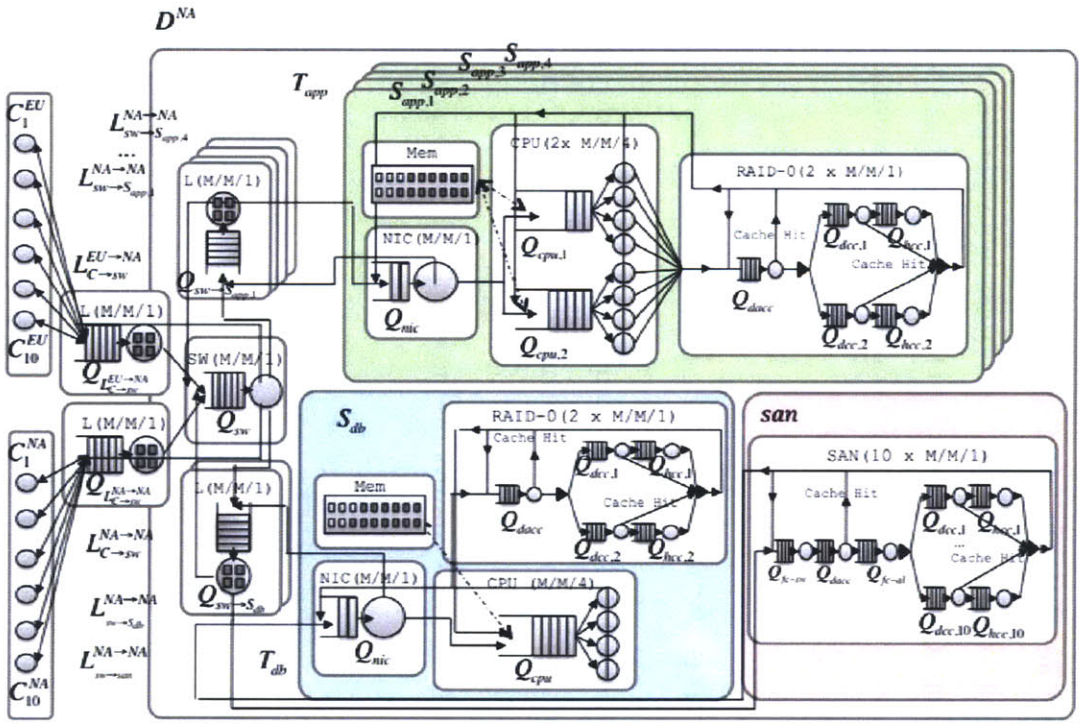

Figure 3-9: Data center model constructed by interconnecting component models.

- $\mathbf{T_{app}}$ is composed by four servers $S_{app,i}$ with $i = 1 \ldots 4$. Each $S_{app,i}$ contains a network card, Q_{nic}, a dual socket quad-core CPU, $(Q_{cpu,1}, Q_{cpu,2})$, and a RAID, Q_{dacc}, $(Q_{dcc,i}, Q_{hdd,i})$ with $i = 1 \ldots 2$. The servers are connected to the network switch through local network links $L^{NA \to NA}_{sw \to S_{app,i}}$.

- $\mathbf{T_{db}}$ is composed by a single server S_{db} and a *san*. S_{db} has a network card, Q_{nic}, a quad-core CPU, Q_{cpu}, and an identical RAID. *san* is formed by a fiber channel switch, Q_{fc-sw}, a disk array controller and cache, Q_{dacc}, a fiber channel arbitrated loop, Q_{fc-al} and an array of disks, $(Q_{dcc,i}, Q_{hdd,i})$ with $i = 1 \ldots 10$. The database servers and SAN are connected to the network switch through $L^{NA \to NA}_{sw \to S_{db}}$ and $L^{NA \to NA}_{sw \to san}$ respectively.

3.5 Software Application Modeling

As introduced in Section 3.3.2, software applications are modeled as collections of client-initiated operations. Typical operations are Login, Search, Open or Save. These operations are represented as sequences of messages, in which each message encodes an array of parameters $\bar{R}$ that conveys the impact of each message as it flows through computer resources.

In order to fully characterize a software application, two data types will have to be provided to the simulator: 1) The application workload. 2) The message tree that defines each operation of the application.

3.5.1 Application Workload

The application workload registers the number of clients that launch an operation by location and time of the day. It also provides information about the distribution of the operation types and their fluctuation throughout the day.

An example of the hourly workload for Application X is shown in Figure 3-10 (left) and the hourly operation distribution in Figure 3-10 (right). The population of clients in NA using Application X ramps up from 8 am to 10 am Eastern Time

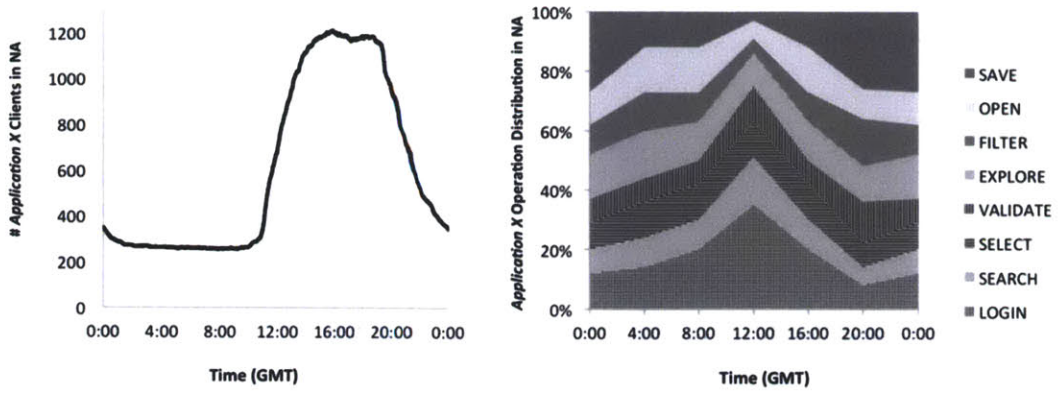

Figure 3-10: Application X workload (left) and operation distribution (right).

(12:00 to 14:00 GMT) growing from 600 users to approximately 1200. The operation distribution indicates that the largest fraction of users are logging in or searching for files at this time of the day, while a marginal fraction is saving data.

Conversely, the population of clients in NA using Application X is reduced from 3 pm to 5 pm Eastern Time (19:00 to 21:00 GMT). The operation distribution indicates that in this period of time the largest fraction of users are saving, opening and filtering files, while a reduced fraction is logging into the system or doing search.

3.5.2 Operation Modeling using a Message Cascade

In this research, each operation is modeled as a collection of sequences of messages that is generated when a client request is served by the infrastructure. Since sequences are initiated by clients in a sequential order, we refer to the collection as a *Message Cascade*. As explained in Section 3.3.2, each message in the sequences represents a relationship between components in the system and encapsulates information about the associated resource allocation and processing cost through the parameter array $\bar{R}$. The message cascade dictates the types of holons involved in each transaction, however, the exact data center, server and hardware instances are decided at run-time by the simulator, based on the input workload and predefined load-balancing strategies.

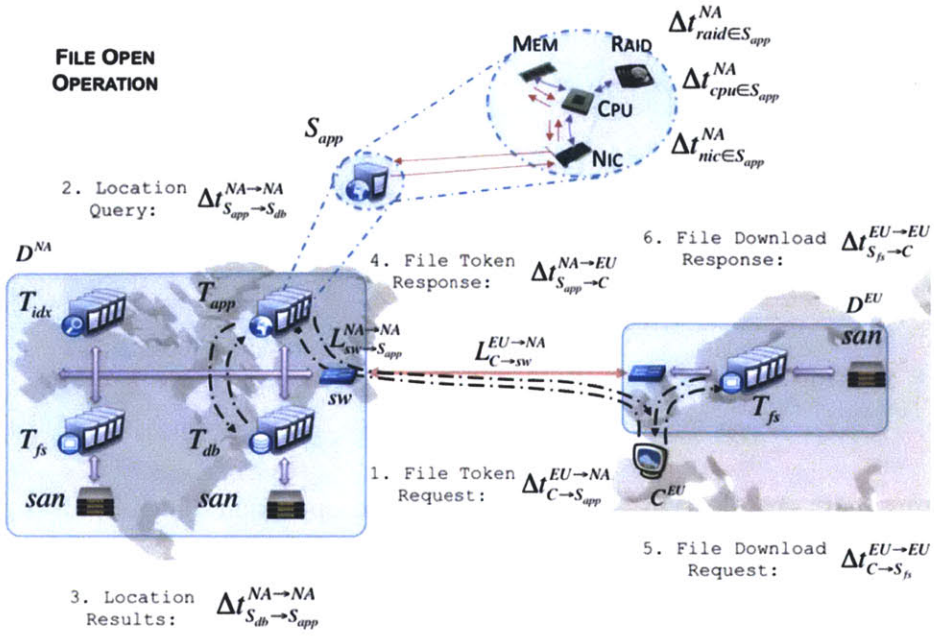

Figure 3-11: Decomposition of the file open operation into messages.

Figure 3-11 illustrates the decomposition of a file open (OPEN) operation into a message cascade. First, the client C^{EU} in Europe makes a request to a server $S_{app} \in T_{app}$ in the data center in North America, D^{NA} for the token needed to download the latest version of a file from a server $S_{fs} \in T_{fs}$ in D^{EU}. The S_{app} checks for metadata about this file in a server $S_{db} \in T_{db}$ to make sure the $T_{fs} \in D^{EU}$ has indeed the latest version, if not, a synchronization request between the $T_{fs} \in D^{NA}$ and the $T_{fs} \in D^{EU}$ would have been triggered through the network link connecting D^{EU} and D^{NA}. Upon token reception by C^{EU}, the token is used to download the file directly from a server $S_{fs} \in T_{fs}$ in D^{EU}.

Figure 3-12 illustrates the message cascade for an OPEN operation. A *Segment* is defined as the sequence of messages m that is originated and finalized in the client C. Each message points to the next message to be processed. This message cascade is composed of two segments. Segment (1) represents the client query to $S_{db} \in T_{db}$ via $S_{app} \in T_{app}$ to obtain a token to download the latest version of the desired file. Using this token, Segment (2) represents the download of the desired file from $S_{fs} \in T_{fs}$.

Next, in order to facilitate the explanation of the response time decomposition,

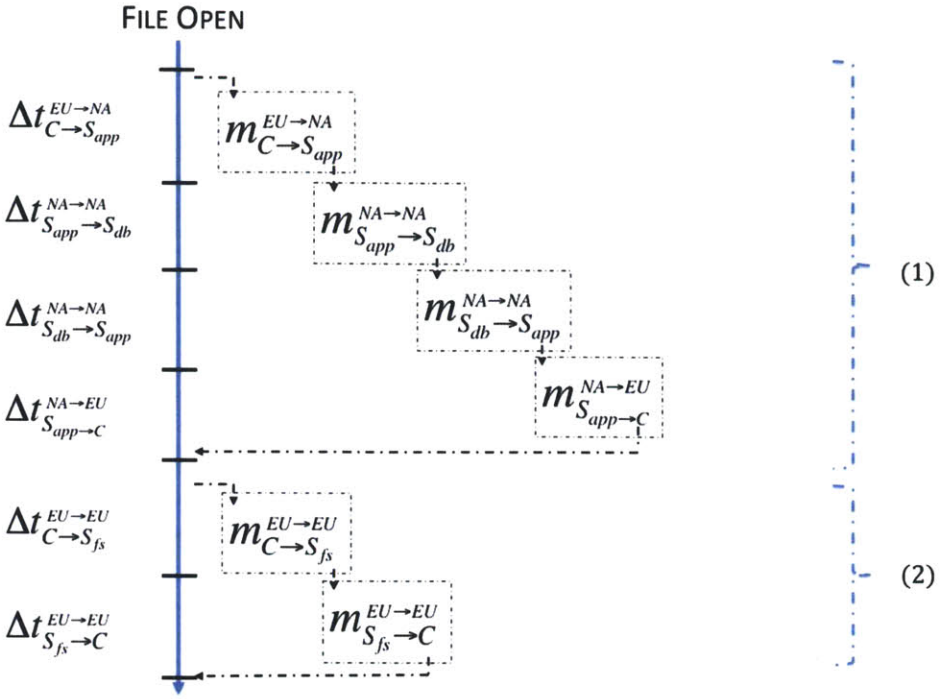

Figure 3-12: Message cascade representation of the file open operation.

the following notation is introduced:

- $\mathbf{\Delta t_A^X}$. A specifies the holon or agent measured and X indicates the data center that the holon or agent belongs to.
- $\mathbf{\Delta t_{A \to B}^{X \to Y}}$. $A \to B$ specifies the holons involved in the transaction, A for the origin and B for the destination holon. $X \to Y$ indicates the data center in which these holons are located, $A \in X$ and $B \in Y$.

The total response time of the OPEN operation is calculated by adding the time measured at each step, as shown in Equation 3.1.

$$T_{Open} = \Delta t_{C \to S_{app}}^{EU \to NA} + \Delta t_{S_{app} \to S_{db}}^{NA \to NA} + \Delta t_{S_{db} \to S_{app}}^{NA \to NA} + \Delta t_{S_{app} \to C}^{NA \to EU} + \Delta t_{C \to S_{fs}}^{EU \to EU} + \Delta t_{S_{fs} \to C}^{EU \to EU} \quad (3.1)$$

Equation 3.2 shows an example on how the timings for the first message of the OPEN operation are calculated. The overall duration for a $\Delta t_{C \to S_{app}}^{EU \to NA}$ message is decomposed into the time elapsed in each step: the time elapsed in the origin holon

Δt_C^{EU}, the transfer time $\Delta t_L^{EU \to NA}$ and the time elapsed at the destination holon $\Delta t_{S_{app}}^{NA}$.

$$\Delta t_{C \to S_{app}}^{EU \to NA}\left(\bar{R}\right) = \Delta t_C^{EU}\left(\bar{R}\right) + \Delta t_L^{EU \to NA}\left(\bar{R}\right) + \Delta t_{S_{app}}^{NA}\left(\bar{R}\right) \tag{3.2}$$

Similarly, the duration of each step can be further decomposed into the time elapsed in each agent. For example, Δt_C^{EU} and $\Delta t_{S_{app}}^{NA}$ are decomposed into the timings measured at the network card, CPU and disk array agents as shown in Equations 3.3 and 3.4.

$$\Delta t_C^{EU}\left(\bar{R}\right) = \Delta t_{nic \in C}^{EU}\left(R_t\right) + \Delta t_{cpu \in C}^{EU}\left(R_m, R_p\right) + \Delta t_{raid \in C}^{EU}\left(R_d\right) \tag{3.3}$$

$$\Delta t_{S_{app}}^{NA}\left(\bar{R}\right) = \Delta t_{nic \in S_{app}}^{NA}\left(R_t\right) + \Delta t_{cpu \in S_{app}}^{NA}\left(R_m, R_p\right) + \Delta t_{raid \in S_{app}}^{NA}\left(R_d\right) \tag{3.4}$$

Equation 3.5 makes an analogous decomposition for the data transfer across the network link $L_{C \to sw}^{EU \to NA}$, network switch sw and local network link $L_{sw \to S_{app}}^{NA \to NA}$ agents.

$$\Delta t_L^{EU \to NA}\left(\bar{R}\right) = \Delta t_{L_{C \to sw}}^{EU \to NA}\left(R_t\right) + \Delta t_{sw}^{NA}\left(R_t\right) + \Delta t_{L_{sw \to S_{app}}}^{NA \to NA}\left(R_t\right) \tag{3.5}$$

$\bar{R}$ Parameter Array Profiling

Every message in a cascade conveys a different $\bar{R}$ parameter array. The agents that compose the destination holon of the message utilize one or more parameters of this array to reproduce the desired interaction by performing work in their internal queues.

The parameter array $\bar{R}$ for each message is obtained by profiling the *canonical cost* of each operation. As introduced in Section 3.2, the execution of a single isolated operation of each type in the infrastructure yields the canonical cost. The canonical cost is defined as the computational, network, disk and memory cost incurred by a single user running the real software on the infrastructure. The fine-grained measurement of these costs in each element of the system delivers the values that populate the

parameter array $\bar{R}$ for each message that composes the operation.

As opposed to continuous profiling mechanisms which can be out of reach for many organizations due to their high cost, the calculation of the $\bar{R}$ parameter array through profiling of each operation and software application in the infrastructure is an inexpensive one-time task that is affordable for any company with a global IT infrastructure.

3.6 Summary

In this chapter, the concepts on top of which the foundation of GDISim is constructed have been introduced. Using hardware specifications, application workloads and profiling measurements, the simulator uses a Holonic Multi-Agent System (HMAS) to reproduce the behavior of the infrastructure and return estimates on hardware utilization, user experience and overall effectiveness. The HMAS is composed of holons and agents that represent the different layers of hardware in the infrastructure, from low-level components such as CPU or disk drives, to entire data centers, connected through agents representing network switches and links. Additionally, the queueing network models utilized to simulate the behavior of each agent type, along with their structured interconnection to produce servers, tiers and data centers is explained. Finally, the message cascade model utilized to represent the execution of client operations as part of a distributed software applications is covered. These message cascades govern the nature of the interactions between the agents and holons populating GDISim.

Chapter 4

Simulation Platform Design & Implementation

4.1 Introduction

In this chapter the details of the implementation of the Global Data Infrastructure Simulator (GDISim) are presented. This chapter focuses on the design and implementation concepts that enable the Holonic Multi-Agent System simulator to accommodate an arbitrary number of infrastructure components, and consequently, large numbers of agents and a greater number of queues. The high degree of parallelism inherent to Multi-Agent Systems was considered in the early stages of the design of the simulator, which allowed exploiting multithreading on a shared memory multiprocessor architectures.

The simulator was constructed using the C# programming language under the .NET environment. Nevertheless, the design and implementation concepts detailed in this chapter are programming language agnostic and the simulator could be reproduced on any other language and runtime environment.

4.2 Asynchronous Messaging

The simulation platform is designed to exploit parallelism by using *asynchronous messaging* for the processing of agents in the system and their interactions. As opposed to synchronous message passing systems, in which the sender and the receiver wait for each other to transfer the message and the sender waits until the message is received by the receiver, asynchronous message passing delivers the message from sender to receiver without waiting for the receiver to be ready. The advantage of asynchronous messaging is that enables sender and receiver to overlap computation and not block each other.

In addition to asynchronous messaging, the simulator is built upon the *Active Message* mechanism that is introduced in Section 4.2.1. This mechanism intends to expose the full hardware flexibility and performance of modern interconnection networks. On top of this mechanism, the simulator uses an abstraction called *Port-based Programming* to hide the low level concepts of active messaging and presents port objects as inputs for the manipulations of agent states. The port object is covered in Section 4.2.2. Finally, the combination of multiple forms of port objects allows constructing *Coordination Primitives* that orchestrate high volumes of asynchronous messages efficiently as described in Section 4.2.3.

4.2.1 Active Messages

Active messages are asynchronous messaging objects capable of performing processing on their own. As opposed to traditional messaging systems in which messages are passive, active messages convey the necessary information to carry out operations with the data they transport. Following this idea, each message contains the address of the handler to be executed on arrival using the message payload as argument [88].

Active message handlers do not have their own execution context, and are executed on the stack of the thread that pulled the active message from the dispatcher queue of the message receiver. Therefore, no stack is allocated and no thread switch is needed to run the handler [57]. With these characteristics active messages deliver increased

performance. Unfortunately, the main caveat is that if an active message handler blocks, the thread cannot be resumed because the handler occupies part of its stack. For this reason, active messages prohibit blocking, and locks or condition variables are not allowed, and thus, alternative mechanisms must be used.

4.2.2 Port-Based Programming

The *Port* concept was first utilized as an automaton to model concurrent processes. Steenstrup et al. [82] utilized ports as the only points of entry to stateful processes. These ports received messages that invoked handler functions to manipulate the internal state of the process using the data transmitted within the message. Upon termination, these processes could post the outcome of the operation to ports corresponding to other processes. Stewart et al. [84] take this concept a step further and define *Port-Based Objects*. Port-based objects have the same properties as standard objects, including internal state, code and data encapsulation, and characterization by its method. The difference is that they communicate exclusively using ports, and their integration into the subsystem is carried out by connecting the output port of a module to the corresponding input of the next module. The construction of systems based on port-based objects is denoted as *Port-Based Programming*. Dixon et al. [24] explore the utilization port-based objects to construct large-scale architectures and show the adaptability and flexibility of this programming model.

In the context of a MAS, processes are replaced by agents. Figure 4-1 illustrates the elements and the utilization of port-based abstraction in the simulator. Agents have an internal *state* and can have multiple input *ports*. Ports are strongly typed and have handlers associated to them. Upon receipt of a message on a port, the message payload along with the handler associated with the port are paired into a *work item* by the *arbiter*. Work items represent the concept of the active message mechanism presented in Section 4.2.1. These work items are submitted by the arbiter to a *dispatcher*, which is in charge of effectively executing work items using available resources. The dispatcher runs a *thread pool* that continuously pulls work items from the dispatcher queue. These threads invoke the handler using the accompanying

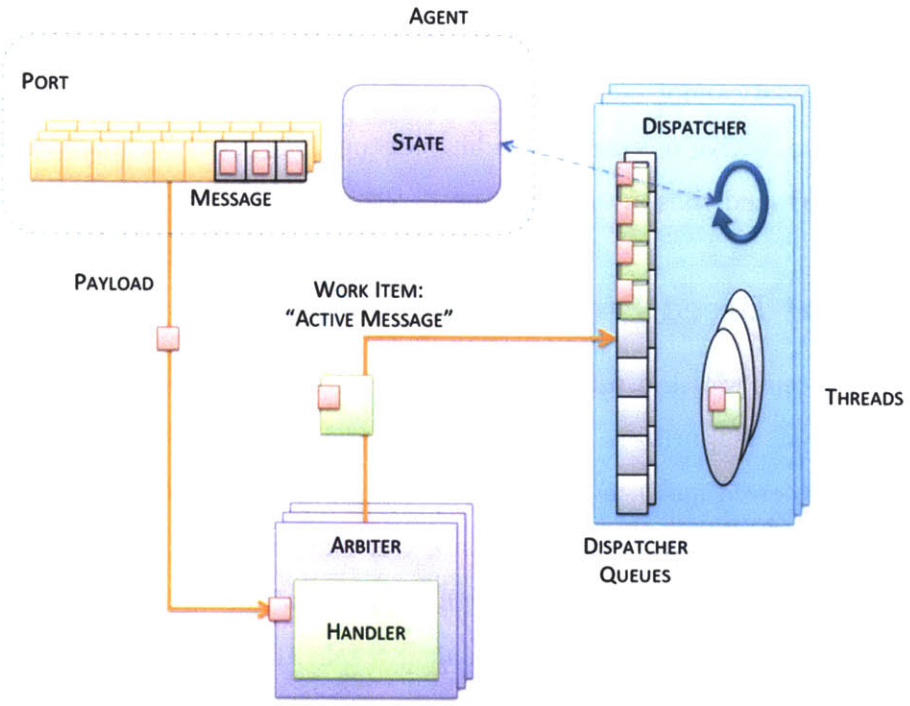

Figure 4-1: Elements composing the port abstraction in the simulator.

payload data, and possibly manipulate the internal state of the agent. A single dispatcher can be responsible of processing the handlers of one or multiple agents. Optionally, upon termination the result can be posted back to a port belonging to the originator of the message.

4.2.3 Coordination Primitives

Chrysanthakopoulos and Singh [18] describe the implementation of high-level coordination primitives using Port-based programming. They present a library called Concurrency and Coordination Runtime (CCR) that implements join patterns [26] and interleaved calculations from basic building blocks and that is designed to support large amounts of fine-grained concurrency. Coordination primitives not only facilitate the orchestration and synchronization of multiple concurrent tasks, but also enable dealing with failure. Additionally, they provide performance measurements to support the utilization of these coordination primitives at the application level.

As follows, the coordination primitives constructed using the elements introduced in Section 4.2.2 are presented:

- *Single Item Receiver*: Registers handler X to be launched when a single message of type M is received in Port A.

- *Multiple Item Receiver*: Registers handler X to be launched when n messages are received in Port A. p messages can be of type M (success) and q messages of exception type E (failures), so that $p + q = n$. The handler X is passed the payload of both types of messages M and E.

- *Join Receiver*: Registers handler X to be launched when one message of type M is received in Port A and another of the same type in Port B. The handler X is passed the payload of both messages.

- *Choice*: Registers handler X to be launched when one message of type M is received in Port A and registers handler Y to be launched when one message of type N is received in Port A. If invoked, handler X will receive the payload of message of type M, and handler Y the payload of message of type N.

- *Interleave*: Registers how handlers associated to port X execute in relation to each other and to their own parallel executions. Handlers belong to three groups: 1) *Teardown*: These handlers are executed one time and atomically. 2) *Exclusive*: These handlers only run when no other handler is running. 3) *Concurrent*: These handlers run in parallel with other invocations of themselves.

Using these primitives a *Scatter-Gather* mechanism was constructed in order to orchestrate thousands of agents running in the MAS. This is illustrated in figure 4-2 and is separated into two phases:

- *Scatter*: A message of type M is posted to each port in an array of ports of type A from a master thread. Each port of type A is registered with a Single-Item Receiver that invokes handler X. Handler X is part of a concurrent execution group. Included in the payload of each message of type M is a reference to a unique instance of a port of type B.

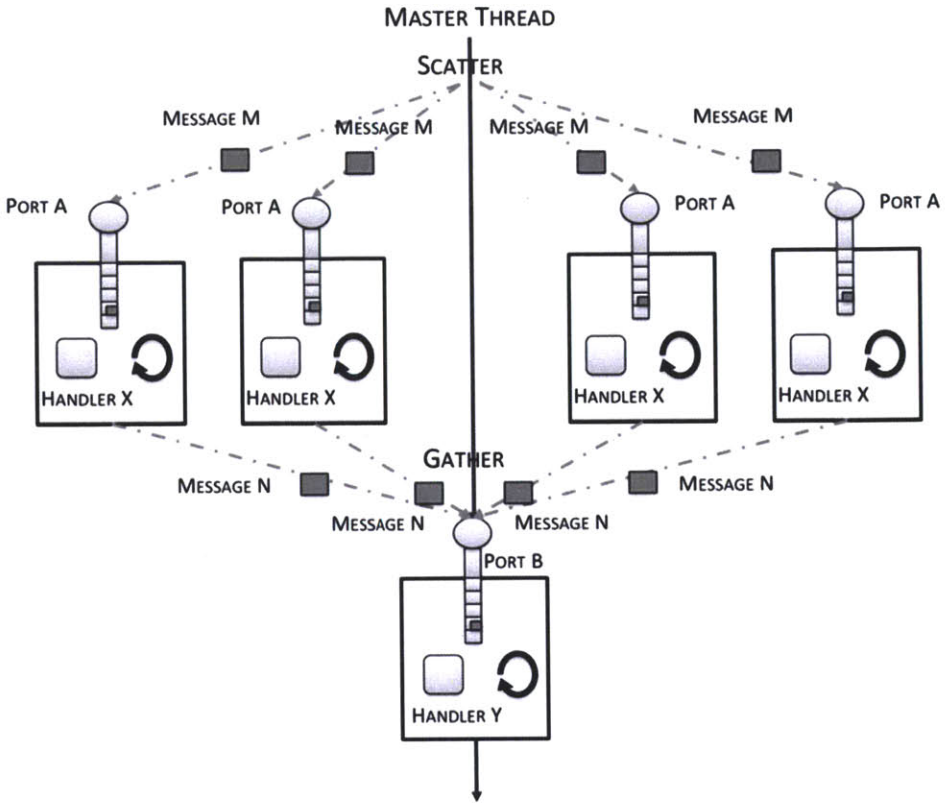

Figure 4-2: Diagram illustrating the construction of the Scatter-Gather mechanism using port-based programming.

- *Gather*: Each port of type A invokes handler X using the data included inside each instance of message M. Multiple handlers X execute concurrently and upon termination they post the results back inside messages of type N using the reference to port B. Port B is registered with a Multiple-Item Receiver that invokes hander Y in the master thread with an array of messages of type N or type E for the case of multiple failures.

4.3 Platform Implementation

Using the asynchronous messaging and coordination principles introduced in Section 4.2 as a foundation, next the details of the implementation of GDISim are presented. The complexity of the distributed computer systems being modeled and the potential

large number of agents involved requires paying particular attention to the scalability of the platform.

This section presents the process followed to optimize the performance on a shared memory multiprocessor architecture and the impact these decisions had on the elapsed simulation time.

4.3.1 Discrete Time Loop

In the core of the platform, a centralized *Timer Component* controls the simulation time. This component has the critical responsibility of ensuring that all agents are synchronized in time. At every time step, the timer behaves as a "heartbeat", it signals all the agents in the simulation and waits for their responses before proceeding to the next time step. The granularity of the time step is configurable by the simulator user. It is recommended to be at least one order of magnitude smaller than the time values measured in the canonical operation set.

Periodically, the state of all the agents in the simulation platform is measured and registered by the *Collector Component.* Once a representative number of samples have been gathered, the simulation averages the samples across this measurement set and generates a snapshot of the status of the infrastructure. In addition to agent state, the platform registers the duration of the operations finalized during measurement interval, and similarly, it averages the samples to provide a snapshot of the response times by operation and data center.

For example, using a time step granularity of one millisecond, the state of all the agents can be measured every 100 milliseconds and a snapshot containing the average value in one minute (600 samples) can be generated and reported to the data center operators. The number of operations finalized during this one minute interval is also reported to operators by type and location.

4.3.2 Agent Control Signals

Agents interact with each other using the information encoded within the messages exchanged by the holons. In addition to these interactions, agents also receive two types of control signals: 1) Time Increment control signal, and 2) Measurement Collection control signal.

Time Increment Control Signal

The time increment control signal is received from the Timer Component at every time step. Upon receipt of the signal, each agent reproduces the effect of time consumption by executing a *Time Increment Handler*. After the corresponding time is consumed, the agent acknowledges the completion back to the Timer Component. The next iteration of time increment control signals is initiated as soon as the acknowledgement signal from all the agents is received. Each agent contains a local timer that is incremented at each time step.

For example, in the case of the CPU component, based on the clock speed specifications, the corresponding number of cycles are consumed from the messages being processed by the queues. For the case of a network link, based on the bandwidth specification and number of concurrent connections, the corresponding number of KB are transmitted through the network.

Measurement Collection Control Signal

A measurement collection control signal is periodically interleaved after a predefined number of time increment control signals. The signal is sent by the *Collector Component* and returns a collection of samples with the state of each agent. The time increment loop is not resumed until all the samples of all the agents have been collected.

After a predefined number of collections have been carried out, the average sample is calculated to generate a snapshot of the infrastructure. This snapshot is reported to data center operators and is registered permanently, while intermediate samples

are dismissed.

4.3.3 Agent Interaction Signals

As described in Section 3.3.2, software operations are decomposed into messages exchanged by holons, that in turn, trigger interactions between agents. An agent interaction represents the execution of a communication or computation task that affects the state of an agent based on the values of the parameter array $\bar{R}$ and the agent type itself. A fraction of the processing is carried out at each time step and, upon termination, the agent interaction is pulled from the current agent and possibly a new interaction is forwarded to the next agent with the corresponding parameter set $\bar{R}$.

The asynchronous messaging nature of the HMAS can potentially lead to inconsistent states. For example, say agent a_0 receives a time increment control signal and moves from t_0 to t_1. During the transition from t_0 to t_1 an agent interaction, r_0, is finalized and the subsequent one, r_1, is forwarded to agent a_1. Nevertheless, agent a_1 has not received the time increment control signal yet and awaits in time t_0. Under these circumstances, if the queue at agent a_1 at time t_0 is empty, a_1 could process r_1 during its transition from t_0 to t_1. This situation would lead to an inconsistent state, since r_1 should not have been processed until $t \geq t_1$.

The simulator enforces a mechanism to check the timestamps of the interactions against the local time of each agent, so as to guarantee that an interaction r scheduled to be initiated at $t \geq t_1$ is not processed during $t_0 < t < t_1$.

4.3.4 Scatter-Gather Parallelization

The natural method to parallelize the execution of time steps and measurements in the HMAS is to use the *Scatter-Gather* coordination mechanism presented in Section 4.2.3. Each agent has three ports: 1) Time Increment port, 2) Measurement Collection port and 3) Interaction port.

During each time step, a time increment control signal message is posted to the

Time Increment port in each agent as part of the *Scatter* phase. The time control message is paired with the time increment handler into a task and enqueued to the dispatcher queue. Then any available thread from the thread pool invokes the handler using the time control message. A single dispatcher containing a dispatcher queue and a thread pool is shared across all the agents in the HMAS. The time increment control signal message conveys a reference to the timer component synchronization port. At termination, each agent posts an acknowledgement message to the synchronization port as part of the *Gather* step.

The execution of the measurement collection scatter-gather mechanism is analogous to the time increment, but using the corresponding messages, ports and handlers associated to the measurement collection process. By design, the measurement collection process is launched after a predefined number of time steps. Therefore there is no execution overlap between time increment handlers or measurement collection handlers in each agent, and consequently there is no need to set an exclusive interleave policy between them.

Conversely, the HMAS allows agent interaction messages to be posted concurrently with the execution of time increment handlers as part of the time increment loop. Since both the time increment handler and the interaction handler of an agent manipulate its state, an exclusive interleave policy must be established between them, in order to guarantee the protection against race conditions over the state of the agent. Parallelization of the HMAS through the scatter-gather mechanism is illustrated in Figure 4-3.

In order to measure the performance and scalability of the implementation of this platform, a series of simulation experiments were executed on a global infrastructure using different sizes of the thread pool in the dispatcher. The global infrastructure simulated is composed of six datacenters containing a total of 14 servers, 432 cores and 168 disks distributed across 14 RAIDs and seven SANs. Data centers are connected through 155 Mbps and 45 Mbps network links and seven network switches. Three software applications run on top of this infrastructure imposing a combined load of 6000 clients at the peak time of the day. Additionally, synchronization and indexing

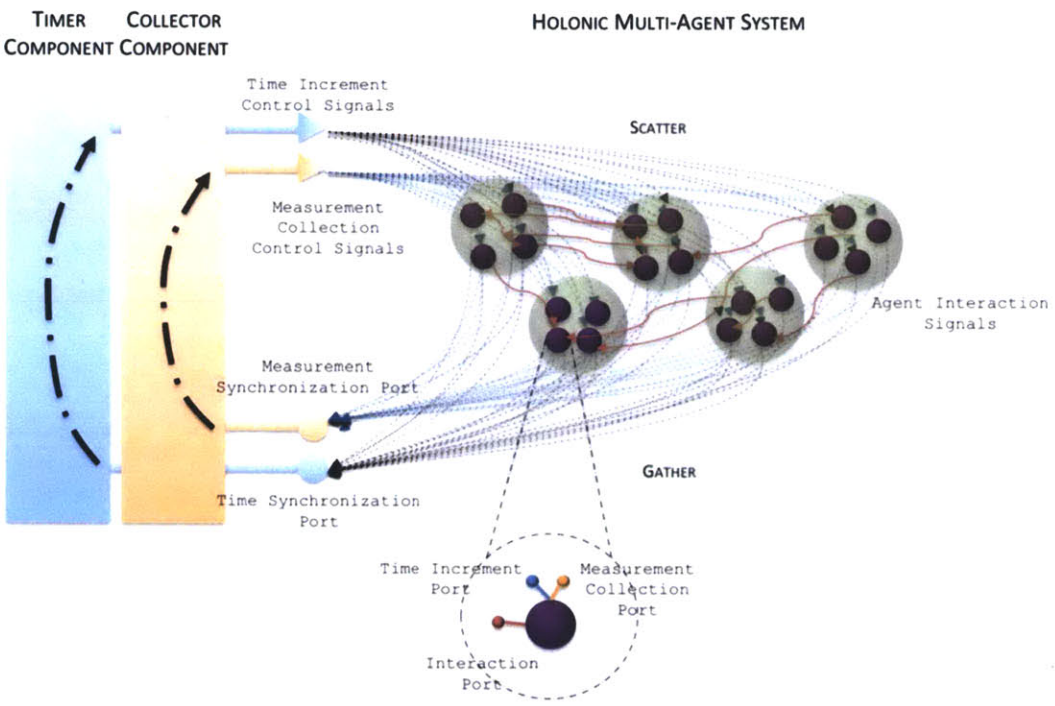

Figure 4-3: Scatter-Gather parallelization in the HMAS for the timer and collector components.

# of Threads	Simulation Time (min)	Speedup (x)
1	9888	1.00
2	9192	1.08
4	10440	0.95
8	10248	0.96
16	10056	0.98

Table 4.1: Simulation time (min) and speedup (x) vs. the # of threads in the thread pool for the classic Scatter-Gather mechanism.

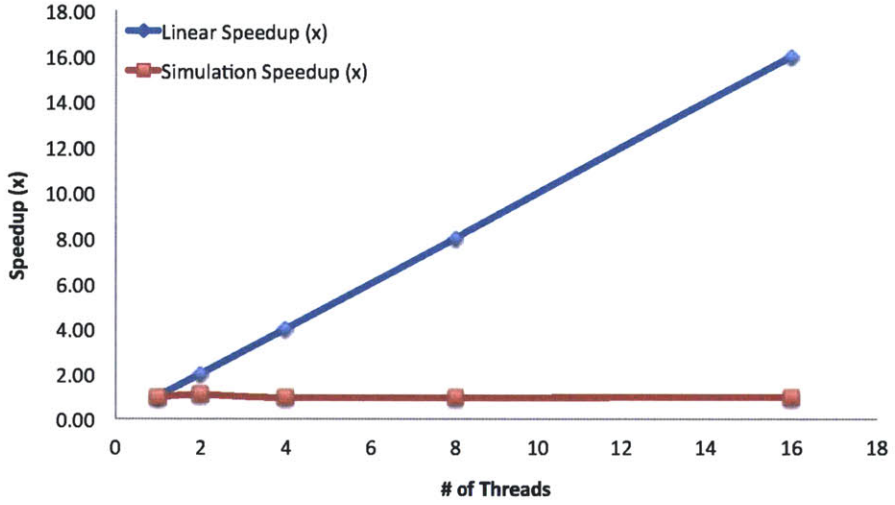

Figure 4-4: Multicore scalability of the classic Scatter-Gather mechanism vs. Linear scalability

processes are executed concurrently with these software applications. A detailed description of this global infrastructure is presented in Chapter 6 as part of the data center consolidation case study.

It can be observed in Table 4.1 and Figure 4-4 that using the scatter-gather mechanism to coordinate the execution of one thread per handler invocation does not provide any benefit on the performance of the simulation platform. The addition of worker threads (cores) to the dispatcher did not reduce the total simulation time. This effect illustrates that the work to be carried out within each thread as a consequence of a handler invocation is too small to justify the overhead of pairing the message and the handler and passing these asynchronously to the thread pool for execution.

4.3.5 H-Dispatch Model

Even though *a priori* the use of the scatter-gather mechanism to coordinate the execution of one thread per handler invocation is the natural method to parallelize the MAS, and consequently speedup the simulation time, the overhead cancelled this effect. In this section, a series of modifications of the scatter-gather mechanism are carried out, so as to push additional work to each thread in the thread pool and not to allow the computation to be hidden by the overhead. For this purpose, an adaptation of the *H-Dispatch* by Holmes et al. was used [41].

Holmes et al. point out that an important drawback of the scatter-gather model implemented with a managed memory language such as C#, is the poor memory efficiency and its implications on the CPU utilization. At each time step, a separate virtual thread is spawned to execute the handler associated to each agent, and during this process a significant amount of memory allocation is carried out. In managed memory languages, these allocations are collected and released by the Garbage Collection (GC) mechanism [63]. Unfortunately, the GC mechanism can prevent concurrent execution by blocking threads while the heap is being reordered and pointers updated.

The H-Dispatch model is presented as a solution to reduce memory utilization and maximize the efficiency of multithreaded execution. Holmes et al. propose to select as many worker threads as cores are available. These threads are always active and process items (in this case agents) sequentially, reusing local variable memory allocations and thus, eliminating the need for garbage collection. Additionally, load balancing is guaranteed by changing the *Push* nature of the classic scatter-gather to a *Pull* mechanism that makes worker threads to request work from a global queue called H-Dispatch queue. H-Dispatch ensures that worker threads are always busy until the global queue is empty.

Figure 4-5 illustrates the adaptation of the H-Dispatch model by Holmes et al. to GDISim. At every time step, a time increment message is posted to the port of each of the worker threads. As opposed to the classic scatter-gather model presented

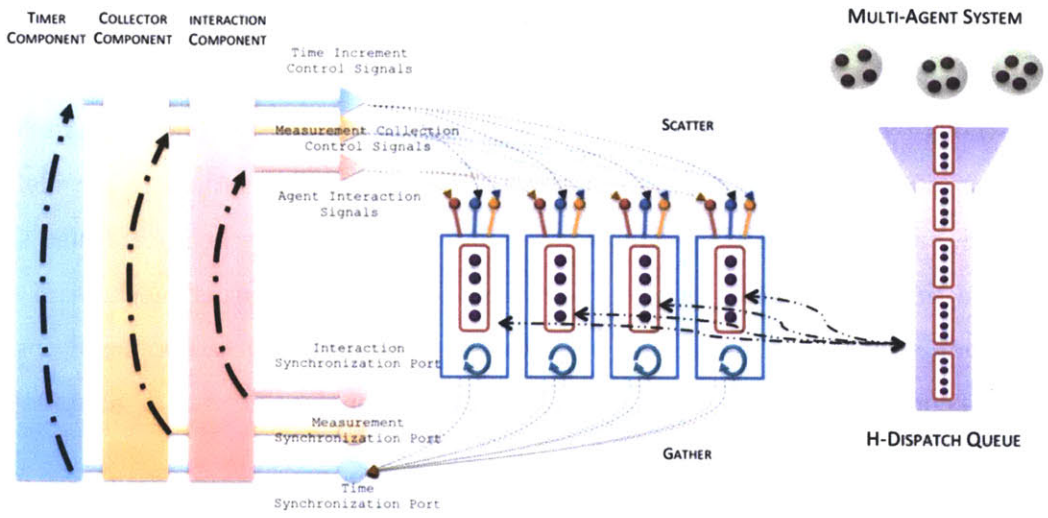

Figure 4-5: H-Dispatch parallelization in the MAS for the timer, collector and interaction components.

in Section 4.3.4 which used a virtual thread for every handler, the adaptation of H-Dispatch encapsulates multiple agents into an *Agent Set* that is passed to the worker thread for sequential execution. The type of handler executed in each agent of the Agent Set is based on the original message received by the worker thread, in this case, the time increment handler. Worker threads iteratively pull Agent Sets from the H-Dispatch queue until the queue is empty, then a time synchronization message is sent to the time synchronization port. Measurement collection follows an analogous procedure.

In the classic scatter-gather, agent interaction handlers were executed concurrently with time increment handlers. Nevertheless, the requirement to group multiple executions of agent interaction handlers for sequential execution in the H-Dispatch model, required to decouple the time increment and the agent interaction phases. In the H-Dispatch model, an Agent Interaction step will be triggered right after every time increment step. At the finalization of a time increment, potential agent interactions are registered for later execution during the agent interaction step.

It can be observed in Table 4.2 and Figure 4-6 that using the H-Dispatch model to coordinate the execution one thread for multiple handlers improved the multicore

# of Threads	Simulation Time (min)	Speedup (x)
1	10728	1.00
2	6278	1.71
4	3353	3.20
8	2074	5.17
16	1331	8.06

Table 4.2: Simulation time (min) and speedup (x) vs. the # of threads in the thread pool for the H-Dispatch mechanism (Agent Set=64).

scalability of the simulation platform. The increase in the number of worker threads utilized increased consistently the speedup to the simulator. An Agent Set of size 64 delivered the best results. Using 16 worker threads the simulation time was reduced from approximately 7.5 days to approximately 1 day. Even though these results present a considerable improvement from the classic scatter-gather approach, it can be observed that as the number of threads increases the efficiency of the multicore scalability drops from $\sim$ 80% with four worker threads to $\sim$ 50% with sixteen worker threads. The speedup evolution shown in Figure 4-6 it is expected to reach saturation. These efficiency results diverge from the efficiency measures provided by Holmes et al., since they demonstrated a 85% of efficiency with 24 threads on a Finite Difference (FD) simulation problem. Nevertheless, there are two critical factors that prevent GDISim from achieving these figures:

1. *Sequential Steps*: As opposed to the FD problem solved by Holmes et al. which only contained a single parallelizable computational step within the time loop, the infrastructure simulation problem intertwines three sequential steps (time update, measure collection and agent interaction).

2. *Locality*: As opposed to FD simulations the problem exposed by the infrastructure simulator does not contain a notion of spatial locality that can be cache optimized.

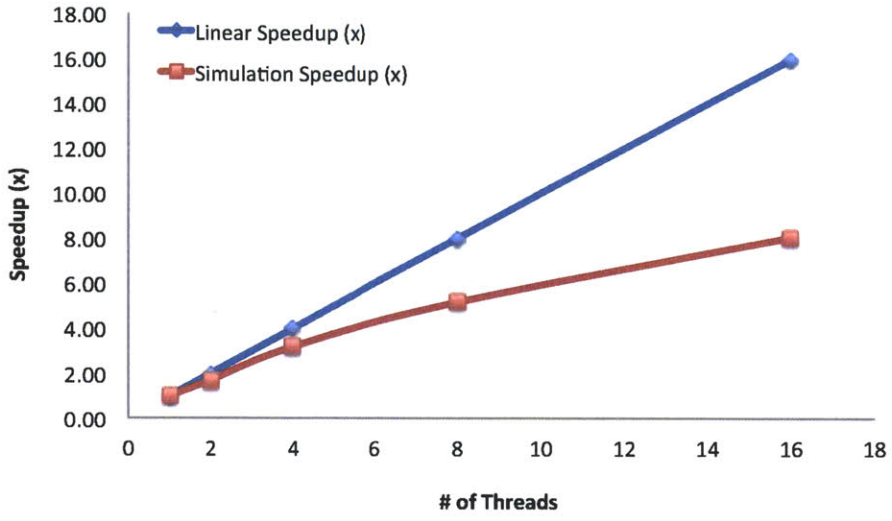

Figure 4-6: Multicore scalability of the H-Dispatch mechanism vs. Linear scalability. (Agent Set=64)

4.4 Summary

This section presented the design and implementation details of GDISim. First, the details and advantages of asynchronous messaging and active messages were introduced, along with the Port-based programming abstraction, which provides an elegant yet computationally efficient programming model for Holonic Multi-Agent Systems. Using Port-based programming as a foundation, a scatter-gather mechanism was constructed using coordination primitives as building blocks.

Next, details on the simulation platform implementation were provided. The platform is driven by three types of signals: Time control signals, Measurement collection signals and Agent interaction signals. These signals are distributed/collected to/from the agents and dictate the behavior of the HMAS. As a first approach, the simulation platform utilized the classic scatter-gather mechanism to orchestrate all the agents in the HMAS by using a single thread to execute each handler. Nevertheless, this approach showed that the mechanism overhead blurs the parallelization targets. Under these circumstances, an event-based algorithm for concurrent task distribution, H-Dispatch, was introduced. The adaptation of the H-Dispatch mechanism, by Holmes

et al., fixed the number of worker threads to the number of cores dedicated to the simulator, reused local variables preventing the garbage collector from blocking the threads and guaranteed load balancing of the computational work. The benefits were reflected in a considerable speedup for the simulator reducing the execution time by almost an order of magnitude.

Chapter 5

Simulation Platform Validation

5.1 Introduction

This chapter presents the methodology utilized to validate the data center infrastructure model described in Chapter 3. The goal is to compare side by side the behavior of a real IT infrastructure in a Fortune 500 company (or a downscaled version of it) running real software applications with its simulated counterpart using GDISim. Both experiments collect measurements on hardware utilization, network occupancy and operation response times, so as to study the divergence between them and provide an assessment of the accuracy of the queueing network model utilized to construct GDISim.

This chapter has three main sections. The first presents the approach utilized to validate the infrastructure model, including detailed information about the data center, details on the application workload and a list of assumptions that must to be considered. Next the outputs of both the real system and the simulator are presented, focusing on computational, memory and network utilization measurements, in addition to operation response times. Finally, the results are gathered side by side and the accuracy of the estimations provided by the simulator are evaluated.

Throughout this chapter, *Physical* infrastructure refers to the real (downscaled) IT infrastructure of the Fortune 500 company, and *Simulated* infrastructure to the simulated version executed by GDISim.

5.2 Validation Approach

This section presents the methodology utilized to validate the data center infrastructure model and the software application model presented in Chapter 3. First, the specifications of the physical infrastructure used for validation are described along with details of the software application and the synthetic workload generated to feed the real system. Next, the canonical operations measured from the physical system and the same synthetic workload are passed as inputs to the simulator modeling the behavior of the infrastructure. Finally, the experimental setup and critical assumptions that put the validation process in context are covered.

5.2.1 Downscaled Infrastructure

For validation purposes, a downscaled version of the physical IT infrastructure in a Fortune 500 company was utilized. This infrastructure is illustrated in Figure 5-1 and consists of a single data center in North America, D^{NA} serving a population of local clients, C^{NA}:

- *Tiers*: The data center is comprised of four server tiers: the application server tier $T_{app}^{(2,4,102)}$, the database server tier $T_{db}^{(1,4,64)}$, the file server tier $T_{fs}^{(1,4,12)}$ and the index server tier $T_{idx}^{(1,4,64)}$. The superscript in $T^{(a,b,c)}$ indicates the number of servers a, the number of cores per server b and the memory per server c in GB.

- *SANs*: T_{fs} and T_{db} are connected to two identical SANs, $san^{(1,20,15K)}$. The superscript in $san^{(a,b,c)}$ indicates the number of SAN servers s, the number of disks b and the speed of each disk c in rpm.

- *Network Links*: There are two types of network links, one interconnecting server tiers, $L^{(1,45)}$, and another one connecting tiers with SANs, $L^{(4,0.5)}$. The superscript in $L^{(a,b)}$ indicates the bandwidth a of the link in Gbps and the latency b in ms.

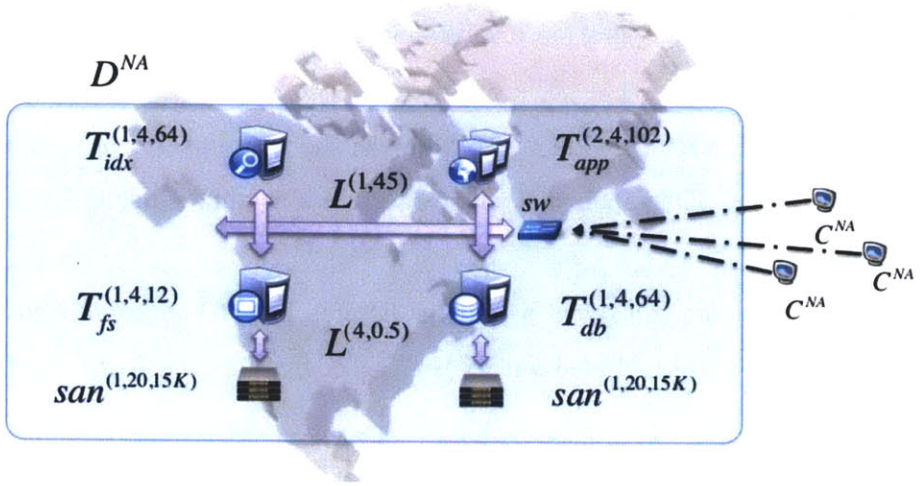

Figure 5-1: Physical (Downscaled) IT infrastructure in a Fortune 500 company utilized for validation of the hardware and software models.

5.2.2 Synthetic Workload

The software that the Fortune 500 company executed for validation purposes on this IT infrastructure is a Computer-Aided Design (CAD) application. CAD software is used in the process of design and documentation of products, and allows multiple designers and engineers to work on different parts of a product concurrently and collaboratively. Popular examples of collaborative CAD software are *CATIA* [20], *AutoCAD* [7] and *Creo Elements/Pro* [70].

The CAD software utilized for the validation experiments is decomposed into eight client, C^{NA}, initiated operations:

1. **LOGIN**: Clients present their credentials to T_{app} and this grants access allowing them to launch other operations.

2. **TEXT-SEARCH**: Clients search for design parts using the text-based search feature. This is done by querying an index file previously created by T_{idx} and that is hosted by T_{app}.

3. **FILTER**: Clients filter search results by introducing additional parameters to the text-search terms submitted to T_{app}.

4. **EXPLORE**: Clients analyze the relationships of a design part by navigating a tree structure. This operation involves metadata queries between T_{app} and T_{db}.

5. **SPATIAL-SEARCH**: Clients can analyze the relationships of a design part by navigating a 3D snapshot of this part and its neighbors using T_{idx}.

6. **SELECT**: Clients can select a specific 3D area in a 3D snapshot and retrieve the list of the parts included within that space by querying T_{db} through T_{app}.

7. **OPEN**: Clients can open a part (selected either via text or spatial search) and get the full detail of the model. This requires querying T_{db} to make sure that the part is available in T_{fs}. Then the file is downloaded directly from T_{fs}.

8. **SAVE**: Clients can save changes carried out in the model. Analogous to the OPEN operation, before the file upload a T_{db} update is required.

A *Series* is defined, as a sequential concatenation of these operations preserving the order in which they have been introduced above. Three different types of series were used for validation: *Light*, *Average* and *Heavy*. The name of these series corresponds to the volume of information manipulated by the operations that compose them. *Light* series manipulates small file sizes, while *Heavy* series manipulates large file sizes. Table 5.1 presents the timings for series type and operation type. It can be observed that the timings for the first six operations (LOGIN, TEXT-SEARCH, FILTER, EXPLORE, SPATIAL-SEARCH and SELECT) are very similar across different series types. On the contrary, the last two operations (OPEN and SAVE) have substantial differences across series types. The manipulation of larger file sizes does not affect the duration of the first six operations, primarily because these operate on metadata, and metadata is not affected by the size of the file. Nevertheless, the last two operations directly operate on (i.e. read and write) the files and therefore their duration is proportional to their size.

Operation Name	Light Series Duration (s)	Average Series Duration (s)	Heavy Series Duration (s)
LOGIN	1.94	2.2	2.35
TEXT-SEARCH	4.9	5.11	4.99
FILTER	2.89	2.6	3
EXPLORE	6.6	6.43	5.92
SPATIAL-SEARCH	12.18	12.15	12.38
SELECT	5.7	6.2	5.34
OPEN	30.67	64.68	96.48
SAVE	36.8	78.21	113.01
TOTAL	**101.68**	**177.58**	**243.47**

Table 5.1: Duration of the operations by type and series.

5.2.3 Message Cascades for CAD Operations

Figures 5-2 through 5-5 illustrate the message sequences for the eight CAD operations utilized in the validation experiments. Different operations can share the same sequences or the entire message cascade, but they differ on the parameter array $\bar{R}$ associated to each message. For example, OPEN and SAVE operations in Figure 5-5 illustrate identical message cascades, but the variations in the parameter array $\bar{R}$ of each message make SAVE approximately 20% more expensive.

The parameter array $\bar{R}$ for each message m in each operation is obtained from a detailed decomposition of the canonical cost of the operation. The canonical cost is obtained by launching CAD operations individually using the real software on the infrastructure presented in Section 5.2.1 and measuring the computational, memory, disk and network cost in every component at every step of the operation.

5.2.4 Experiments & Assumptions

The validation was carried out running three separate experiments. These experiments utilize the three series types (light, average and heavy) defined in Section 5.2.2. Each experiment indicates the frequency of launch of each series type. For example, during Experiment-1 (15-36-60), one light series is launched every 15 seconds, one average series every 36 seconds and one heavy series every minute.

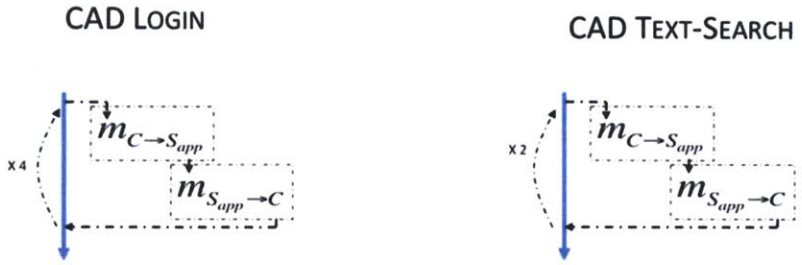

Figure 5-2: Message cascades for CAD LOGIN and TEXT-SEARCH operations.

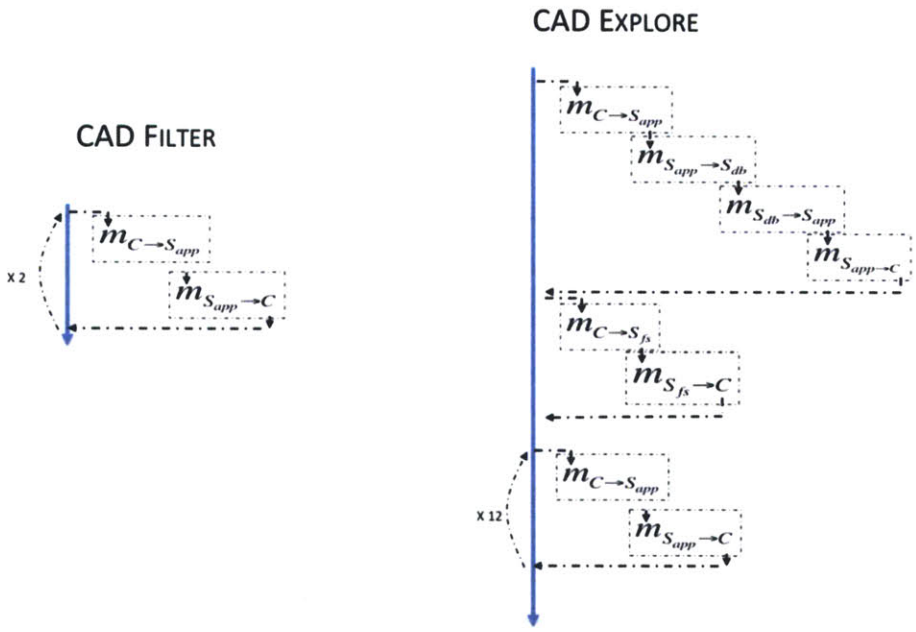

Figure 5-3: Message cascades for CAD FILTER and EXPLORE operations.

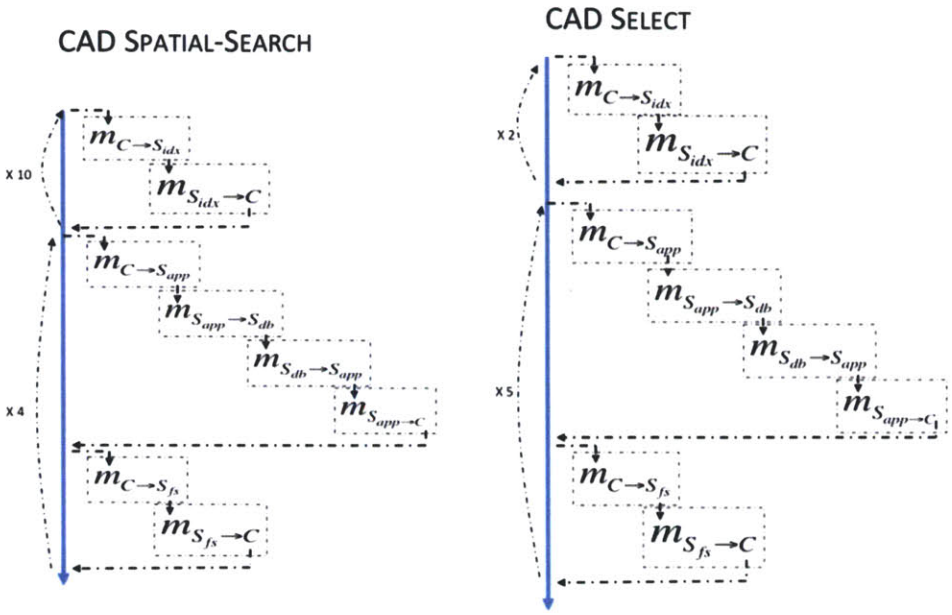

Figure 5-4: Message cascades for CAD SPATIAL-SEARCH and SELECT operations.

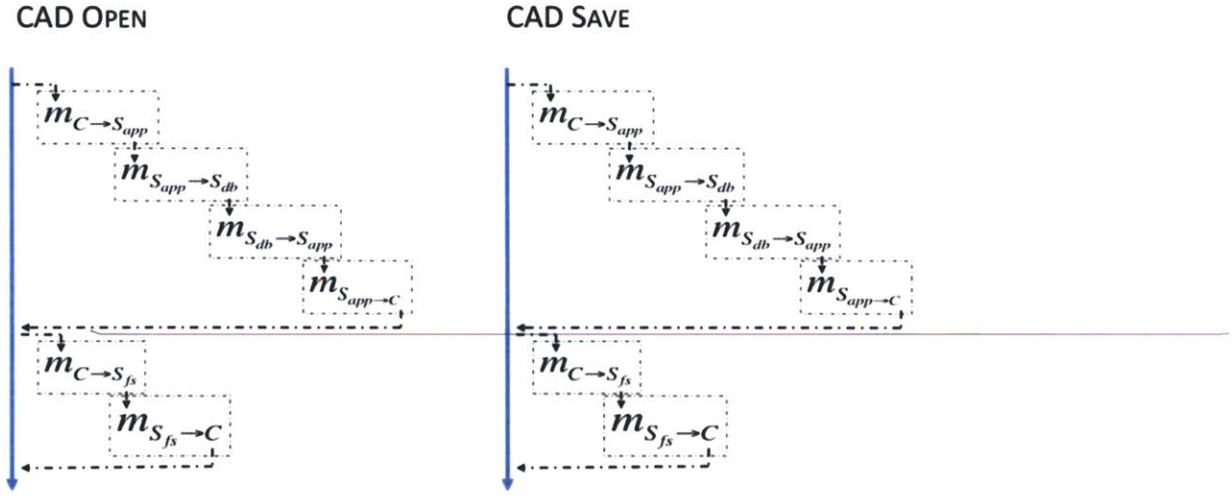

Figure 5-5: Message cascades for CAD OPEN and SAVE operations.

1. **Experiment-1 (15-36-60)**: 15s (Light) - 36s (Average) - 60s (Heavy)

2. **Experiment-2 (12-29-48)**: 12s (Light) - 29s (Average) - 48s (Heavy)

3. **Experiment-3 (10-24-40)**: 10s (Light) - 24s (Average) - 40s (Heavy)

All the frequencies chosen for the experiments are shorter than the duration of the shortest series, therefore, overlap between series is expected. When more than one series overlap, multiple messages compete for resources in the infrastructure and this is reflected in the utilization of the hardware simulated. Experiment-1 specifies the largest periods of time between series and imposes the least pressure on the infrastructure. On the contrary, Experiment-3 sets the shortest periods between series, and hence, puts the highest pressure on the system.

For each experiment, a "cold" start of the infrastructure is assumed. Each series is considered to be initiated by a different client. Clients do not have local copies of the files requested (local cache empty), and hence, a file download is always required. No caching between tiers of the data center is allowed. No background jobs were executed concurrently with the experiments in neither the physical infrastructure nor the simulated infrastructure.

Each experiment has three phases, namely, Initial Transient State, Steady State, and Final Transient State. The duration of each experiment is established by setting the duration of the Steady State to 31 minutes. In the three cases the duration of the entire experiment resulted to be approximately 38 minutes. Data collection for the comparison between the physical and the simulated infrastructures was performed by sampling all the component states in both systems every six seconds.

The goal of the validation experiments was to compare both systems within the linear operation zone, without reaching the saturation of any of the resources. Saturation of resources leads to nonlinear behaviors, which is beyond the scope of this research.

5.3 Simulation Result Evaluation

In this section the results of the three experiments defined in Section 5.2.4 in the physical and simulated infrastructure are directly compared. First, the number of concurrent clients running in both systems is compared. Next, the CPU utilization in application (T_{app}), database (T_{db}), file (T_{fs}) and index (T_{idx}) server tiers is compared, along with the corresponding memory measurements. Finally, the accuracy results are summarized and compared against the results presented by previous work on evaluation of computer systems.

5.3.1 Concurrent Client Validation

The number of concurrent clients in D^{NA} is equivalent to the number of series under execution that overlap in each experiment. Since both systems are fed with the same workload and both are operating within the linear operation zone, it is expected that the number of concurrent clients predicted using GDISim will follow closely the numbers measured in the physical infrastructure.

Figure 5-6 illustrates the evolution of the number of concurrent clients in the physical and simulated infrastructures for each of the three experiments (1-2-3). It can be observed that Experiment-1 results in approximately 22 concurrent clients in steady state, while Experiment-3 imposes the most pressure on the system with approximately 35 clients in steady state.

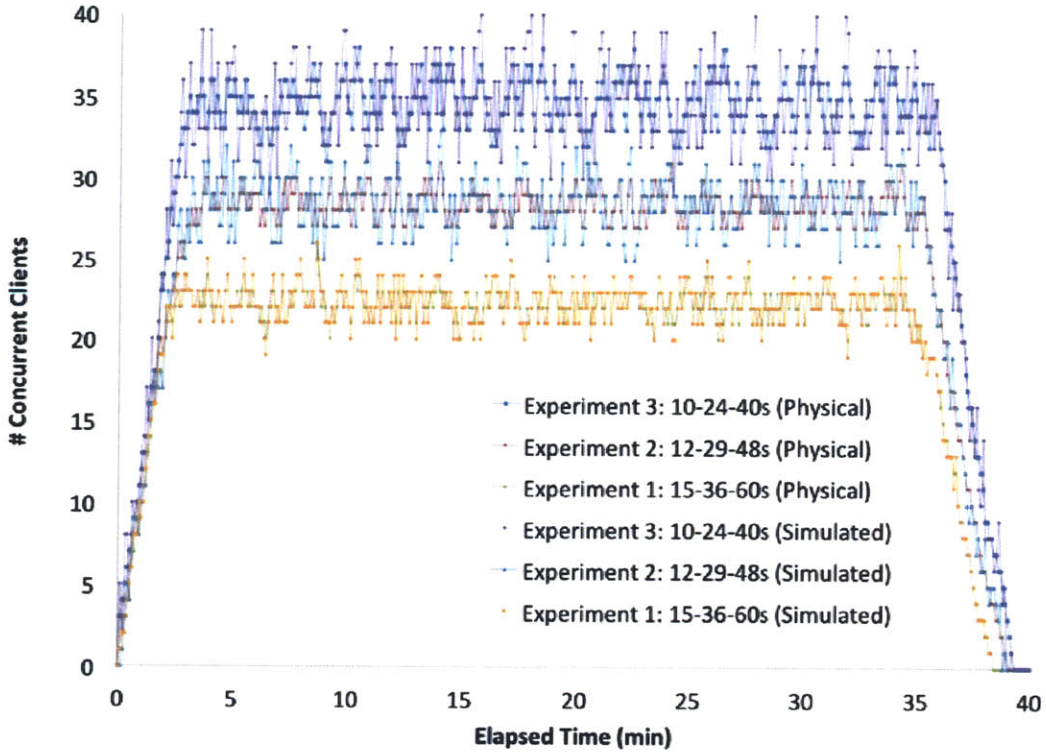

Figure 5-6: Comparison between the number of concurrent clients by experiment in the physical and simulated infrastructures.

5.3.2 CPU

Figures 5-7 to 5-10 illustrate the CPU utilization measurements for the T_{app}, T_{db}, T_{fs} and T_{idx} tiers in the three experiments (1-2-3) for both the physical infrastructure and the simulated infrastructure. The three phases (Initial Transient, Steady State and Final Transient) mentioned in Section 5.2.4 are noticeable in the three experiments for the four tiers measured.

The CPU utilization of a tier x at time sample t_n during experiment y is obtained by averaging the utilization of all the cores across the servers that compose that tier. The value measured from the physical infrastructure is represented by $P^y_{T_x}(t_n)$, and its simulated counterpart by $\widehat{P}^y_{T_x}(t_n)$. The average utilization and standard deviation in the steady state phase for tier x and experiment y are represented by $\mu_{P^y_{T_x}}$ (Equation 5.1) and $\sigma_{P^y_{T_x}}$ (Equation 5.2) for the physical infrastructure, and $\mu_{\widehat{P}^y_{T_x}}$ (Equation 5.3) and $\sigma_{\widehat{P}^y_{T_x}}$ (Equation 5.4) for the simulated infrastructure, respectively.

These results are summarized in Table 5.2. N_s is the number of samples taken during the steady state phase starting at t_0 and ending at t_{N_s-1}.

$$\mu_{P^y_{T_x}} = \frac{1}{N_s} \sum_{t_i=t_0}^{t_i=t_{N_s-1}} P^y_{T_x}(t_i) \tag{5.1}$$

$$\sigma_{P^y_{T_x}} = \sqrt{\frac{1}{N_s} \sum_{t_i=t_0}^{t_i=t_{N_s-1}} \left(P^y_{T_x}(t_i) - \mu_{P^y_{T_x}}\right)^2} \tag{5.2}$$

$$\mu_{\widehat{P}^y_{T_x}} = \frac{1}{N_s} \sum_{t_i=t_0}^{t_i=t_{N_s-1}} \widehat{P}^y_{T_x}(t_i) \tag{5.3}$$

$$\sigma_{\widehat{P}^y_{T_x}} = \sqrt{\frac{1}{N_s} \sum_{t_i=t_0}^{t_i=t_{N_s-1}} \left(\widehat{P}^y_{T_x}(t_i) - \mu_{\widehat{P}^y_{T_x}}\right)^2} \tag{5.4}$$

As expected, measurements consistently show that Experiment-3 imposes the highest pressure on the infrastructure during the steady state, while Experiment-1 the least. This effect is present in all the tiers of the infrastructure. In general, the average CPU utilization predicted for the steady state phase follows the average

measured in the physical infrastructure, with a maximum error of approximately 6% for the database tier T_{db}. The standard deviation of the predicted CPU utilization in steady state follows closely, with a maximum value of approximately 15% for the file server tier T_{fs}. In general, predicted values result in a slightly higher standard deviation than the values measured during physical experiments.

	Experiment-1				Experiment-2				Experiment-3			
	$\mu_{P^y_{T_x}}$	$\mu_{\widehat{P}^y_{T_x}}$	$\sigma_{P^y_{T_x}}$	$\sigma_{\widehat{P}^y_{T_x}}$	$\mu_{P^y_{T_x}}$	$\mu_{\widehat{P}^y_{T_x}}$	$\sigma_{P^y_{T_x}}$	$\sigma_{\widehat{P}^y_{T_x}}$	$\mu_{P^y_{T_x}}$	$\mu_{\widehat{P}^y_{T_x}}$	$\sigma_{P^y_{T_x}}$	$\sigma_{\widehat{P}^y_{T_x}}$
T_{app}	55.84	58.59	4.27	5.71	71.60	72.80	5.64	6.68	81.81	79.80	4.79	7.18
T_{db}	39.04	43.07	4.54	5.76	49.20	54.98	4.61	5.48	57.20	62.83	6.30	7.82
T_{fs}	40.60	42.93	10.87	11.26	49.87	48.63	10.66	10.98	56.68	52.55	12.06	14.70
T_{idx}	19.04	19.91	4.34	5.06	29.20	28.87	4.61	5.22	36.99	33.03	6.43	7.92

Table 5.2: $\mu_{P^y_{T_x}}$, $\mu_{\widehat{P}^y_{T_x}}$, $\sigma_{P^y_{T_x}}$ and $\sigma_{\widehat{P}^y_{T_x}}$ by experiment and measurement.

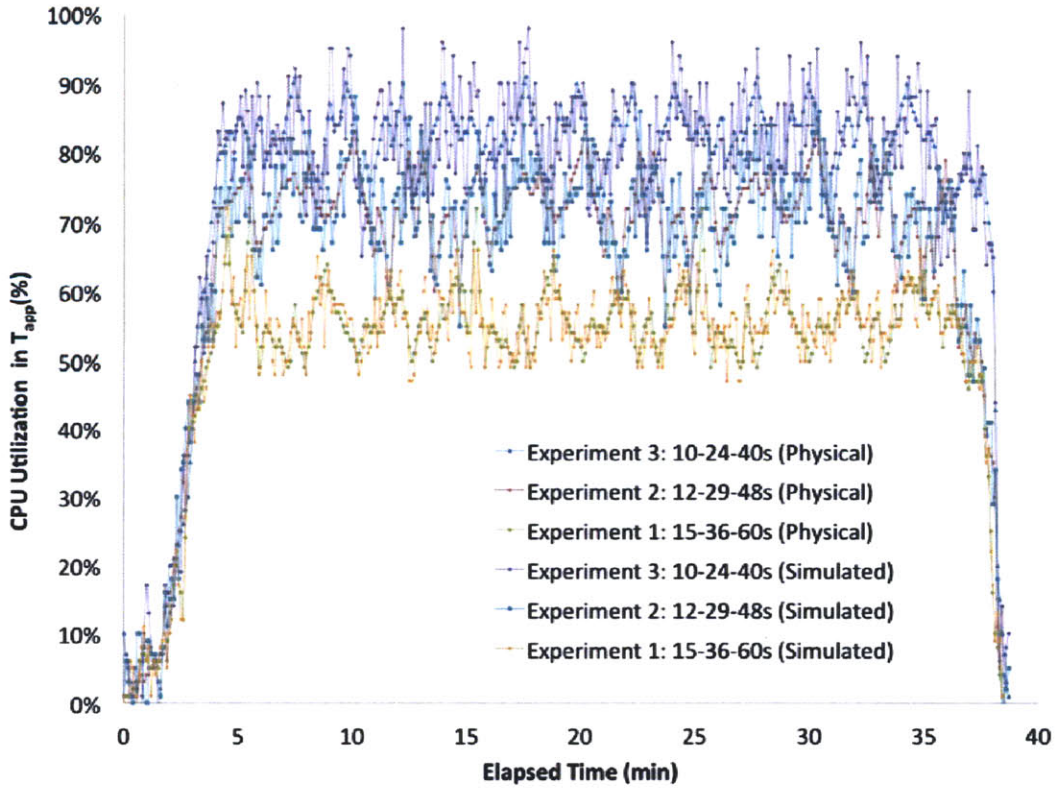

Figure 5-7: Comparison between CPU utilization in T_{app} by experiment in the physical and simulated infrastructures.

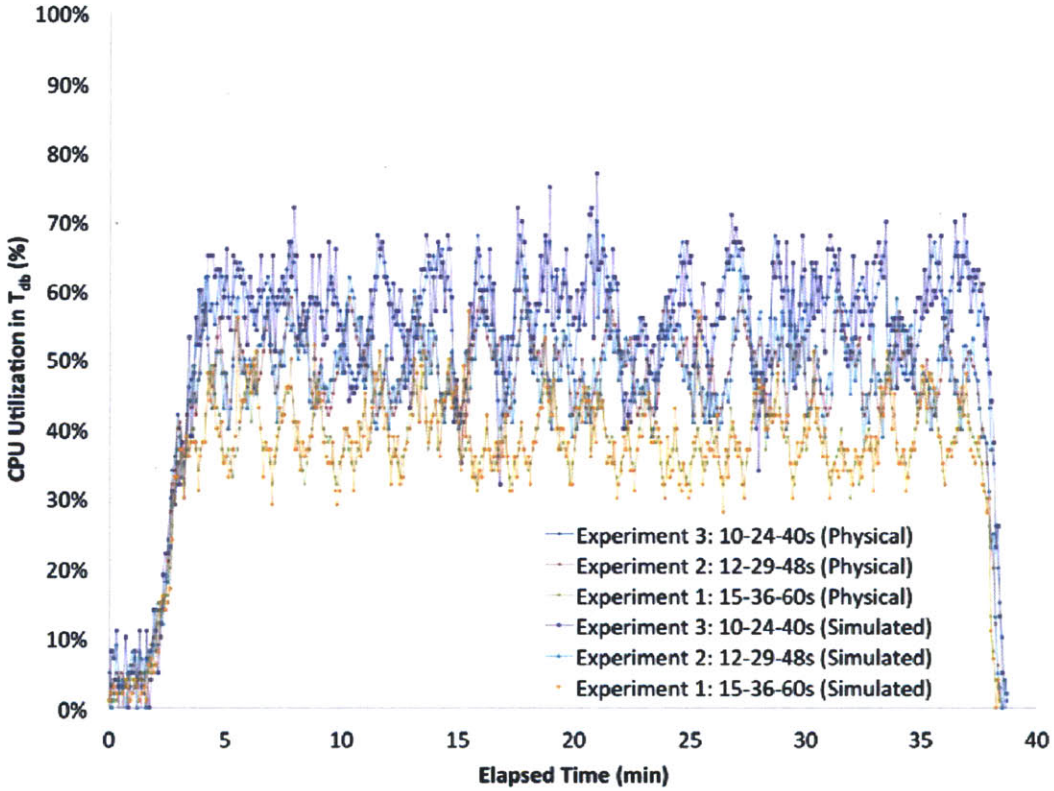

Figure 5-8: Comparison between CPU utilization in T_{db} by experiment in the physical and simulated infrastructures.

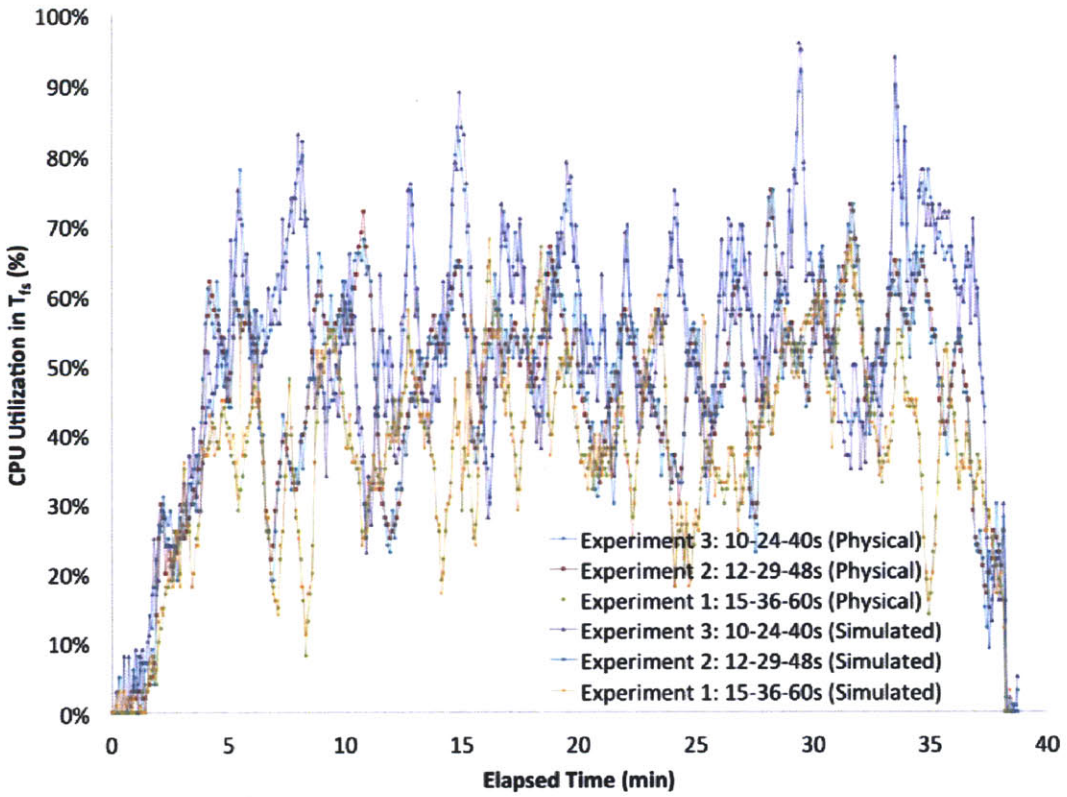

Figure 5-9: Comparison between CPU utilization in T_{fs} by experiment in the physical and simulated infrastructures.

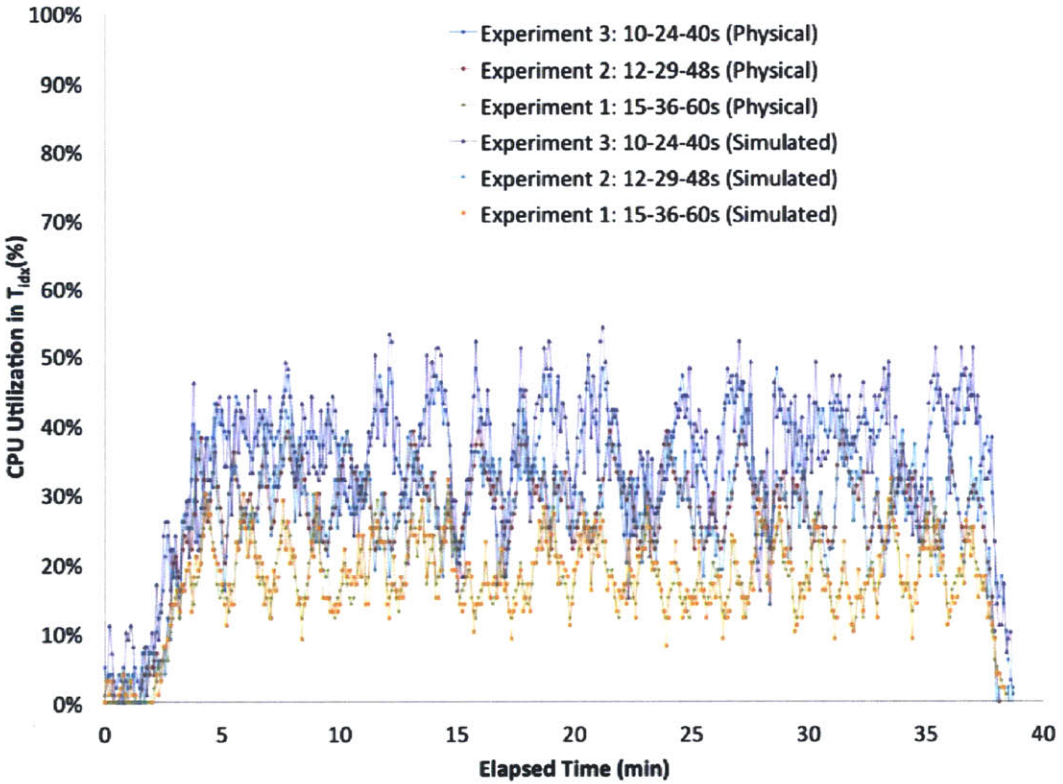

Figure 5-10: Comparison between CPU utilization in T_{idx} by experiment in the physical and simulated infrastructures.

5.3.3 Memory Validation

The simulator takes as input the memory cost of each message in each operation. The memory cost is obtained by profiling the execution of an operation launched by a single client in the physical infrastructure. GDISim computes the memory utilization of a server by accumulating the total memory usage of concurrent clients simultaneously allocating memory in that machine. Unfortunately the model does not take into account alternative components such as memory pools in the kernel or memory pools in the runtime environment.

Measurements of memory utilization of the physical infrastructure showed the following behavior for each tier:

1. T_{app}: A flat utilization of 32 GB was measured in the application servers for the duration of the entire experiment for the three experiments executed.

2. T_{db}: A flat utilization of 28 GB was measured in the database server for the duration of the entire experiment for the three experiments executed.

3. T_{fs}: A flat utilization of 12 GB was measured in the file server for the duration of the entire experiment for the three experiments executed.

4. T_{idx}: A flat utilization of 12 GB was measured in the index server for the duration of the entire experiment for the three experiments executed.

For the three experiments and all the tiers of the physical infrastructure the absence or existence of workload did not augment or shrink the memory utilization. The memory utilization remained flat, as set to the size of the memory pools. The kernel maintains a flat memory profile by swapping intermediate data to disk, while the memory utilization caused by the clients in the simulated experiments is orders of magnitude smaller than the memory pool size of the runtime environment.

Under these circumstances, it is concluded that the current method to estimate memory allocation is not sophisticated enough and requires the addition of models considering the effects of the Operating System and the runtime environments.

Experiment	CPU T_{app}	CPU T_{db}	CPU T_{fs}	CPU T_{idx}	#C	R_t
1: 15-36-60	9.07%	11.41%	7.51%	6.12%	5.98%	5.01%
2: 12-29-48	9.94%	12.56%	7.05%	5.40%	5.12%	6.92%
3: 10-24-40	10.11%	11.29%	7.42%	5.83%	6.52%	6.62%

Table 5.3: Root Mean Square Error (RMSE) by experiment and measurement.

5.3.4 Accuracy Assessment

In Section 5.3.2 the average and standard deviation CPU utilization of the physical and simulated steady state phases were compared for each tier (T_{app}, T_{db}, T_{fs} and T_{idx}) and in each of the three experiments. However, the goal of this section is not only to compare average and standard deviation values in steady state, but to calculate the error between measured and predicted values for the entire duration of each experiment and provide a measure of accuracy. Table 5.3 presents the Root Mean Square Error (RMSE) for each experiment and tier. Additionally, the number of concurrent clients and response times are compared using this metric as well. The RMSE is calculated as in Equation 5.5.

$$RMSE = \sqrt{\frac{1}{N}\sum_{t_i=t_0}^{t_i=t_{N-1}}\left(P_{T_x}^{y}(t_i) - \widehat{P}_{T_x}^{y}(t_i)\right)^2} \tag{5.5}$$

$P_{T_x}^{y}(t_i)$ is the average CPU utilization value measured in the physical infrastructure at time step t_i at tier x during experiment y, while $\widehat{P}_{T_x}^{y}(t_i)$ is its simulated counterpart. N is the number of samples taken during the experiment starting at t_0 and ending at t_{N-1}.

Table 5.3 shows that the RMSE of the CPU utilization between the physical and simulated infrastructures ranged between approximately 5% to 13%, with T_{db} and T_{fs} consistently producing the largest disparity across the three experiments. The RMSE of the number of concurrent clients between the physical and simulated infrastructures was 5.1-6.5%, while the response time error ranged between 5.0-6.9%

The response time estimates of this work are comparable to the results provided by previous work on evaluation of computer systems. Urgaonkar et al. reported

estimates of response time in their analytic model for multi-tier data centers with confidence intervals of 95% [85]. Similarly, MDCSim reported an average deviation of 10% in the latency of the operations simulated.

5.4 Summary

In this chapter the process to validate the hardware and software models presented in Chapter 3 has been presented. The procedure utilized is based on the comparison between a physical infrastructure running a CAD software application and a simulated infrastructure modeling both the hardware and the CAD software application.

First, details on the physical infrastructure used for validation and operations composing the CAD software were provided. Next, the three experiments executed in both infrastructures, physical and simulated, were explained, along with the underlying assumptions. The measurements collected in both systems were gathered and reported, in order to facilitate the assessment of the feasibility and accuracy of the data center and software models constructed in Chapter 3. Results show that not only are response times comparable to previous work in this field, but also that data center operators can benefit from CPU utilization estimates as they provide insight into the capacity planning of a system. Finally, the results of the analysis indicate that memory allocation models require more granularity on the OS and runtime environment in order to be valuable.

Chapter 6

Data Serving Platform Consolidation

6.1 Introduction

In this chapter, the potential of GDISim is demonstrated through a data center consolidation case study carried out for a Fortune 500 company. This company is currently running eleven data centers spread across different continents. These data centers are responsible for providing the employees the capability to create and manipulate information assets throughout the organization, particularly by establishing the foundation to run collaborative Computer-Aided Design software, which is key for the creation of products in this company.

The continuous improvement of hardware, software and networks, along with the constant process of cost reduction and simplification of IT infrastructures, have prioritized the target to cut costs in the Data-Serving Platform of this Fortune 500 company. The approach is to reduce costs by consolidating data centers, while 1) maintaining the same user experience for clients consuming the data, 2) utilizing in a cost-effective manner the hardware allocated for each data center, and 3) optimizing the background processes that enable having productivity and availability enhancing features such as search or replication.

The chapter is organized as follows. First, the characteristics of a Data-Serving

Platform are introduced, along with the custom requirements established by the Fortune 500 company. Next, the simulation inputs and the corresponding predicted outputs are presented. Finally, the results of this case study are summarized.

6.2 Data Serving Platform

Data-Serving Platforms are hosted, centrally-managed, but geographically distributed systems that serve data to one or multiple internet-scale web or native applications running in mobile or desktop environments. Figure 6-1 illustrates a traditional data-serving platform for a global organization. Data centers in different locations are responsible for giving service to the subset of clients in their geographic proximity. This distributed nature has two inherent advantages:

- *Low-Latency*: Reduced distances between clients and the data result in lower latency impact for access to information. Greenberg et al. [38] corroborate the importance of geographical distribution in the pursuit of lowering latency.
- *High-Availability*: Multiple redundant copies of data files increase the availability of the information, and the existence of multiple replicas reinforces the fault-tolerance of the system.

Unfortunately, the positive characteristics provided by geographical distribution carry inherent negative side effects. Problems that do not exist in geographically centralized systems suddenly become challenging in distributed infrastructures. Typically two problems surface: *Stale Data* and *Unsearchable Data.*

- *Stale Data*: Propagation of file changes in one location is not carried out instantaneously to the others, and clients in remote locations may access "stale" versions of the recently modified file.
- *Unsearchable Data*: "Fresh" data files need to be searchable by clients in any data center, but their registration into the index may require not only the analysis of these new data files but also all their relationships.

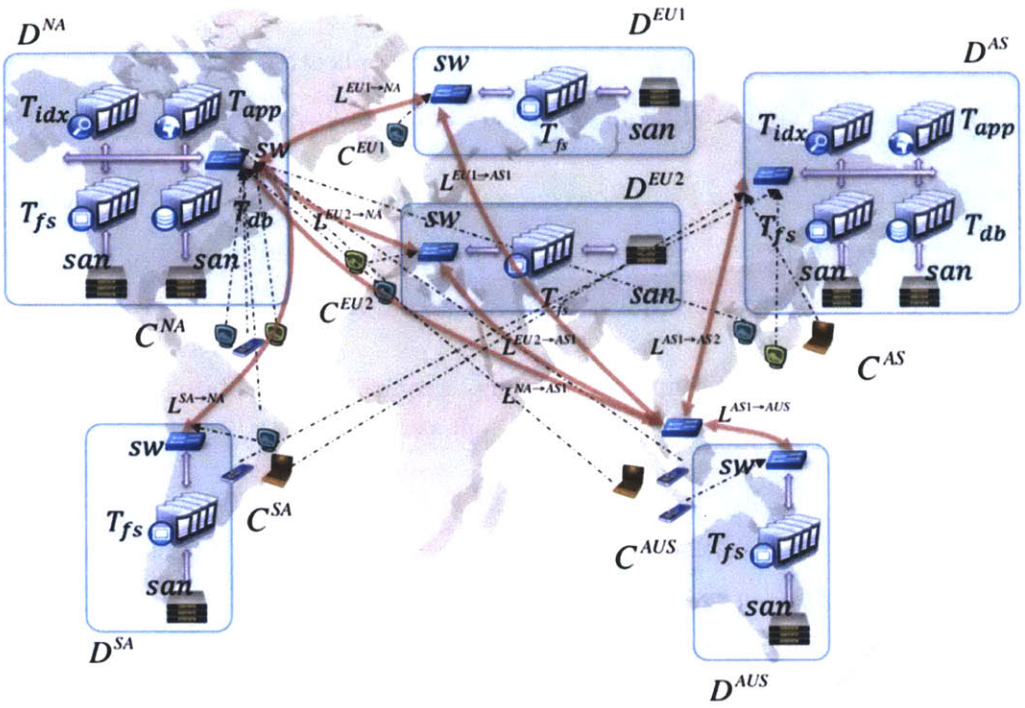

Figure 6-1: Illustration of a generic Data Serving Platform.

Typically, these effects can be alleviated by the introduction of automated background processes that, while running simultaneously with other software applications, perform the data movement and processing tasks necessary to minimize the intervals in which stale versions are exposed or data is unsearchable. In order to support these background processes some (but not all) data centers will require enhanced data management capabilities. Furthermore, background processes must be carefully monitored and the underlying infrastructure of the platform appropriately dimensioned, taking into account the impact of these processes in conjunction with application workloads. It is critical to guarantee that background processes have no interference with software applications and that they do not degrade client experience.

Summarizing, the design of Data Serving Platforms seeks the compromise of the following five characteristics:

- *Scalability*: It is desirable to have the ability to scale up on data volume generated and client population served by adding resources with minimal operational

effort and minimal impact on system performance.

- *Response Time and Geographic Scope*: Fast application response times to geographically distributed clients under dynamic load conditions is required.

- *High Availability and Fault Tolerance*: High degree of availability, with application-specific tradeoffs in the degree of fault-tolerance, is required in conjunction with a degree of consistency that is deemed acceptable in the presence of faults.

- *Consistency*: Specific applications may require serializability of transactions. Nevertheless, serializability can be inefficient and often unnecessary; hence many distributed replicated systems go to the extreme of providing only eventual consistency. The platform is required to provide different consistency levels, strict and eventual, on an application basis.

- *Cost*: The cost of a globally distributed system under dynamic workloads can be easily underestimated or overestimated. In order to run a successful platform platform it is necessary to fulfill the requirements established by the other four characteristics while building a cost-effective infrastructure.

6.3 Platform Requirements

In this section, the requirements established by the proposal to reduce IT infrastructure costs are presented. These requirements are divided into three groups: 1) Infrastructure and Network requirements, 2) Software and Workload requirements, and 3) Performance and Service level requirements.

6.3.1 Infrastructure and Network Requirements

The cost reduction initiative proposes downsizing the IT infrastructure from eleven data centers to six, one per continent (except Antarctica). Each data center will be responsible for serving data to geographically proximal clients (i.e. in its same continent). Data centers will be connected through high-speed networks. Background

processes will guarantee that the latest version of data files is accessible and searchable by clients associated to any data center.

Data centers will have different capabilities and responsibilities. In the consolidated infrastructure, one data center, D^{NA} will be denoted as the *Master Data Center* (MDC), while the rest (D^{EU}, D^{AS}, D^{SA}, D^{AFR} and D^{AUS}) will be *Slave Data Centers* (SDC). In addition to the data-serving capability provided by file server tiers (T_{fs}) in the SDCs, the MDC contains additional resources (application servers T_{app}, database servers T_{db} and index servers T_{idx}) for file management operations. The consolidated infrastructure is illustrated in Figure 6-2. The infrastructure must provide the following file management capabilities:

- *Authentication & Authorization*: T_{app} is responsible for checking the credentials of the clients demanding access to the data served by the infrastructure. Additionally, T_{app} checks whether the client has permission to launch the operation requested. For example, some clients will be granted read-only access to the data and will not be authorized to save files.

- *Versioning*: T_{db} entries store meta-data associated with each data file. The database server is responsible for registering the history of modifications of each file, along with pointers to the current and older versions. It also keeps information about the status of the versions stored in the SDCs.

- *Synchronization & Replication*: In order to guarantee that file changes are propagated to data centers with stale versions, the MDC runs a continuous synchronization process that copies files across data centers. Additionally, file replication policies will keep multiple copies of one file in the same data center and across data centers, so as to guarantee fault tolerance.

- *Indexing*: In order to make newly created or modified data searchable by clients, it is necessary to update search indices. For the case of text files, the words present and their relevance are considered towards the index build process. For the case of 3D model files, a navigable snapshot representing the spatial

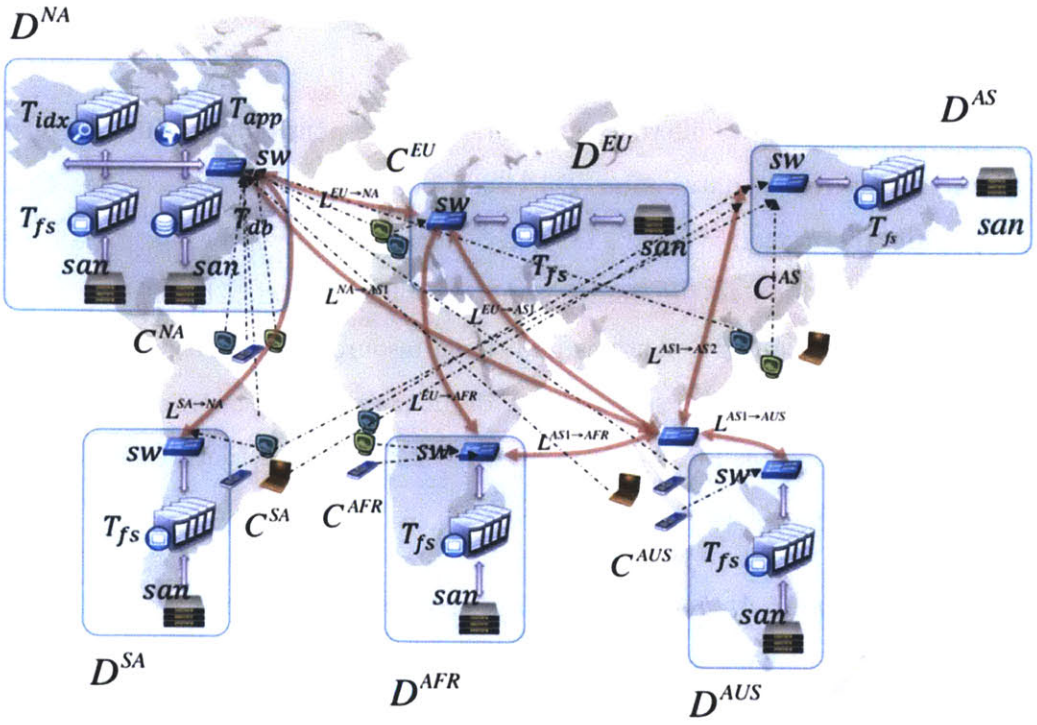

Figure 6-2: Proposed consolidated Data Serving Platform for the Fortune 500 company.

relationships between files is constructed. Index build processes are carried out in T_{idx}.

6.3.2 Software and Workload Requirements

The Fortune 500 company is required to run three software applications that manipulate the files provided by the Data-Serving Platform: *Computer-Aided Design (CAD)*, *Visualization (VIS)* and *Product Data Management (PDM)*. Simultaneously, the Data-Serving Platform executes two background processes to enable enhanced data management capabilities, these processes are: *Synchronization & Replication (SR)* and *Index Build (IB)*.

- *Computer-Aided Design (CAD)*: CAD software is created to facilitate the process of design and design-documentation. Typically, CAD environments not

only produce drafts of technical and engineering drawings, but also convey relevant information such as materials, processes, dimensions, and tolerances. CAD operations are the same as in Section 5.2.2: LOGIN, TEXT-SEARCH, FILTER, EXPLORE, SPATIAL-SEARCH, SELECT, OPEN and SAVE.

- *Visualization (VIS)*: VIS software is created to provide quick access to 2D and 3D models, their relationships and supporting information, in order to facilitate decision-making. VIS operations are analogous to CAD operations, but the volume of the data manipulated during file opening and saving is considerably smaller. VIS operations are: LOGIN, TEXT-SEARCH, FILTER, EXPLORE, SPATIAL-SEARCH, SELECT, OPEN and SAVE.

- *Product Data Management (PDM)*: PDM software is a tool to track and control data related to a particular product. Typically, the data associated to this product involves technical specifications, specifications for manufacture and development and the list of materials required to produce it. PDM operations are: BILL-OF-MATERIALS, EXPAND, PROMOTE, UPDATE, EDIT, DOWNLOAD AND EXPORT.

- *Synchronization & Replication (SR)*: SR is a background process running in the MDC that periodically schedules a sequence of file movements between data centers to ensure that multiple copies of the latest files exist and are available in all geographic locations. This process has two phases: *Pull* and *Push*. During the Pull phase, the subset of files that were modified since the last execution of the SR process are collected and transmitted from SDCs to the MDC. Upon reception of these files the Push phase is started, the MDC keeps a copy of each new file and scatters another to all the SDCs except from the file creator. SR has a single operation called SYNCHREP.

- *Index Build (IB)*: IB is a background process running in the MDC that periodically analyzes newly created or modified files and updates both the text-search index and the 3D snapshots for spatial-search. The subset of files brought to

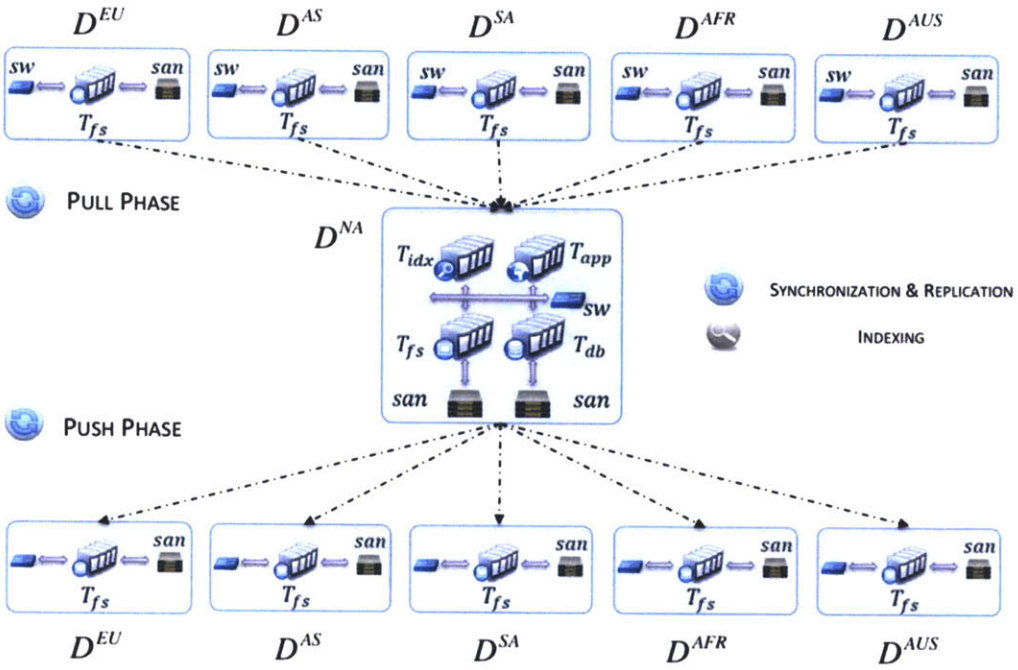

Figure 6-3: Background processes running on the IT Infrastructure of the Fortune 500 company.

MDC as part of the Pull phase of the SR process is flagged to be indexed. Periodically, the IB process collects and processes the files flagged. IB has a single operation called INDEXBUILD.

SR and IB processes are illustrated in Figure 6-3. The consolidated infrastructure must serve data to the same client population that the original infrastructure supported. In some geographic locations multiple data centers will be merged into one. In these cases the resulting data center must be resized to be able to accommodate the sum of the application workloads of the original merged data centers.

SR is launched every ΔT_{SR}. If the duration of ΔT_{SR} is less than the duration of SYNCHREP, then multiple independent SYNCHREP operations will overlap. On the contrary, IB is launched ΔT_{IB} after the last INDEXBUILD operation concluded. Therefore, only one INDEXBUILD operation can run at a time.

6.3.3 Performance and Service Level Requirements

The primary performance requirement for the consolidated infrastructure is to have the capacity to accommodate the worldwide workload of the original infrastructure during the peak periods of the day, and during the busiest days of the week, without jeopardizing the client experience and keeping operation response times aligned to their baseline values. Saturation of resources in any of the servers comprising the infrastructure produces nonlinear behavior that results in the degradation of client experience. Therefore, the consolidated infrastructure must guarantee capacity to absorb peak workloads while keeping a sensible distance from saturation and utilizing infrastructure resources in a cost effective manner.

The secondary performance requirement for the consolidated infrastructure is to be able to run CAD, VIS and PDM operations along with SR and IB background processes without exceeding 20% of the capacity of the network. In the future, the organization running the system may need to increase the number of applications running on top of this data center infrastructure or may need to deploy a parallel data center infrastructure that is interconnected through the same network infrastructure. Under these circumstances, the company allocates 20% of the network for these specific applications and requires that this limit is not exceeded.

The primary service level requirement for the consolidated infrastructure is to serve data files to geographically distributed clients with the lowest latency possible. Ideally, for this to happen, clients should be able to receive the latest versions of the files they need from their local (closest) data center. Unfortunately, propagation of file changes does not happen instantaneously and the probability of the latest version of a file being locally available depends on the frequency of execution of the synchronization and replication process. The effectiveness of the SYNCHREP operation is indicated by the maximum time that a stale file can reside in a data center without being replaced by its latest version, this is denoted as R_{SR}^{max}. On the contrary, overly frequent execution of the synchronization operations imposes an additional load on the system that could lead to saturation of the servers. For these

reasons, it is necessary to find a synchronization operation frequency that yields a compromise, keeping R_{SR}^{max} at acceptable levels whilst not exposing the infrastructure to the risk of saturation.

The secondary service requirement for the consolidated infrastructure is to minimize the time interval between the creation or manipulation of data files and the instant that this new information is searchable within geographically distributed clients. Analogous to the synchronization operation, the performance of the index build process depends on the launch frequency of the operation as well as the resources utilized for the process. Nevertheless, unlike synchronization which can be carried out in parallel, indexing requires the analysis of relationships between multiple interrelated files and this step might not be parallelizable. The effectiveness of the INDEX BUILD operation is indicated by the maximum time that new data remains unsearchable by other clients. This is denoted as R_{IB}^{max}. Hence, it is necessary to configure an index build operation that yields a compromise of keeping R_{IB}^{max} at acceptable levels whilst not placing the infrastructure at risk of saturation.

6.4 Consolidation: Simulation Inputs

In this section the inputs passed to GDISim to evaluate the effects of the cost reduction proposal are presented. The simulator accepts information about the data centers forming the consolidated infrastructure and the network topology that interconnects them. Next, the definitions of the operations representing the software that will be launched on this infrastructure along with their corresponding workloads are presented. Finally, the definitions and schedules of the operations representing background processes are covered.

6.4.1 Infrastructure Hardware & Topology

Figure 6-4 illustrates the global network of data centers that comprise the consolidated infrastructure. The hardware specifications for each tier in each data center and the connectivity characteristics between them are indicated using the same notation as

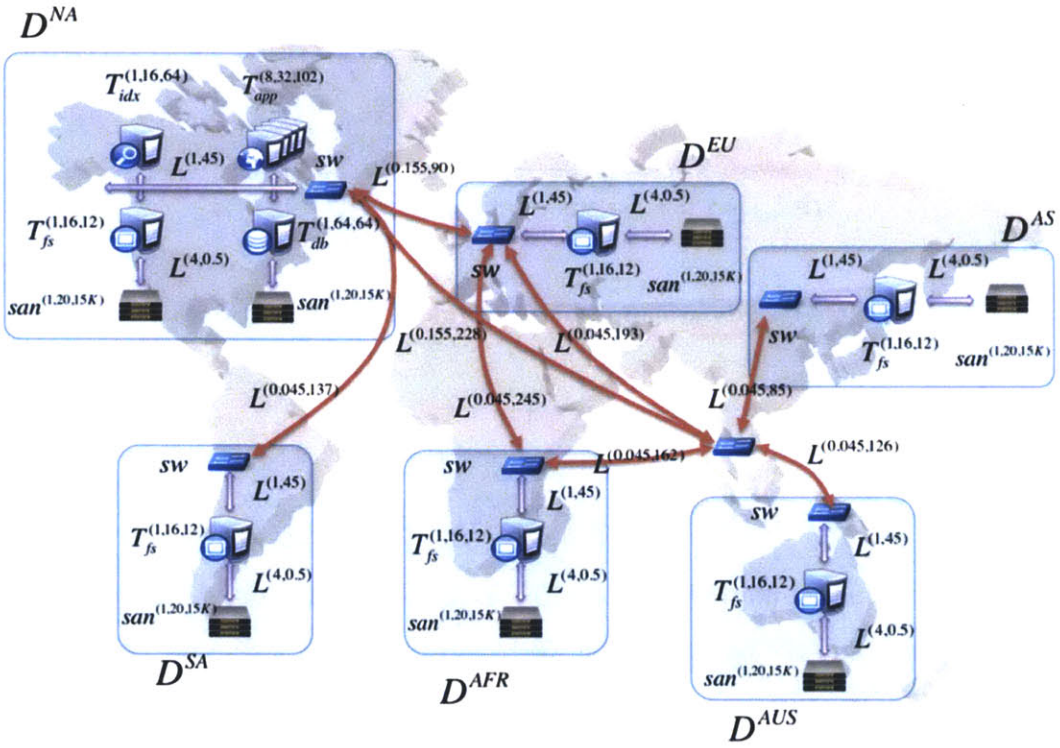

Figure 6-4: Proposed consolidated infrastructure hardware and network topology specifications.

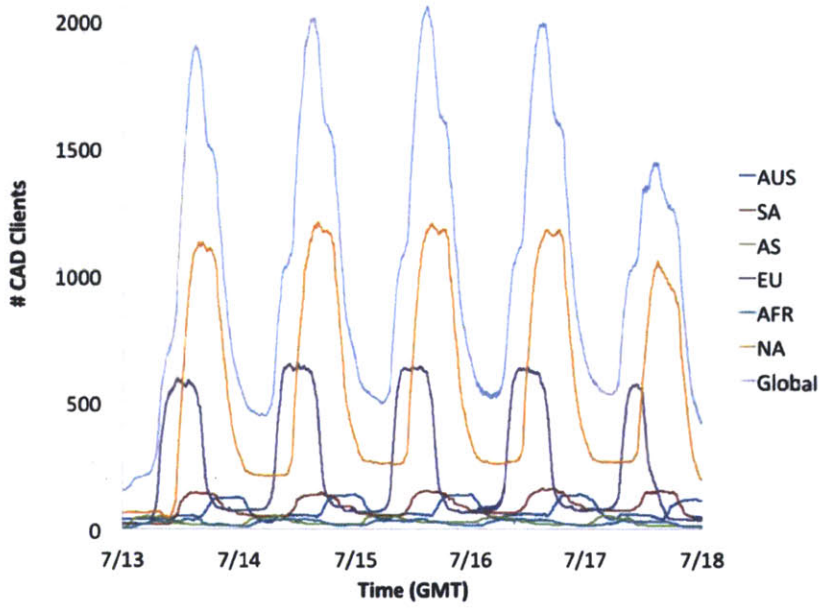

Figure 6-5: CAD software application workload in different data centers.

in Section 5.2.1.

6.4.2 Software Applications & Workload

This section includes information about the message cascades used to represent software operations and the corresponding workloads to be launched in the consolidated simulated infrastructure.

Computer Aided Design (CAD)

Figure 6-5 shows the weekly (workdays) CAD workload to be imposed on the consolidated infrastructure by each data center. CAD workloads follow repetitive patterns, but it is noticeable that Wednesdays receive the highest population of CAD clients. The simulator will focus on the worst case scenario taking this day as a reference workload. The peak time of the day occurs when D^{NA} and D^{SA} overlap with D^{EU}, with a peak population of over 2000 CAD clients.

The distribution of operation types is assumed to be uniform throughout the day

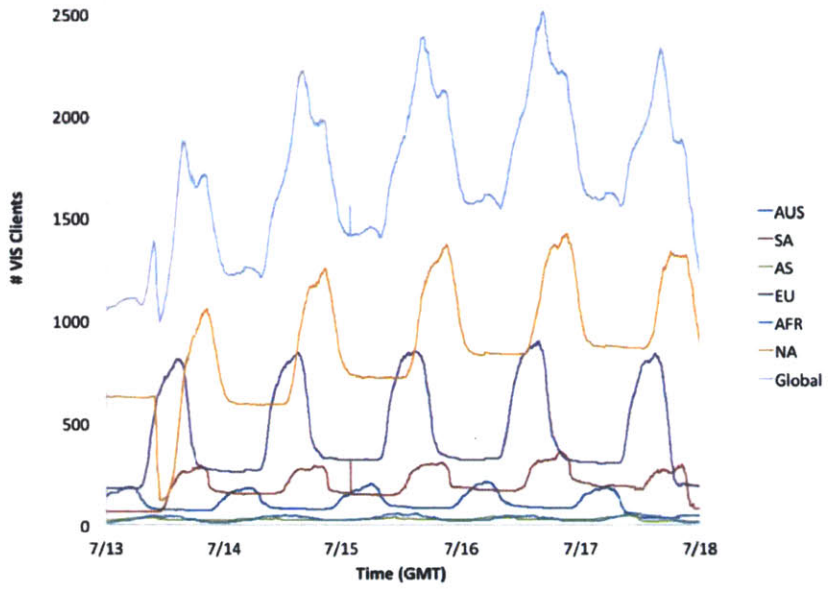

Figure 6-6: VIS software application workload in different data centers.

for these simulation experiments. The CAD operation definitions are identical to the ones presented in Section 5.2.3.

Visualization (VIS)

Figure 6-6 shows the weekly (workdays) VIS workload to be imposed on the consolidated infrastructure by each data center. Thursdays show the highest population of VIS clients and the simulator will take this day as reference. The peak time of the day occurs when D^{NA} and D^{SA} overlap with D^{EU}, with a population of over 2500 VIS clients.

The distribution of operation types is assumed to be uniform throughout the day for these simulation experiments. The VIS operation definitions are identical to the CAD operations. They only differ on the $\bar{R}$ parameter arrays that encode the cost of each message.

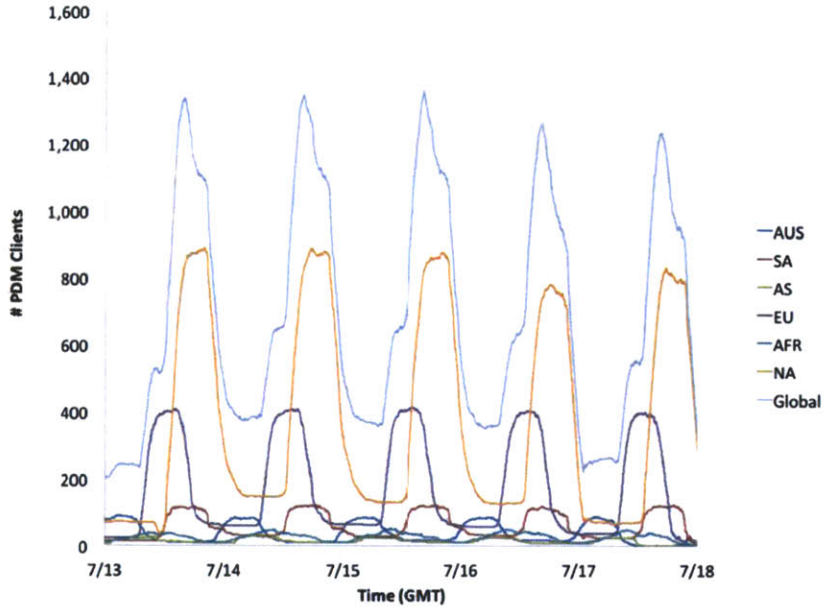

Figure 6-7: PDM software application workload in different data centers.

Product Data Management (PDM)

Figure 6-7 shows the weekly (workdays) PDM workload to be passed to the consolidated infrastructure by each data center. PDM workloads follow repetitive patterns, but it is noticeable that Wednesdays receive the highest population of PDM clients. The simulator will focus on the worst case scenario taking this day as a reference workload. The peak time of the day occurs when D^{NA} and D^{SA} overlap with D^{EU}, with a population of almost 1400 PDM clients.

The distribution of operation types is assumed to be uniform throughout the day for these simulation experiments. Primarily, PDM operations represent database transactions of different types. These involve long sequences of interactions between clients C and T_{db} via T_{app}. No other tiers are involved. The operation definition for PDM operations is omitted for simplicity.

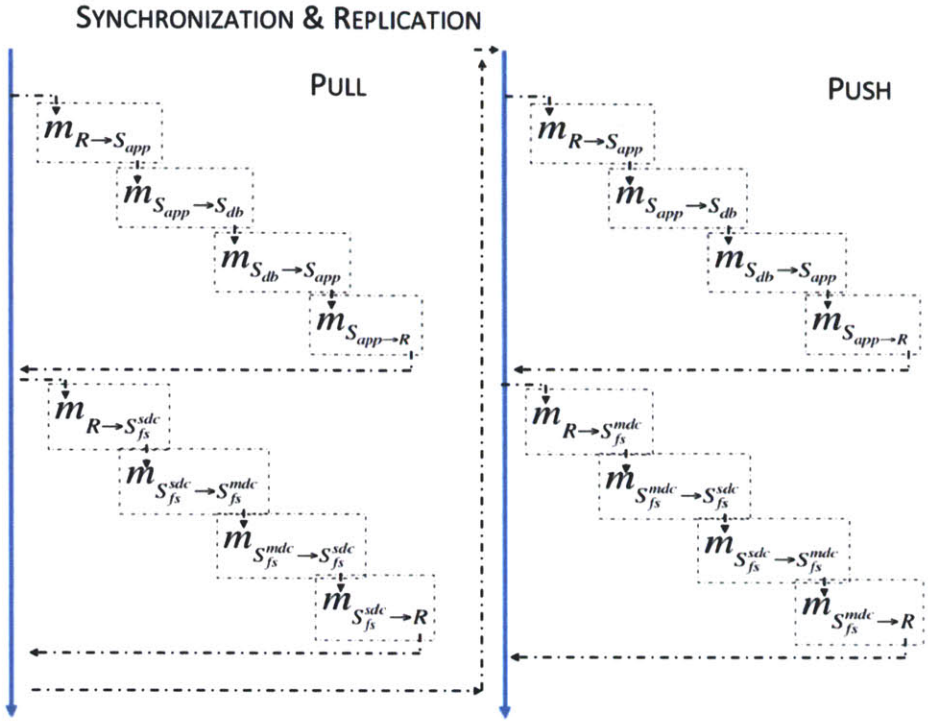

Figure 6-8: Message cascade for the SYNCHREP operation

6.4.3 Background Processes

While CAD, VIS and PDM applications are launched by clients, SR and IB are triggered by lightweight daemon processes running within D^{NA}.

Synchronization & Replication

Figure 6-8 describes the sequences of messages that occur as part of the SYNCHREP operation. R represents the daemon process responsible for scheduling and managing the synchronization & replication process. As explained in Section 6.3.2, SYNCHREP can be divided into two phases, *Pull* and *Push*.

During the Pull step, R queries the database for a list of files that have been modified in a specific data center and that need to be propagated to the rest. Then, these files are copied from that SDC to the MDC. Pull steps corresponding to different data centers are executed simultaneously.

The Push step carries out the opposite action. R queries the database for a list

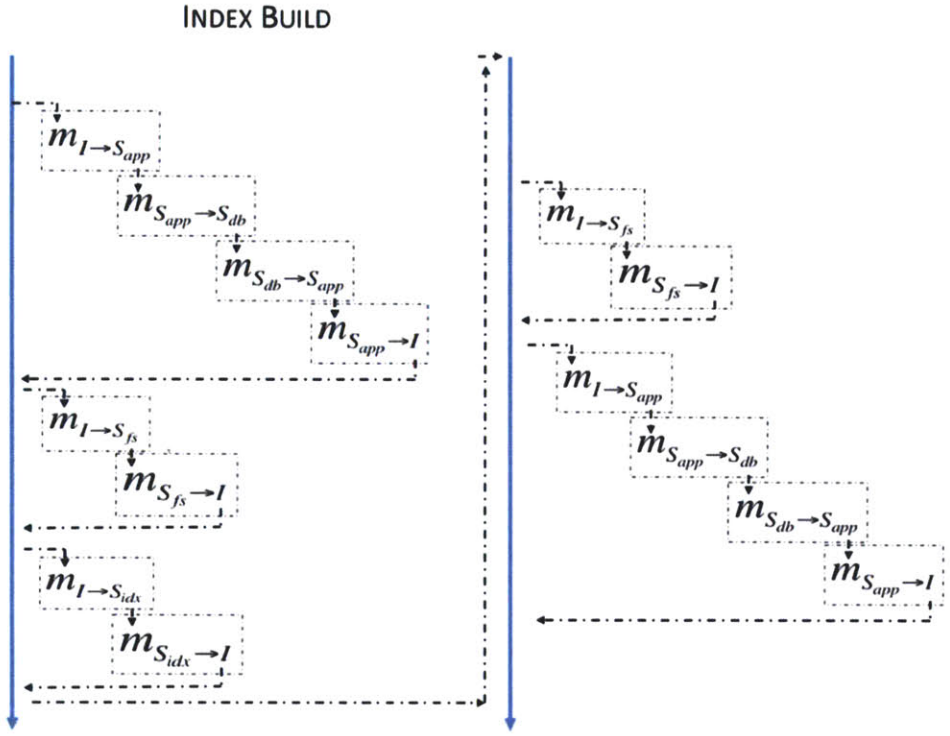

Figure 6-9: Message cascade for the INDEXBUILD operation

of files whose latest version is missing in a specific SDC and need to be copied from the MDC to that SDC. Similarly, Push steps corresponding to different SDCs are executed simultaneously.

SYNCHREP operations are launched every $\Delta T_{SR} = 15$ min, regardless of whether there are other SYNCHREP operation instances running at the same time. Each SYNCHREP operation takes care of the subset of files modified during that $\Delta T_{SR} =$ 15 min interval.

Index Build

Figure 6-9 describes the sequences of messages that occur as part of the INDEXBUILD operation. I represents the daemon process responsible for scheduling and managing the index build process. The index build process is launched every $\Delta T_{IB} = 5$ min after the completion of the previous INDEXBUILD operation execution. Therefore, only one INDEXBUILD can be running at a time.

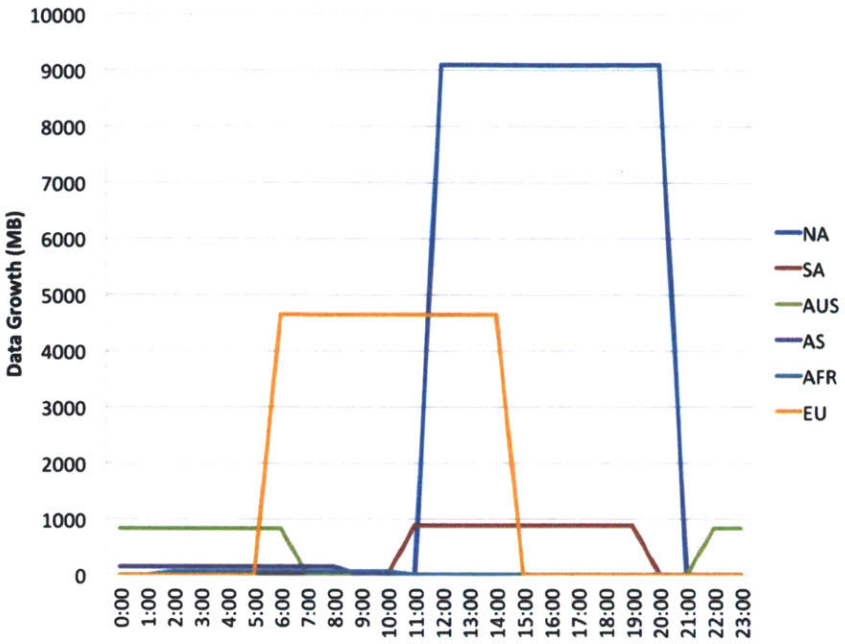

Figure 6-10: Data growth (MB) by hour by data center

Data Growth

The impact and effectiveness of the SR and IB processes is directly related to the volume of new data generated in different data centers at different times of the day. GDISim takes information about the data growth in each data center and uses the average file size to estimate the number of files to be transferred during the Pull and Push steps of the SYNCHREP operation. The average file size for this simulation is 50 MB. Figure 6-10 shows the data growth measurements provided by the Fortune 500 company to be utilized in the simulation.

Using the data growth information, it is possible to derive the volume of information that will be moved during the Pull and Push phases at different times of the day. This is illustrated in Figure 6-11. As it could be expected, during the peak workload hours (D^{NA} and D^{SA} overlap with D^{EU}) the largest volume of information is generated and must be propagated to the rest of data centers.

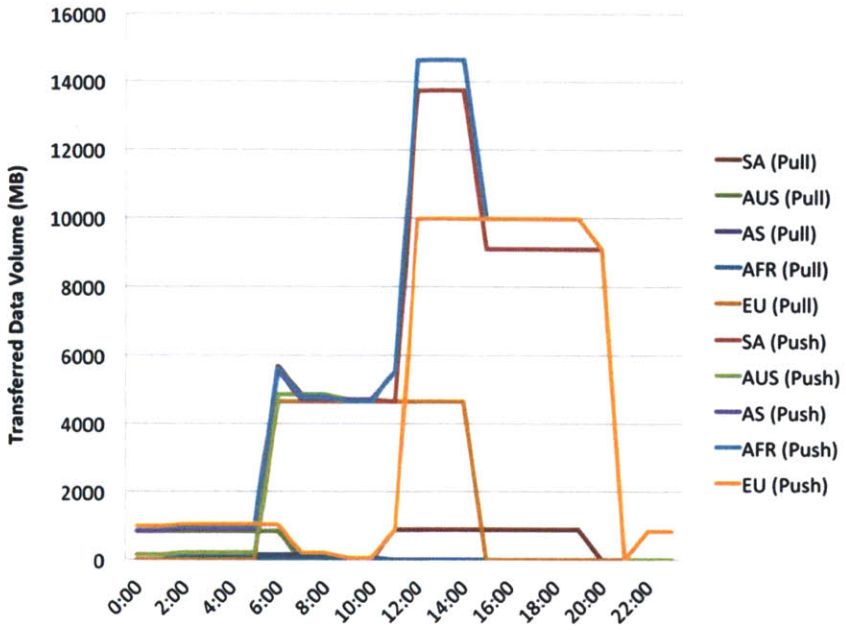

Figure 6-11: Data volume (MB) to be transferred during Pull/Push phases to/from D^{NA}

6.5 Consolidation: Simulation Outputs

This section presents the results produced by GDISim after running it with the information about the consolidated infrastructure gathered in Section 6.4. The results are classified into four sections: First, the CPU utilization measurements for different tiers and data centers are presented. Second, observations about the network utilization of the infrastructure are explained. Third, the effectiveness of the SR and IB operations is analyzed by estimating R_{SR}^{max} and R_{IB}^{max}. Finally, the client experience in different data centers is evaluated.

6.5.1 Computation Performance Results

Figures 6-12 and 6-13 illustrate the CPU utilization predictions delivered by GDISim for the server tiers in D^{NA} and in D^{AUS} respectively. The CPU utilization values shown represent the average utilization across servers in the same tier. The remaining data centers yield comparable conclusions to D^{AUS} and are omitted for simplicity.

D^{NA} is the master data center and all the operations launched worldwide are

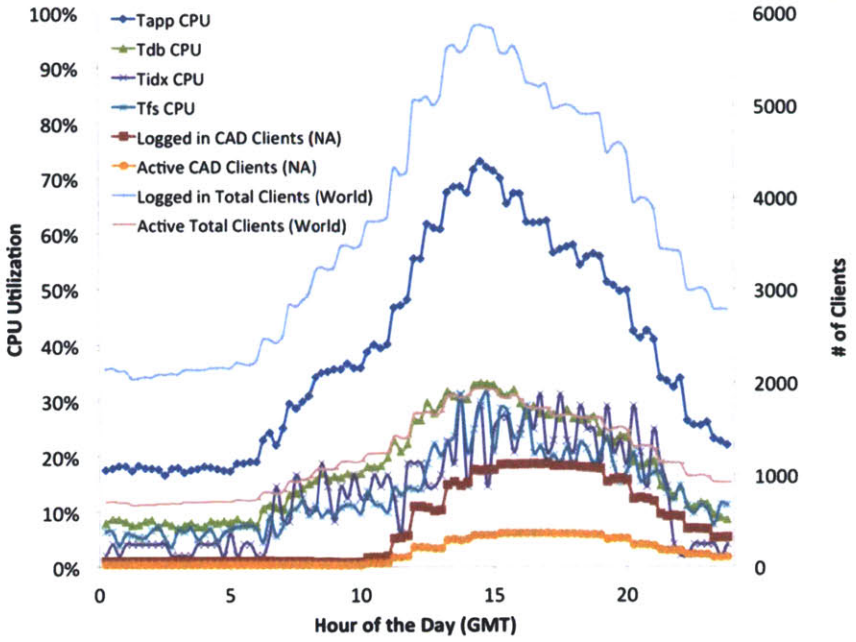

Figure 6-12: CPU Utilization (T_{app}, T_{db}, T_{fs} and T_{idx}) in D^{NA}

authorized through this data center. For these reasons, D^{NA} is expected to be the busiest data center and it is critical to keep it under saturation limits. Figure 6-12 shows an estimation of approximately 73% utilization in T_{app} at 15:00 (GMT), which is the busiest time of the day with ~6000 clients logged into the system and ~2000 concurrently manipulating data. T_{db} (32%), T_{idx} (30%) and T_{fs} (31%) also have their peaks near 15:00 (GMT) but their utilization is below T_{app}.

D^{AUS} is a slave data center that serves files exclusively to the clients that are geographically close to it. As shown in Figure 6-13, the utilization of T_{fs} follows closely the local workload and the estimation is approximately 3.5% which presents a very low saturation risk.

6.5.2 Network Performance Results

In this section, a *sleeping* data center is defined as a data center location outside its 9-hour local business hour window, while an *active* data center is defined as a data center location within its local business hour window. The period with the largest load on the network, 12:00-16:00 (GMT) is caused by two phenomena that are directly

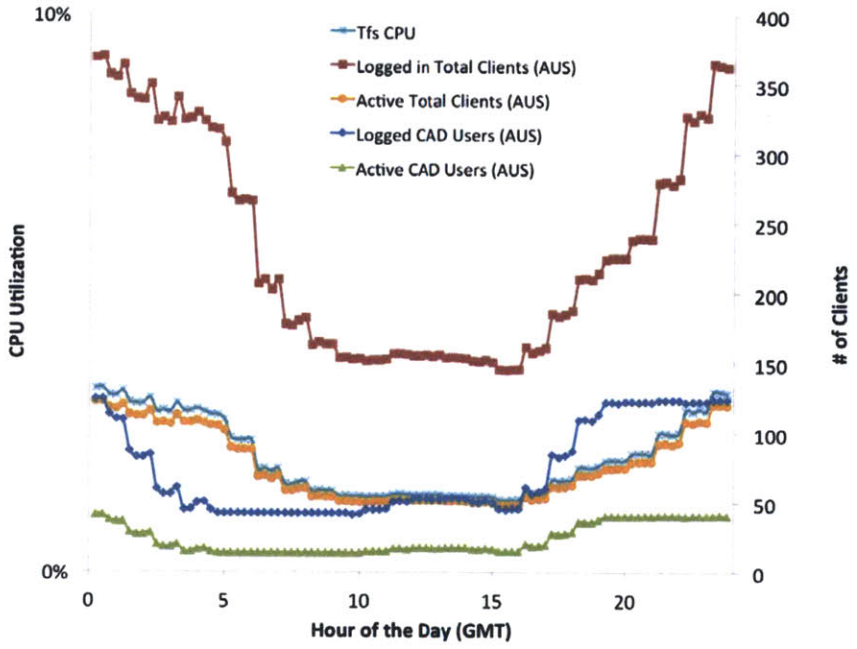

Figure 6-13: CPU Utilization (T_{fs}) in D^{AUS}

related to each other.

- *Peak Workload*: 12:00-16:00 (GMT) is the peak workload time, when the largest population of users are using the global network to initiate operations that require communication between the SDCs and the MDC. In particular, this affects the network links of the data centers that overlap during this period of time, $L^{NA \rightarrow EU}$ and $L^{NA \rightarrow SA}$.

- *Peak Data Growth*: The largest population of concurrent clients coincides with the largest volume of data creation during 12:00-16:00 (GMT). D^{NA} and D^{EU} are the data centers reporting the largest volumes of new files. During this overlap period, the MDC pushes the files created by D^{NA} and D^{EU} to the rest of the data centers simultaneously. In particular, this affects all the links connecting D^{NA} with all the sleeping SDCs. It is important to note that although D^{EU} is an active SDC, the $L^{NA \rightarrow EU}$ link is loaded not only with the requests generated by its active client C^{EU} population, but also with the data transfers to (Pull) and from (Push) D^{NA}.

	μ_U (%)
$L^{NA \rightarrow SA}$	48
$L^{NA \rightarrow EU}$	43
$L^{NA \rightarrow AS1}$	59
$L^{EU \rightarrow AFR}$	0
$L^{EU \rightarrow AS1}$	0
$L^{AS1 \rightarrow AFR}$	53
$L^{AS1 \rightarrow AS2}$	47
$L^{AS1 \rightarrow AUS}$	54

Table 6.1: Average utilization of the allocated capacity during 12:00-16:00 (GMT) for each network link.

Table 6.1 presents the estimations produced by GDISim for the average utilization, μ_U of the allocated capacity (20%), for the 12:00-16:00 (GMT) interval by network link. $L^{EU \rightarrow AFR}$ and $L^{EU \rightarrow AS1}$ are redundant network links that are used only in case of failure. Even if these backup links could be used for load balancing the traffic to AS1 across network links, this is out of the scope of this experiment. The results illustrate that pushing the new data to sleeping data centers takes almost 60% of the allocated capacity (12% from total) in the $L^{NA \rightarrow AS1}$ link. $L^{NA \rightarrow EU}$ is a critical network link that must be monitored, not only because it is receiving new data from D_{NA} but because it is also serving a population of approximately 1700 clients simultaneously. The utilization of this link is 43% (8.6% from total).

6.5.3 Background Process Performance Results

Figure 6-14 illustrates the response time estimation of the Synchronization & Replication and Index Build background processes predicted by GDISim. The response time represents the duration of the operation, from the time it was launched until its conclusion. In both cases, the duration of these processes is directly dependent to the volume of new information generated.

The interval of the day with the largest generation of new data, 12:00-15:00 (GMT), results in the period of time with the largest response time for the SYNCHREP operation. From Figure 6-14 it can be estimated that $R_{SR}^{max} = 31$ min. Stale

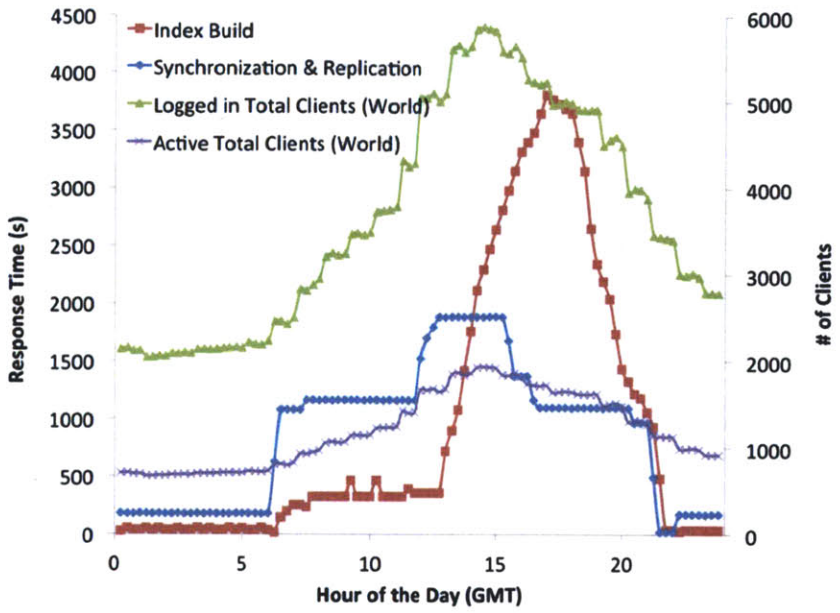

Figure 6-14: Response time of the background processes (SR and IB)

versions of data files will be exposed for a maximum time of approximately 31 minutes during the peak workload period of the day.

The INDEXBUILD operation behaves slightly differently. This operation is also directly affected by the period of time with the largest data generation. Nevertheless, the fact that each operation is launched a constant amount of time after the previous iteration concluded, allows new files to be accumulated while the operation is running. This execution policy results in a cumulative effect in which INDEX BUILD operations take longer, even after the peak workload has been left behind. For this reason, the largest response time for INDEXBUILD is produced at 17:00 GMT and it is estimated that $R_{IB}^{max} = 63$ min.

6.5.4 Client Experience Results

Figures 6-15 through 6-20 illustrate the evolution of the operation response time through the day for the CAD, VIS and PDM software applications at the D^{NA} and D^{AUS} data centers as predicted by GDISim. The estimations for the remaining data centers are analogous to D^{AUS} and are omitted for simplicity.

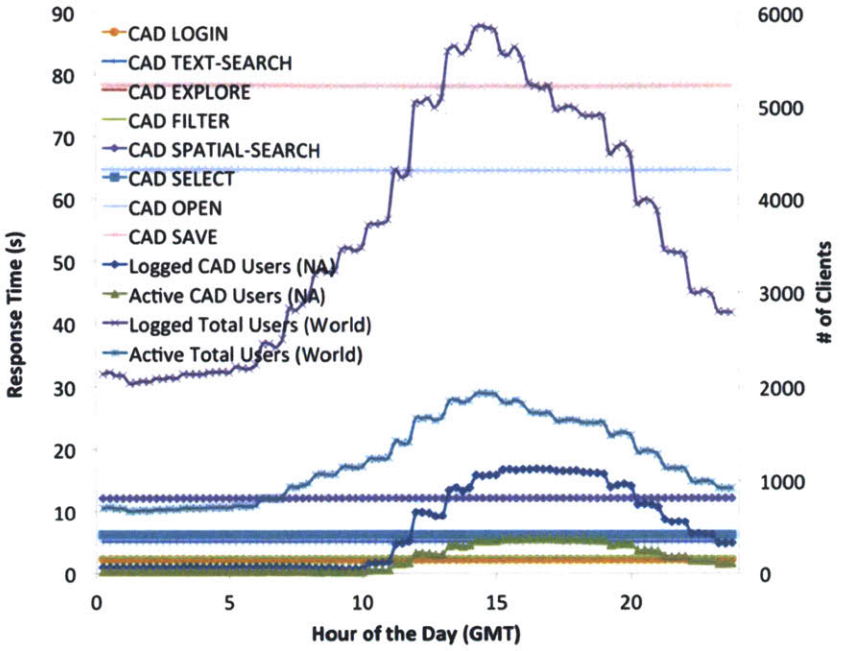

Figure 6-15: Response times for CAD operations in D^{NA}

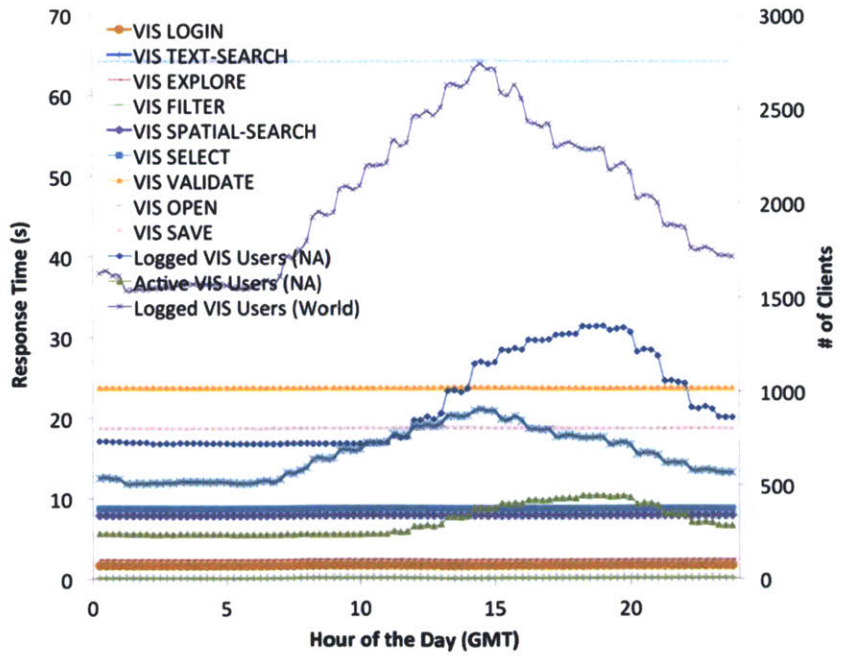

Figure 6-16: Response times for VIS operations in D^{NA}

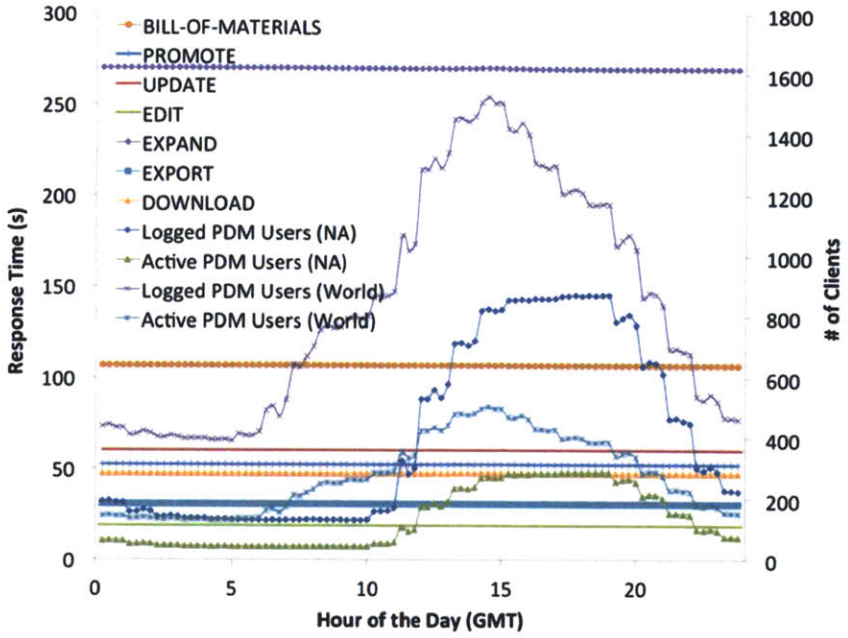

Figure 6-17: Response times for PDM operations in D^{NA}

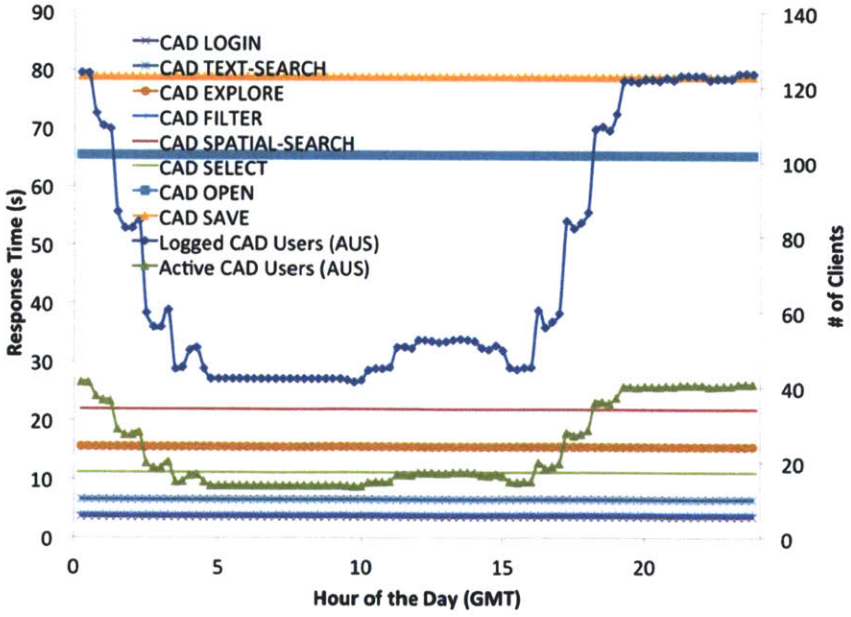

Figure 6-18: Response times for CAD operations in D^{AUS}

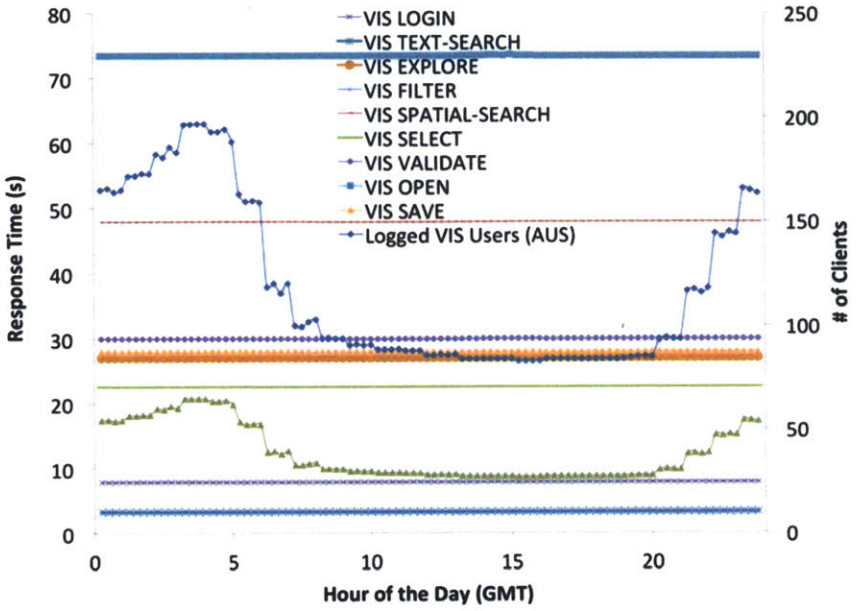

Figure 6-19: Response times for VIS operations in D^{AUS}

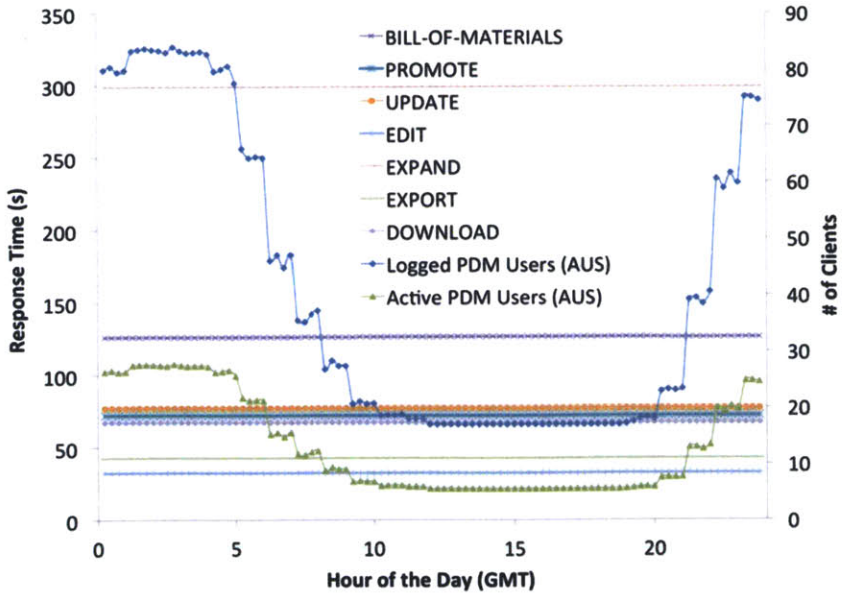

Figure 6-20: Response times for PDM operations in D^{AUS}

Operation *op*	R_{op}^{NA}	R_{op}^{AUS}	S	$\Delta R_{op}/R_{op}^{NA}$ (%)
CAD LOGIN	2.2	3.62	4	64.54
CAD TEXT-SEARCH	5.11	6.51	2	27.39
CAD FILTER	2.6	4.00	2	53.84
CAD EXPLORE	6.43	15.53	13	141.52
CAD SPATIAL-SEARCH	12.15	21.95	14	80.65
CAD SELECT	6.2	11.1	7	79.03
CAD OPEN	64.68	65.38	1	1.08
CAD SAVE	78.21	78.91	1	0.89

Table 6.2: Response time variation for CAD operations caused by the latency in D^{AUS}.

It was a requirement of the consolidated infrastructure that operation response times for all the applications remain unperturbed in spite of the peak application workloads. Below hardware saturation limits, response times are expected not to show any degradation, and this was confirmed by GDISim in both data centers D^{NA} and D^{AUS} for all the applications.

It is necessary to point out the impact of latency in operation response times. While there is no degradation of response times caused by workload variations, there are some differences in these timings for geographically distant data centers. This divergence is caused by network latency and becomes significant for operations involving multiple MDC-SDC interactions.

Table 6.2 illustrates the impact of latency for clients launching operations from the D^{AUS} data center compared to operations initiated locally by D^{NA} clients. R_{op}^{NA} and R_{op}^{AUS} represent response times for operation *op* in D^{NA} and D^{AUS} respectively. S represents the number of round-trips between D^{NA} and D^{AUS} as part of operation *op*. ΔR_{op} is the difference between R_{op}^{AUS} and R_{op}^{NA}. $\Delta R_{op}/R_{op}^{NA}$ represents the difference relative to the response time in D^{NA}. It is noticeable that lightweight operations involving multiple round trips (EXPLORE, SPATIAL-SEARCH or SELECT) suffer a significant degradation due to latency, while the impact of latency is negligible for heavy operations involving few trips (OPEN and SAVE).

6.6 Result Evaluation

This chapter presented the evaluation of a proposal to cut costs in the IT infrastructure of a Fortune 500 company. The proposal suggested to reduce the number of data centers from eleven to six, one in each continent (except Antarctica), while maintaining the service levels to the same client population and using resources in a cost effective manner. For this purpose, GDISim was utilized to assess the feasibility of the changes proposed in the consolidated infrastructure.

First, the nature and purpose of a Data-Serving Platform is introduced in Section 6.2. Geographical distribution of data centers is presented as a double-edged sword. The high availability and low-latency benefits are emphasized, while explaining the challenges arising as a consequence of manipulating data files concurrently in different geographical locations. Next, the performance and service requirements imposed by the Fortune 500 company on its Data-Serving Platform are introduced in Section 6.3.

In Section 6.4, the inputs passed to GDISim to reproduce the behavior of the consolidated infrastructure are gathered. This section covers details on the hardware and network specifications of the proposed system along with information on the software application workload and background processes scheduled.

The execution of the simulator using these inputs produced a report containing estimations that are explained in detail in Section 6.5. The hardware and network specifications of the consolidated infrastructure established by the Fortune 500 company resulted in a peak CPU utilization of 73% in $T_{app} \in D^{NA}$ and a peak occupation of 60% in the $L^{NA \rightarrow AS1}$ network link. Client experience was not degraded and operation response times were kept workload-agnostic in all data centers.

In addition to evaluating the resources necessary to serve a worldwide client population working with CAD, VIS and PDM software, GDISim provided valuable feedback on the impact and effectiveness of the background processes utilized for global synchronization and indexing. The maximum time a file version could remain stale before receiving an update is approximately 31 minutes. Similarly, the maximum time a new file could remain unsearchable by some data centers without being registered

in the index is approximately 63 minutes. The Fortune 500 company found these values acceptable.

Additionally, GDISim facilitated the understanding of the impact of latency in remote data centers, along with workload and data growth dynamics allowing data center operators to identify critical parts of the infrastructure and inefficiencies of the software.

Chapter 7

Background Process Optimization

7.1 Introduction

In this chapter the potential of GDISim is demonstrated again by estimating the impact of alternative synchronization, replication and indexing mechanisms for Data Serving Platforms based on recent research on very-large data repositories.

First, the concepts of *Data Ownership* and *Relaxed Consistency* and their implications are introduced. Next, the set of new parameters required by the simulator to reproduce the behavior of an infrastructure composed of multiple master data centers is presented. GDISim takes these parameters along with parameters introduced in Chapter 6 with minor alterations. Upon execution of GDISim with the new configuration, the computational utilization, network occupancy and response times are presented, focusing particularly on unveiling the effects of *Data Ownership* and *Relaxed Consistency*. Finally, the conclusions of these experiments and the potential benefits for the Fortune 500 company are summarized.

Throughout this chapter, the infrastructure simulated in Chapter 6 is referred as the *Consolidated Infrastructure*, while the new system is called the *Multiple master infrastructure*.

7.2 Data Ownership & Relaxed Consistency

In this section the Data Ownership and Relaxed Consistency ideas and their implications are explained.

7.2.1 Data Ownership

Chapter 6 did not explicitly introduce the concept of ownership of files by data centers, nevertheless the ownership idea was utilized in an implicit manner. *Data Ownership* by a data center in a Data Serving Platform can be defined as "*the exclusive right for a data center to control and be responsible of the management operations associated to a file hosted by the Data Serving Platform*". Therefore, when a data center owns a file it is responsible for coordinating the metadata operations that control versioning, synchronization with other data centers, fault tolerance through replication and indexing for text and spatial search.

In Chapter 6 the master data center, D^{NA}, was responsible for the metadata management operations of all the files served by the Data Serving Platform. Consequently, D^{NA} was the *owner* of all the files in the system and slave data centers only provided fast local access to information.

On the contrary, it is proposed in Chapter 7 that this responsibility is shared by upgrading all six locations to master data centers. In this scenario, files are assigned to the data center that is geographically closest to the largest volume of requests. This is illustrated in Figure 7-1.

With this new configuration, all data centers have file management capabilities and are responsible for smaller subsets of files. This requires the upgrade of five data centers form slaves to masters and results in a smaller number of managed files per data center when compared to the original D^{NA} in the consolidated infrastructure. Therefore, the resources required by the multiple master data center are expected to be less than in the original D^{NA}.

Access patterns for a file can change over time and this trend can lead to the majority of requests for a file to shift to an area closer to a data center that is not

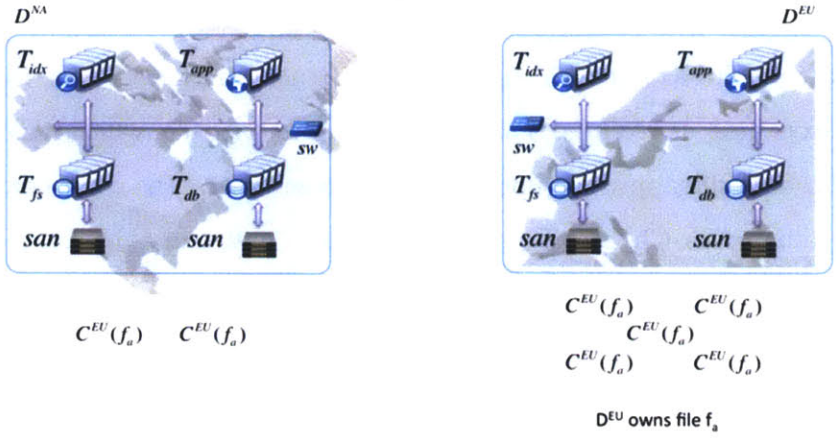

Figure 7-1: Example illustrating that file f_a is owned by D^{EU} because the largest volume of requests is originated from this data center.

the owner data center. These dynamics can be accommodated by transferring all the metadata associated to a file from the old owner data center to the new owner.

7.2.2 Relaxed Consistency

Due to the nature of the applications executed on top of the Data-Serving Platform, no inconsistent states are allowed. Nevertheless, in the pursuit of guaranteeing consistency, traditionally two extreme consistency models are contemplated:

1. *Serializable Transactions*: A set of transactions is serializable if its outcome is equal to the outcome of the transactions executed serially [10]. Supporting serializable transactions over a geographically replicated distributed system can be very expensive, and often inefficient [39]. Due to the performance and availability requirements imposed by the Fortune 500 company, serializability for transactions is considered impractical.

2. *Eventual Consistency*: Many geographically distributed systems go to the opposite extreme by providing only eventual consistency. In eventual consistency, clients update any replica of a file and eventually all the updates across different replicas are applied, but in different orders in different replicas [69]. This mechanism is too weak for the requirements imposed by the applications running in

the Fortune 500 company.

The Fortune 500 company demanded consistency guarantees that fall between these two extremes. It required *Timeline Consistency* per file, which means that all replicas of a data file apply the updates in the same order. The consolidated infrastructure evaluated in Chapter 6 provided timeline consistency per file by applying changes during the pull and push phases in order.

Nevertheless, in the context of this Data-Serving Platform comprised of multiple MDCs, it is necessary to distinguish between two different consistencies, *Data Consistency* and *Index Consistency*. Data consistency refers to the consistency of the data files served by the Data Serving Platform, while index consistency refers to consistency of the index files generated by running the index build process in multiple master data centers.

1. *Data Consistency*: The synchronization and replication of one data file is independent from the rest of the files served by the Data Serving Platform. Consequently, having six master data centers launching six independent SYNCHREP operations that apply pull/push phases in order, delivers the same timeline consistency guarantees that the single master data center did, for the consolidated infrastructure. However, since each master data center pulls/pushes only the modified file subset that is responsible for, it is expected that the duration of the SYNCHREP operation will be shorter. In turn, six concurrent SYNCHREP operations are expected to increase the network occupancy.

2. *Index Consistency*: As opposed to the synchronization and replication process, in which each file can be treated independently, the index build process not only requires the fresh file to be indexed, but also depends on multiple files through spatial or textual relationships. In the case of the single master data center, the index process is straightforward. All the required relationships and the file that is going to be indexed are locally available, and more importantly, these are the latest versions of each file. Consequently, the spatial or text index generated is completely up to date. The case of multiple master data centers is different.

The indexing process in a data center is guaranteed to have the latest version of the file to be indexed because it is owned by this data center. But may not have the latest of one or more of its relationships, particularly if these are owned by different data centers. Consequently, the spatial or text index generated is not inconsistent, but is only "partially consistent". Upon arrival of the latest version of the relationship files via the push phase of other SYNCHREP operations, the index is updated and will become "eventually consistent". It is expected that the durations of the INDEXBUILD operations in the multiple master case will be shorter than the INDEXBUILD for the single master case.

While a Data Serving Platform comprised of multiple masters delivers the same timeline consistency guarantees for data files that the single master platform did, the process of generating the spatial and text indices in different master data centers can only guarantee eventual consistency. The feasibility of a Data Serving Platform comprised of multiple masters relies on the capacity to accept the relaxation of the consistency guarantee of the indexing process, from timeline consistency to eventual consistency. At this point, it is the responsibility of the data center operators to analyze the impact of eventual consistency of the indexing process on current and future software applications exploiting data served by the infrastructure.

7.2.3 Related Work

In this Section two pieces of recent work are presented which focused on the optimization of synchronization, replication and indexing. In both cases, the system pursues high availability and low latency access for a large population of clients and is designed around the concepts of ownership and consistency.

PNUTS is a massively parallel and geographically distributed database system constructed by Yahoo for serving data to web applications [19]. PNUTS is designed to provide access to database records in ordered tables for large numbers of client requests (read and write) with low latency and per-record consistency guarantees. In order to be able to support a larger number of applications, PNUTS allows relaxed

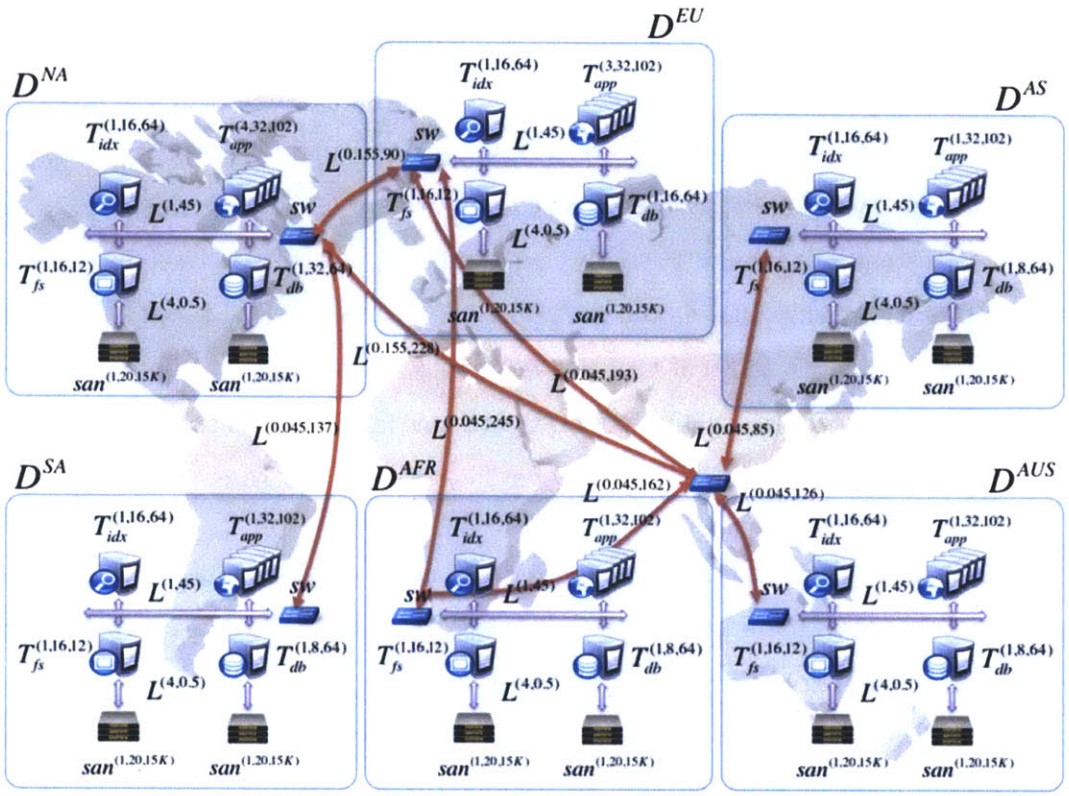

Figure 7-2: The proposed new infrastructure is comprised by six master data centers.

consistency guarantees. The platform does not require serializable transactions, but imposes stronger requirements than only eventual consistency. PNUTS provides per-record timeline consistency by requiring replica updates to be applied in order.

Amazon's Dynamo is a key-value storage system than prioritizes high-availability to guarantee that clients receive an "always-on" experience [22]. As opposed to PNUTS which provided a relaxed consistency guarantee, Dynamo only guarantees eventual consistency. Eventual consistency is achieved through a quorum-like technique and a decentralized replica synchronization protocol.

7.3 Multiple Master: Simulation Inputs

7.3.1 Infrastructure Hardware & Topology

Figure 7-2 illustrates the global network of master data centers that compose the new infrastructure. The hardware specifications for each tier in each data center and the connectivity characteristics between them are indicated using the same notation as in Section 5.2.1.

As opposed to the infrastructure in Section 6.4, this new configuration required to upgrade five data centers from SDCs to MDCs, but in return, the specifications of D^{NA} were scaled down. $T_{app} \in D^{NA}$ was reduced from eight servers to four and $T_{db} \in D^{NA}$ decreased its number of cores from 64 to 32. The rest of the data centers, except from D^{EU}, utilized a single machine for each tier with the same machine specifications for S_{app}, but a smaller database in S_{db} (eight cores as opposed to 32). D^{EU} is the second largest population and file owner in the infrastructure, and therefore, it requires three servers in T_{app} and 16 cores in S_{db}.

Memory, network and SAN storage specifications remained unaltered for the simulation of the multiple master infrastructure.

7.3.2 Software Applications, Workloads & Access Patterns

The message cascades modeling software operations and their corresponding workloads remain unchanged and are the same as in the consolidated infrastructure simulated in Chapter 6. Nevertheless, in this Section a new input parameter that reflects access patterns in different data centers is introduced. This is denoted as the *Access Pattern Matrix (APM)*. The APM indicates the percentages of ownership of the files requested by each data center. The APM matrix for the consolidated infrastructure in Chapter 6 is shown in Table 7.1. In the simulation of the consolidated infrastructure all the files accessed from any of the six data centers were owned by D^{NA}.

For the simulation of the multiple master infrastructure the Fortune 500 company measured the APM, and the results are presented in Table 7.2. It is noted from

Data Access (%)	Data Owner						Total
	D^{EU}	D^{NA}	D^{AUS}	D^{SA}	D^{AFR}	D^{AS}	
D^{EU}	0	100	0	0	0	0	100
D^{NA}	0	100	0	0	0	0	100
D^{AUS}	0	100	0	0	0	0	100
D^{SA}	0	100	0	0	0	0	100
D^{AFR}	0	100	0	0	0	0	100
D^{AS}	0	100	0	0	0	0	100

Table 7.1: Access pattern matrix for the consolidated infrastructure.

Data Access (%)	Data Owner						Total
	D^{EU}	D^{NA}	D^{AUS}	D^{SA}	D^{AFR}	D^{AS}	
D^{EU}	83.65	12.71	1.67	1.04	0.13	0.81	100
D^{NA}	15.47	81.87	1.56	0.91	0.01	0.18	100
D^{AUS}	31.24	13.72	50.28	0.18	4.35	0.23	100
D^{SA}	38.99	17.55	3.42	39.87	0.08	0.09	100
D^{AFR}	36.49	31.38	13.45	0.26	17.66	0.78	100
D^{AS}	61.00	30.45	2.39	0.85	0.04	5.27	100

Table 7.2: Access pattern matrix for the multiple master infrastructure.

this table that D^{EU} and D^{NA} own the subsets of files with the largest demand from the rest of data centers. Most of their local requests are addressed to themselves or between them. D^{AUS} and D^{SA} direct most of their requests to themselves, but with significant fractions to D^{EU} or D^{NA} and with negligible percentages between them. D^{AFR} and D^{AS} manipulate files primarily owned by the largest data centers.

7.3.3 Background Processes

The message cascades of the SR and IB operations along with the data growth information utilized in Chapter 6 remain unaltered. The main difference for the infrastructure with multiple masters is that each data center will run its own SR and IB processes exclusively for the data files they own. This is illustrated in figure 7-3. Consequently, each data center has a pair of parameters (ΔT_{SR}, ΔT_{IB}) parameters for scheduling these processes and another pair (R_{SR}^{max}, R_{IB}^{max}) for evaluating the response time of them in each data center.

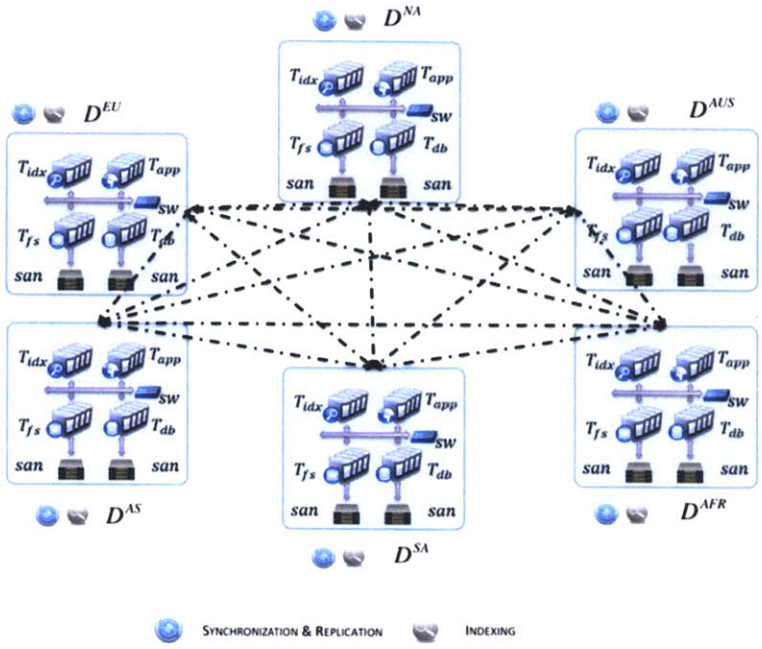

Figure 7-3: The multiple master infrastructure runs six background processes, one for each data center.

Using the data in Figure 6-10 combined with the APM presented in Table 7.2, it is possible to estimate the new data volume (MB) to be transferred during the Pull/Push phases of the SYNCHREP operations. In the new infrastructure, six SYNCHREP operations can run simultaneously. The data volume transferred by SYNCHREP in D^{NA} is illustrated in figure 7-4. It is noted that during the period 12:00-15:00 (GMT) the peak data volume transfers have been reduced from approximately 14.25 GB in the consolidated infrastructure to 8 GB. This is a reduction of 43%.

D^{EU} is the second largest data producer of the Data Serving Platform. In the new infrastructure, D^{EU} runs its own SYNCHREP operation. The data volume transferred is illustrated in Figure 7-5, and during the peak period the volume is approximately 5.5 GB.

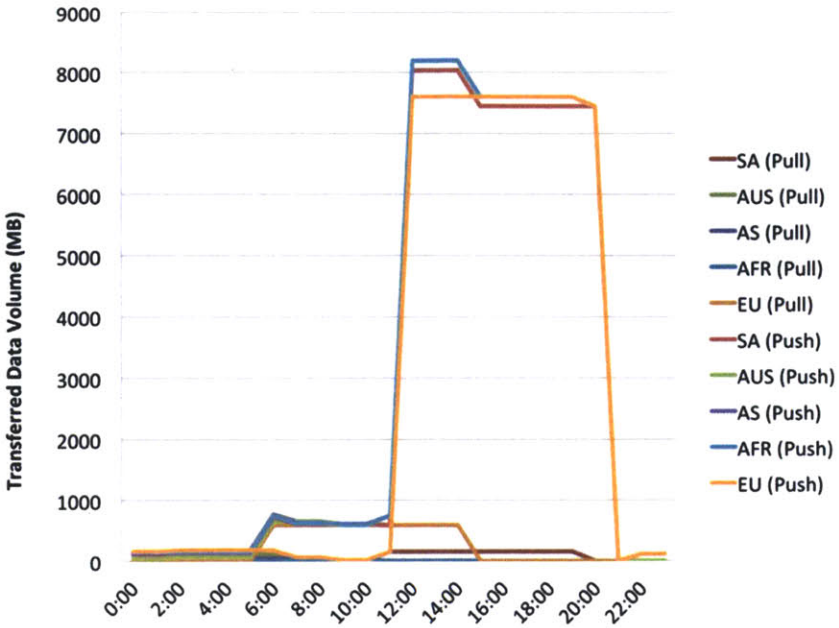

Figure 7-4: Data volume (MB) to be transferred during Pull/Push phases to/from D^{NA}

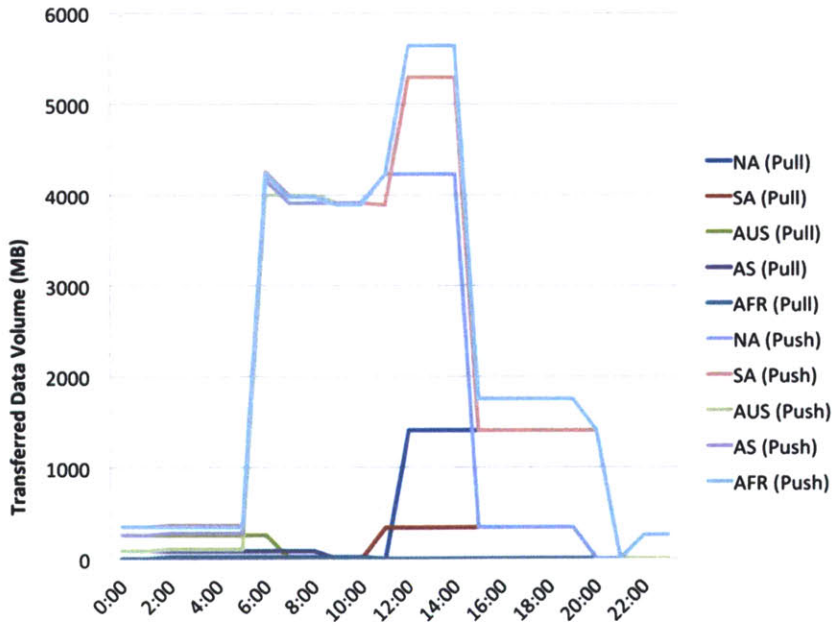

Figure 7-5: Data volume (MB) to be transferred during Pull/Push phases to/from D^{EU}

7.4 Multiple Master: Simulation Outputs

This section presents the results produced by GDISim after launching it with the information provided about the multiple master infrastructure. Analogous to Chapter 6, the computational, network, background process and client experience estimations are analyzed. Special attention is given to the impact caused by the alterations explained throughout this chapter.

7.4.1 Computational Performance Results

The benefits of running multiple master data centers are primarily observed in D^{NA}. In Chapter 6, all of the workload originating in the slave data centers directly affected D^{NA}, since all the operations were routed through this data center. Additionally, D^{NA} was responsible for the execution of background processes over the entire volume of files served by the platform. The circumstances for D^{NA} in this new infrastructure is significantly different. Two effects must be taken into account:

1. *Global Workload Offload*: The APM shown in Table 7.2 indicates that while most of the population in $D^{NA}(\sim 82\%)$ manipulates local data, the rest of the data centers address their requests only a fraction of what they used to (13-31%). More importantly, the second largest population of the platform, C^{EU}, which also overlaps with C^{NA}, only sends 12.71% of the load to D^{NA}. Consequently, it is concluded that not only does D^{NA} enjoy a significant offload of requests, but it also depends primarily on the dynamics of its own population of clients C^{NA}.

2. *Synchronization & Indexing Offload*: As it was reported in Section 7.3.3, the peak data volume to transfer by D^{NA} was reduced approximately 43%. Section 6.4.3 showed that SR and IB processes also require interactions with T_{app} and T_{db}. In this new infrastructure, the volume of transactions directed to the application servers and database servers is also significantly reduced.

For brevity, this section focuses on the impact that these two offload effects have in $T_{app} \in D^{NA}$, $T_{db} \in D^{NA}$. These two servers tiers are the cornerstone of the consolidated infrastructure in Chapter 6. In the multiple master infrastructure, during the 12:00-16:00 (GMT) interval, GDISim reported a maximum utilization of 78% in $T_{app} \in D^{NA}$ and 39% in $T_{db} \in D^{NA}$. These values are slightly superior to the measurements reported in Section 6.5 for the consolidated infrastructure, but it must be noted that $T_{app} \in D^{NA}$ is formed by half of the servers simulated in this new infrastructure and $T_{db} \in D^{NA}$ reduced the number of cores by a factor of two as well. On the contrary, the second largest data center in the infrastructure, D^{EU}, required additional computational resources. GDISim reported a CPU utilization of 57% in $T_{app} \in D^{EU}$ for a tier comprised of three servers with 32 cores each and 48% for a $T_{db} \in D^{EU}$ for a server with 16 cores.

7.4.2 Network Performance Results

Table 7.2 presents the estimates produced by GDISim for the average utilization, μ_U, of the allocated capacity (20%), for the 12:00-16:00 (GMT) interval and by network link. Even though the total volume of information to be transferred remains the same, the execution of multiple SYNCHREP operations concurrently reduced the response times and allowed transferring the same volume of information in shorter intervals of time. Transfers carried out by SYNCHREP operations originated in different data centers will share the same network links. During the 12:00-16:00 (GMT) interval, the largest data producers, D^{NA} and D^{EU} will be sharing the same resources for their respective Pull and Push phases. GDISim reported that, in general, the occupancy of the network links raised. The busiest link, $L^{NA \rightarrow AS1}$, reached 76% of occupancy (15% from total).

Given the full connectivity of the Data Serving Platform, it was proposed to the Fortune 500 company to utilize the $L^{EU \rightarrow AFR}$ and $L^{EU \rightarrow AS1}$ links, so as to route the data transfers associated to D^{EU} through these links and hence alleviate $L^{NA \rightarrow AS1}$ and $L^{AS1 \rightarrow AFR}$.

	μ_U (%)
$L^{NA \rightarrow SA}$	53
$L^{NA \rightarrow EU}$	51
$L^{NA \rightarrow AS1}$	76
$L^{EU \rightarrow AFR}$	0
$L^{EU \rightarrow AS1}$	0
$L^{AS1 \rightarrow AFR}$	67
$L^{AS1 \rightarrow AS2}$	56
$L^{AS1 \rightarrow AUS}$	66

Table 7.3: Average utilization of the allocated capacity during 12:00-16:00 (GMT) interval for each network link.

7.4.3 Background Process Performance Results

Figure 7-6 illustrates the response time prediction produced by GDISim for the Synchronization & Replication and Index Build background processes running in D^{NA}. The response time represents the duration of the operation, from launch time to its conclusion. In both cases, the duration of these processes is directly dependent to the volume of new information generated. This section focuses on D^{NA} because it is the largest data producer and hence, imposes the worst case scenario.

It was noticeable that the reduction in transferred data volumes by D^{NA} had a beneficial impact by reducing R_{SR}^{max} and R_{IB}^{max}.

The interval of the day with the largest generation of new data, 12:00-15:00 (GMT), results in the period of time with the longest response time for the SYNCHREP operation. In Figure 7-6 it can be estimated that $R_{SR}^{max} = 19$ min. Stale versions of data files will be exposed for a maximum time of approximately 19 minutes during the peak workload period of the day. Similarly, the maximum longest response time for INDEX BUILD is estimated to be $R_{IB}^{max} = 37$ min. Therefore, the maximum time an index file is outdated was also reduced to 37 min.

7.4.4 Client Experience Results

As was expected from operation below hardware saturation limits, response times for CAD, VIS and PDM operations launched from different data centers did not show any

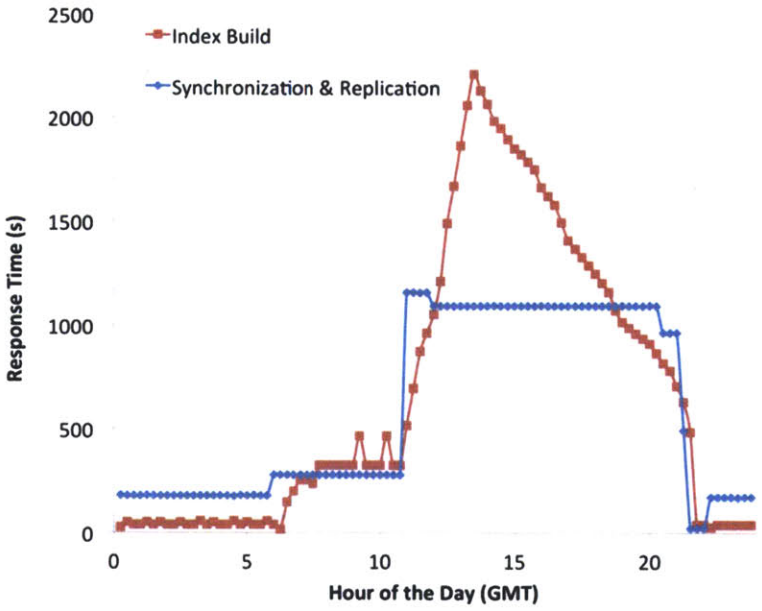

Figure 7-6: Response time of background processes (SR and IB) in D^{NA}

degradation. The response time values reported by GDISim for the multiple master infrastructure were flat and are almost identical to the ones presented in Section 6.5.4. These are omitted for simplicity.

Ideally, the possibility to carry out metadata related operations involving multiple round trips (EXPLORE, SPATIAL-SEARCH or SELECT) locally, as opposed to the single data center configuration in which metadata operations for remote data centers involved long distance communication, should have a positive impact on the effect of latency. Nevertheless, in the case of the Fortune 500 company, most of the requests originated in remote locations such as D^{AUS} and D^{AS} are addressed to data owned by D^{EU} and D^{NA}, and thus, in the majority of cases long distance round-trips are unavoidable. For these reasons, the impact of latency remained unaltered for the remote data centers in this multiple master infrastructure.

7.5 Result Evaluation

This chapter presented the GDISim evaluation of a proposed system aimed at optimizing the execution of background processes in the Data-Serving Platform presented in Chapter 6. The proposed infrastructure added file management capabilities to all data centers by upgrading these from SDCs to MDCs. Then, using the idea of data ownership, every data file was associated to a data center that would be responsible for its synchronization, replication and indexing processes.

By making every data center responsible for the execution of background processes on a smaller subset of files, the maximum stale replica and maximum outdated index timings were significantly reduced. Additionally, this distribution of responsibility allowed the redistribution of hardware infrastructure from D^{NA} to the other data centers, while maintaining the same client experience results. An increase in the network occupancy was observed, caused by the concurrent execution of synchronization operations.

While the utilization of multiple master data centers did not require modification of the relaxed consistency model for data files, the execution of the indexing process in this system relies on the acceptance of eventual consistency guarantees for the index files. Data center operators in the Fortune 500 company will be required to compromise consistency requirements in the indexing process of the Data Serving Platform, in exchange for effectiveness and performance improvements in the background processes executed.

Chapter 8

Contributions and Lessons Learned

This chapter gathers the lessons learned during the design and implementation of the Global Data Infrastructure Simulator along with conclusions derived from its application in resolving the problems that exist in the real IT infrastructure of a globalized organization.

8.1 Summary of Contributions

In this section, the contributions made by this work to the field of computer system evaluation are summarized:

1. *Horizontal & Vertical Global Infrastructure Evaluation*: As opposed to previous research, which focused on the evaluation of an isolated data center, computer system or hardware component, GDISim provides a simulation platform that enables evaluating the system not only horizontally, but also vertically at different granularities. The simulator can provide a macroscopic view of the operation and dynamics of the global IT infrastructure, but is designed to allow data center operators to navigate down to individual servers or hardware components and hence, to facilitate bottleneck detection and optimization tasks.

2. *Software Application Diversity*: The decomposition of software applications into operations and the representation of these as cascades of messages, which encode

the resource allocation inflicted on the components they flow through, provided the capability to represent arbitrary complex interactions between client software and interconnected server tiers across the globe. Data repositories are not an integral part of the software application anymore. Today, infrastructures for data collaboration separate software applications from the data itself, creating Data-Serving Platforms. These infrastructures are responsible for providing access to information with high availability and low latency for an ecosystem of decentralized software applications. GDISim is aligned with this model and enables data center operators to reproduce the behavior of concurrent software applications that manipulate the same pool of files served by a Data-Serving Platform.

3. *Background Jobs*: Geographically distributed Data-Serving Platforms are required to be consistent, fault-tolerant and capable of performing large-scale search. These characteristics are achieved by scheduling synchronization, replication and indexing processes that move, copy and analyze data files in the background, concurrently with client generated workloads. Previous research does not contemplate the impact that these processes have on resource utilization and the subsequent risk of jeopardizing client experience. GDISim enables data center operators tuning these processes to maximize their effectiveness while remaining seamless for clients manipulating the information.

4. *Simulator Validation*: GDISim was validated on a downscaled version of the real infrastructure of a Fortune 500 company running a real software application with different workload intensities. The RMSE metric was utilized for the comparison between the physical and simulated systems. The RMSE for operation response times was comparable to previous research on analytic data center modeling and simulation. Nevertheless, as opposed to other initiatives, GDISim provided details on CPU utilization of the infrastructure that resulted in a maximum deviation of 13%. Validation experiments also concluded that memory allocation models require additional sophistication in order to take into

account the effects of the Operating System and runtime environments.

5. *Platform Scalability*: Simulation time is directly related to the number of hardware components and client population that form the simulated infrastructure. During the last decade, the advent of cloud computing has made clients lighter and IT infrastructures have grown to thousands or tens of thousands of servers. Experiments with GDISim were carried out for a small IT infrastructure in a Fortune 500 company, but still simulation time was found to be significant. Nevertheless, the goal is to have the capacity to reproduce the behavior of larger infrastructures. As important as getting accurate results from a simulator is to get these results on a timely manner. For these reasons GDISim imported parallelization techniques from MAS in other fields of research so as to provide a degree of scalability with multicore.

8.2 Lessons Learned

1. *Consensus-Seeker*: The complexity of modern IT infrastructures makes it impossible for a single data center operator to understand and master all the dynamics of workload, data growth, hardware, software or network. For these reasons, the design and maintenance of an IT infrastructure is carried out by groups of IT professionals specialized in particular components of the system. Typically, these individuals do not have detailed knowledge beyond their area of responsibility. GDISim not only allows IT professionals to understand the behavior of the system in and out of their area of expertise,but also serves as a baseline from which decision makers can align their thinking and reach consensus on system alterations. The process of collecting and gathering data for the simulation of the consolidated infrastructure and the interpretation of the predictions produced by GDISim was not only useful in terms of quantitative results for the Fortune 500 company, but it was also a valuable exercise that allowed the IT professionals involved in the study to gain a broader understanding of the system.

2. *Componentized Modular Simulation*: This research did not construct new queueing models for hardware components, but utilized models designed and validated for the last two decades as a foundation for the construction of a large-scale network of queues that represents an entire IT infrastructure. This research showed that the implementation of these queueing models into components and their interconnection to create higher level entities allowed construction of a modular and flexible platform that reproduces the behavior of arbitrarily complex and different computer systems. While other approaches were purely analytical, purely profiling or purely simulation, GDISim merges the three techniques into one by implementing analytic models, feeding these with data obtained by profiling and reproducing the behavior of the system through simulation.

3. *Data Ownership*: GDISim showed that providing file management capabilities to a Data Serving Platform through multiple smaller master data centers as opposed to having a single data center responsible for all metadata related operations can be beneficial in many different ways. If the index file consistency can be relaxed, the responsibility to run synchronization, replication and index build background processes can be partitioned across data centers by file ownership. A data center owns a file if most of the activity pertaining to that file is originated by the clients local to this data center. Distribution of file management responsibility among data centers has beneficial consequences, such as, smaller footprints for master data centers, better utilization of networks and minimization of the maximum stale replica and outdated index intervals.

Chapter 9

Future Work

The goal of the GDISim simulation platform was to facilitate the answering of questions related to hypothetical design scenarios for a global IT infrastructure. This was achieved for the infrastructure of a Fortune 500 company. However the design and development of this simulator also revealed a number of unknowns and research directions that could be investigated further. The goal of this chapter is to briefly describe these potential future research initiatives that could further develop GDISim. These initiatives are divided into three groups, *Hardware Modeling*, *Software Modeling* and *Simulation Platform.*

9.1 Hardware Modeling

9.1.1 Multithreading

The software applications modeled in this research launched operations that were executed on a single thread in each server. Pull and Push phases transferred data from/to different data centers in parallel, but launching a single thread in each location. Hence, GDISim did not contemplate the possibility of using multiple threads for the execution of a single operation. The CPU model presented in Section 3.4.2 assigned a single thread per processing task queued in the component. Future work could consider the possibility of combining queue-servers dynamically for the execu-

tion of multiple threads per processing task and hopefully reproduce the performance boost provided by multithreaded platforms.

9.1.2 Cache Hierarchy

GDISim modeled caching effects by enabling data center operators to specify percentages of cache hits that would allow operations bypassing computation or I/O steps. Nevertheless, these models do not reflect the specifications of the underlying memory hierarchy of the CPU or the NAS and SAN controllers. Additionally, caching hierarchy has proven to be beneficial under some circumstances, as indicated by Kazempour et al. [47], particularly for multithreaded execution. For these reasons, in the future computational and I/O models should reflect the particularities of caching hierarchy in the hardware modeled.

9.2 Software Modeling

9.2.1 Client Behavior

The simulation of the infrastructure of the Fortune 500 company was carried out assuming that clients launched operations independently and that the probability distribution of operation types remained uniform throughout the day. Even though for simulation purposes these assumptions might be sufficient, neither of these assumptions follow real client behavior.

As explained in Section 3.5, clients may choose to launch a specific operation type with higher probability depending on the time of the day. For example, it is likely that most of the client population will be logging into the system in the morning to begin their workday.

Additionally, operations are not independent of each other and the probability to launch an operation may depend on the history of previous operations launched. For example, after a client carries out a text search and an exploration of the tree structure of parts, it is likely that the chosen part will be opened. Similarly, after a

spatial-search and a 3D selection of parts, it is likely that the chosen parts will be downloaded.

For these reasons, in future work clients should carry an identity and launch operations based on their previous activity and time of the day. Given a sample of real client activity throughout the day, data can be partitioned into uniform intervals and for each partition a Markov Chain can be constructed to produce the matrix of initial and transition probabilities for that specific interval of the day. Transition probabilities indicate the probability of launching an operation based on the previous activity. Markov chains are a popular tool for modeling internet user behavior [9] [62] [32].

9.2.2 Operating System & Runtime Impact

In this research, the impact of the OS and runtime environments on the utilization of hardware resources was assumed to be negligible. Nevertheless, depending on the OS and runtime configuration chosen by data center operators this might not be the case. Multiple processes with different responsibilities can be scheduled to be executed while the system is being loaded with clients. This can have an impact on the performance of the system and subsequently, on the experience of clients. Examples of these processes are: software updates, scheduled security software runs, scheduled defragmentation, etc. Typically, it is preferred to configure the OS to execute these processes manually as it is found convenient by the operator, rather than automatically.

Similarly, Section 5.3.3 emphasized the impact of memory pools in the memory utilization of the servers while these are under zero or minimal load conditions. In order to model memory utilization of different software applications accurately, it is necessary to construct models that incorporate the details of memory allocation in the kernel and in the runtime. Additionally, in some cases Garbage Collectors (GC) can pause the execution of all application threads. This is also an effect that should be taken into account when considering CPU utilization.

Future research should account for the impact of these OS and runtime effects if

they are found to interfere with the software applications being modeled. For this purpose, the construction of a configurable OS model with its corresponding set of operations is proposed. Data center operators will be able to enable/disable these effects by passing these as simulator inputs analogous to other software applications.

9.2.3 File Identity

GDISim does not assign identities to files and assumes that all files have the same importance. However, in practice some files will have more demand than others and multiple clients may collide when requesting the same information simultaneously. If the volume of requests for an object produces a spike, this object is denoted as a *Hotspot* or *Hot Object*. Bodik et al. worked on the characterization and modeling of these objects [12].

The existence of Hotspots has positive and negative implications. It is beneficial to have the possibility to cache the object in memory and accelerate access to the information for clients. Similarly, the object can be replicated across machines in the same T_{fs} so as to increase its availability. Unfortunately, the increased demand also increases the probability of conflicts when committing file updates. The larger the time a replica of a hot object remains stale, the higher the probability of collisions will be. Under these circumstances, it is critical to keep the R_{SR}^{max} and R_{IB}^{max} values for Hot Objects to a minimum.

The introduction of file identity in GDISim would open the possibility of simulating and understanding a relevant space of problems that might be of interest to data center operators and critical for any organization.

9.3 Simulation Platform

9.3.1 Cross-Machine Scalability

GDISim was designed to obtain acceleration from multithreaded execution and scale with the number of cores. Nevertheless, GDISim targeted organizations in which

IT infrastructure was not the core of the business itself and its footprint was small compared to the tens of thousands of servers running in internet-scale organizations. In order to simulate the infrastructures in these companies, it is necessary for GDISim to be able to run in a cluster.

The port abstraction in Port-Based programming does not impose a communication protocol between agents. An agent sending a message to the port of another agent may choose the communication protocol based on the location of the receiver and its own location. Agents sharing the same machine can communicate through cross-thread exchange of pointers to locations in shared memory, while agents sitting on different machines can communicate via MPI or web protocols. An implementation of communication between ports distributed across local networks or the internet is given by Decentralized Software Services (DSS) [40] a runtime environment that sits on top of the CCR library that was introduced in Section 4.2.3.

Another possibility for the execution of GDISim in a cluster is the utilization of a cluster technology called *Virtualization for Aggregation.* This technology provides a hypervisor that combines commodity x86 servers to create a virtual symmetric multiprocessing system. A commercial example of this technology is ScaleMP [77]. Virtualization for aggregation combines the resources of multiple servers and makes these available as a single aggregated machine with a single Operating System sitting on top of a hypervisor. This setup allows GDISim to run on a larger number of cores than is available in a single machine. Nevertheless, further optimization of the task dispatcher mechanism introduced in Section 4.3.5 would be required in order to get performance improvements in spite of cross-machine communication.

9.3.2 Visualization, Restoration Points & Branches

Currently, GDISim does not have a visualization component that enables viewing the state of the infrastructure as the simulation progresses. The collector component measures the state of every component and writes the data to the disk for post-processing. Construction of charts and tables is executed automatically offline upon termination of the simulation.

In addition to the possibility of visualizing and evaluating the state of the simulated infrastructure through graphs and charts in runtime, it would be valuable to provide the capability to create *Restoration Points*. Restoration points are snapshots of the entire infrastructure at specific time steps. These snapshots do not only register the averaged utilization and occupancy of the components but also contain the location and contents of the messages flowing across the infrastructure. Restoration Points provide the capability of navigating simulation time backwards and replaying intervals of the day with alterations of the input parameter set. These modified simulation paths are called *Branches*. Branches enhance the capabilities of the simulator by enabling data center operators to introduce unexpected events at any point of the simulation and compare different paths, for example, reproducing failures in servers, incorrect configurations or broken network links in a situation of peak workload.

9.4 Concluding Remarks

Evaluation techniques for computer systems has been an active field of study for the last four decades and with high probability will continue to be as systems become larger, more heterogeneous and complex. In the past, many research initiatives in this field were created to model accurately a single component, software or situation for a specific purpose. Even if these succeeded in their mission, in most of the cases the initiative was finalized without leaving open paths for further application or study. In this dissertation, GDISim is presented as a platform that interconnects many of these older initiatives to create a large-scale yet flexible and modular simulator that reproduces the behavior of IT infrastructures. The author of this work believes that in the same way that the creation of civil infrastructures is supported by the utilization of simulators, the operation of IT infrastructures will not only consider tools like GDISim valuable but eventually necessary.

Appendix A

Notation

A.1 Kendall's Notation

In queueing theory, Kendalls notation is the standard system used to describe and classify the queueing model that a queueing system corresponds to [49]. Initially, it was presented as a three factor notation system $A/B/C$, but later versions have enhanced the notation to include three additional factors $A/B/C/K/N - D$.

- A: represents the *arrival process.* Most common distributions are Markovian (M), Erlang (E_k) and General (G).
- B: represents the *service time distribution.* Most common distributions are Markovian (M), Erlang (E_k), Degenerate (D) and General (G).
- C: represents the *number of service channels.*
- K: represents the *total capacity* of the system. If the number is omitted it is assumed to be unlimited (∞).
- N: represents the *size of the source.* If the number is omitted it is assumed to be unlimited (∞).
- D: represents the *queueing discipline.* Commonly used disciplines are First Come First Served ($FCFS$), Priority ($PNPN$) and Processor Sharing (PS_k, where k is maximum number of simultaneous customers).

In this dissertation $A/B/C - D$ will be used since K and N are assumed to be ∞.

Bibliography

[1] D. Agrawal, M. Choy, H. Va Leong, and A. K. Singh, "Maya: a simulation platform for distributed shared memories," *SIGSIM Simul. Dig.*, vol. 24, pp. 151–155, July 1994.

[2] J. Akella, H. Buckow, and S. Rey, "IT architecture: Cutting costs and complexity ," *McKinsey Quarterly*, Aug. 2009.

[3] A. Allen, "Queueing models of computer systems," *Computer*, vol. 13, no. 4, pp. 13 –24, april 1980.

[4] O. Almer, I. Bohm, T. von Koch, B. Franke, S. Kyle, V. Seeker, C. Thompson, and N. Topham, "Scalable multi-core simulation using parallel dynamic binary translation," in *Embedded Computer Systems (SAMOS), 2011 International Conference on*, july 2011, pp. 190 –199.

[5] J. M. Anderson, L. M. Berc, J. Dean, S. Ghemawat, M. R. Henzinger, S.-T. A. Leung, R. L. Sites, M. T. Vandevoorde, C. A. Waldspurger, and W. E. Weihl, "Continuous profiling: where have all the cycles gone?" *ACM Trans. Comput. Syst.*, vol. 15, pp. 357–390, November 1997.

[6] T. Austin, E. Larson, and D. Ernst, "Simplescalar: an infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59 –67, feb 2002.

[7] "Autocad," AutoDesk,Inc, 2011. [Online]. Available: http://usa.autodesk.com/autocad/

[8] Y. Bard, "An analytic model of the vm/370 system," *IBM Journal of Research and Development*, vol. 22, no. 5, pp. 498 –508, sept. 1978.

[9] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida, "Characterizing user behavior in online social networks," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, ser. IMC '09. New York, NY, USA: ACM, 2009, pp. 49–62.

[10] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

[11] J. Bi, Z. Zhu, R. Tian, and Q. Wang, "Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, july 2010, pp. 370 –377.

[12] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "Characterizing, modeling, and generating workload spikes for stateful services," in *Proceedings of the 1st ACM symposium on Cloud computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 241–252.

[13] J. M. Bradshaw, *An introduction to software agents.* Cambridge, MA, USA: MIT Press, 1997, pp. 3–46.

[14] R. M. Brown, J. C. Browne, and K. M. Chandy, "Memory management and response time," *Commun. ACM*, vol. 20, pp. 153–165, March 1977.

[15] J. P. Buzen, "Queueing network models of multiprogramming," Ph.D. dissertation, Harvard University, AUG 1971.

[16] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, pp. 23–50, January 2011.

[17] E. Cecchet, A. Chanda, S. Elnikety, J. Marguerite, and W. Zwaenepoel, "Performance comparison of middleware architectures for generating dynamic web content," in *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, ser. Middleware '03. New York, NY, USA: Springer-Verlag New York, Inc., 2003, pp. 242–261.

[18] G. Chrysanthakopoulos and S. Singh, "An asynchronous messaging library for c sharp," in *Synchronization and Coordination in Object Oriented Languages*, 2005.

[19] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "Pnuts: Yahoo!'s hosted data serving platform," *Proc. VLDB Endow.*, vol. 1, pp. 1277–1288, August 2008.

[20] "Computer aided three-dimensional interactive application (catia)," Dassault Systemes, 2011. [Online]. Available: http://www.3ds.com/products/catia

[21] J. Dean, "Large-scale distributed systems at google: Current systems and future directions," Keynote presentation at LADIS 2009, 2009.

[22] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 205–220, October 2007.

[23] P. Dickens, P. Heidelberger, and D. Nicol, "Parallelized direct execution simulation of message-passing parallel programs," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 7, no. 10, pp. 1090 –1105, oct 1996.

[24] K. Dixon, T. Pham, and P. Khosla, "Port-based adaptable agent architecture," in *Self-Adaptive Software*, ser. Lecture Notes in Computer Science, P. Robertson, H. Shrobe, and R. Laddaga, Eds. Springer Berlin / Heidelberg, 2001, vol. 1936, pp. 181–198.

[25] R. P. Doyle, J. S. Chase, O. M. Asad, W. Jin, and A. M. Vahdat, "Model-based resource provisioning in a web service utility," in *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, ser. USITS'03. Berkeley, CA, USA: USENIX Association, 2003, pp. 5–5.

[26] F. L. Fessant and L. Maranget, "Compiling join-patterns," *Electronic Notes in Theoretical Computer Science*, vol. 16, no. 3, pp. 205 – 224, 1998.

[27] B. Gates, *Business at the Speed of Thought: Using a Digital Nervous System.* New York, NY, USA: Warner Books, Inc., 1998.

[28] N. Gaud, S. Galland, V. Hilaire, and A. Koukam, "An organisational platform for holonic and multiagent systems," in *Programming Multi-Agent Systems*, ser. Lecture Notes in Computer Science, K. Hindriks, A. Pokahr, and S. Sardina, Eds. Springer Berlin / Heidelberg, 2009, vol. 5442, pp. 104–119.

[29] D. P. Gaver, Jr., "Probability models for multiprogramming computer systems," *J. ACM*, vol. 14, pp. 423–438, July 1967.

[30] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. Cameron, "Powerpack: Energy profiling and analysis of high-performance systems and applications," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 5, pp. 658 –671, may 2010.

[31] G. Gilder, "Fiber keeps its promise: Get ready, bandwidth will triple each year for the next 25," *Forbes*, April 7, 1997.

[32] V. Gopalakrishnan, R. Jana, R. Knag, K. K. Ramakrishnan, D. F. Swayne, and V. A. Vaishampayan, "Characterizing interactive behavior in a large-scale operational iptv environment," in *Proceedings of the 29th conference on Information communications*, ser. INFOCOM'10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 246–250.

[33] V. Gorodetski and I. Kotenko, "The multi-agent systems for computer network security assurance: frameworks and case studies," in *Artificial Intelligence Systems, 2002. (ICAIS 2002). 2002 IEEE International Conference on*, 2002, pp. 297 – 302.

[34] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini, "Statistical profiling-based techniques for effective power provisioning in data centers," in *Proceedings of the 4th ACM European conference on Computer systems*, ser. EuroSys '09. New York, NY, USA: ACM, 2009, pp. 317–330.

[35] S. L. Graham, P. B. Kessler, and M. K. Mckusick, "Gprof: A call graph execution profiler," *SIGPLAN Not.*, vol. 17, pp. 120–126, June 1982.

[36] J. Gray and G. Graefe, "The five-minute rule ten years later, and other computer storage rules of thumb," *SIGMOD Rec.*, vol. 26, pp. 63–68, December 1997.

[37] J. Gray and P. Shenoy, "Rules of thumb in data engineering," in *Proceedings of the 16th International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2000, pp. 3–.

[38] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 68–73, December 2008.

[39] P. Helland, "Life beyond distributed transactions: an apostate's opinion," in *CIDR*, 2007, pp. 132–141.

[40] G. C. Henrik Frystyk Nielsen, "Decentralized software services protocol dssp/1.0," Microsoft Corporation.

[41] D. W. Holmes, J. R. Williams, and P. Tilke, "An events based algorithm for distributing concurrent tasks on multi-core architectures," *Computer Physics Communications*, vol. 181, no. 2, pp. 341–354, Feb. 2010.

[42] Y.-Q. Huang, H.-L. Li, X.-H. Xie, L. Qian, Z.-Y. Hao, F. Guo, and K. Zhang, "Archsim: A system-level parallel simulation platform for the architecture design of high performance computer," *Journal of Computer Science and Technology*, vol. 24, pp. 901–912, 2009.

[43] "Intel vtune performance analyser," Intel Corporation, 2010. [Online]. Available: http://software.intel.com/en-us/ intel-vtune/

[44] "Intel parallel amplifier 2011," Intel Corporation, 2011. [Online]. Available: http://software.intel.com/en-us/articles/intel-parallel-amplifier/

[45] A. Kansal and F. Zhao, "Fine-grained energy profiling for power-aware application design," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, pp. 26–31, August 2008.

[46] S. Karnouskos and M. M. J. Tariq, "An agent-based simulation of soa-ready devices," in *Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on*, april 2008, pp. 330 –335.

[47] V. Kazempour, A. Fedorova, and P. Alagheband, "Performance implications of cache affinity on multicore processors," in *Euro-Par 2008 Parallel Processing*, ser. Lecture Notes in Computer Science, E. Luque, T. Margalef, and D. Bentez, Eds. Springer Berlin / Heidelberg, 2008, vol. 5168, pp. 151–161.

[48] R. W. Kembel, *Fibre Channel: A Comprehensive Introduction.* Northwest Learning Associates, inc, 2009.

[49] D. G. Kendall, "Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain," *The Annals of Mathematical Statistics*, vol. Vol. 24, no. No. 3, pp. 338–354, Sept. 1953.

[50] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, pp. 41–50, January 2003.

[51] L. Kleinrock, "Analysis of a time-shared processor," *Naval Research Logistics Quarterly*, vol. 11, no. 1, pp. 59–73, 1964.

[52] ——, "Time-shared systems: a theoretical treatment," *J. ACM*, vol. 14, pp. 242–261, April 1967.

[53] S. Y. Ko, R. Morales, and I. Gupta, "New worker-centric scheduling strategies for data-intensive grid applications," in *Proceedings of the 8th ACM/IFIP/USENIX international conference on Middleware*, ser. MIDDLEWARE2007. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 121–142.

[54] A. Koestler, *The ghost in the machine.* Hutchinson, London, 1976.

[55] S. Kounev, "Performance modeling and evaluation of distributed component-based systems using queueing petri nets," *Software Engineering, IEEE Transactions on*, vol. 32, no. 7, pp. 486 –502, july 2006.

[56] S. Kumar, "How project triforce prepared our software stack for prineville," Facebook, May 2011.

[57] K. Langendoen, R. Bhoedjang, and H. Bal, "Models for asynchronous message handling," *Concurrency, IEEE*, vol. 5, no. 2, pp. 28 –38, apr-jun 1997.

[58] R. Lantz,. "Parallel simos: Scalability and performance for large system simulation." Ph.D. dissertation., Stanford University, 2007.

[59] A. S. Lebrecht, N. J. Dingle, and W. J. Knottenbelt, "Analytical and simulation modelling of zoned raid systems," *Comput. J.*, vol. 54, pp. 691–707, May 2011.

[60] J. Levon and P. Elie, "Oprofile - a system profiler for ulinux." [Online]. Available: http://oprofile.sourceforge.net/doc/index.html

[61] S.-H. Lim, B. Sharma, G. Nam, E. K. Kim, and C. Das, "Mdcsim: A multi-tier data center simulation, platform," in *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, 31 2009-sept. 4 2009, pp. 1 –9.

[62] Y. Liu, B. Gao, T.-Y. Liu, Y. Zhang, Z. Ma, S. He, and H. Li, "Browserank: letting web users vote for page importance," in *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '08. New York, NY, USA: ACM, 2008, pp. 451–458.

[63] J. McCarthy, "Recursive functions of symbolic expressions and their computation by machine, part i," *Commun. ACM*, vol. 3, pp. 184–195, April 1960.

[64] D. Menasce, "Web server software architectures," *Internet Computing, IEEE*, vol. 7, no. 6, pp. 78 – 81, nov.-dec. 2003.

[65] L. Meyer, J. Annis, M. Wilde, M. Mattoso, and I. Foster, "Planning spatial workflows to optimize grid performance," in *Proceedings of the 2006 ACM symposium on Applied computing*, ser. SAC '06. New York, NY, USA: ACM, 2006, pp. 786–790.

[66] J. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, jan. 2010, pp. 1 –12.

[67] W. J. Mitchell, *City of bits: space, place, and the infobahn.* Cambridge, MA, USA: MIT Press, 1995.

[68] M. Niazi and A. Hussain, "Agent-based tools for modeling and simulation of self-organization in peer-to-peer, ad hoc, and other complex networks," *Communications Magazine, IEEE*, vol. 47, no. 3, pp. 166 –173, march 2009.

[69] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers, "Flexible update propagation for weakly consistent replication," *SIGOPS Oper. Syst. Rev.*, vol. 31, pp. 288–301, October 1997.

[70] "Creo elements/pro," PTC, 2011. [Online]. Available: http://www.ptc.com/products/creo-elements-pro/

[71] K. Ramachandran and B. Sikdar, "A queuing model for evaluating the transfer latency of peer-to-peer systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 3, pp. 367 –378, march 2010.

[72] S. Ranjan, J. Rolia, H. Fu, and E. Knightly, "Qos-driven server migration for internet data centers," in *Quality of Service, 2002. Tenth IEEE International Workshop on*, 2002, pp. 3 – 12.

[73] G. Ren, E. Tune, T. Moseley, Y. Shi, S. Rus, and R. Hundt, "Google-wide profiling: A continuous profiling infrastructure for data centers," *IEEE Micro*, vol. 30, pp. 65–79, 2010.

[74] S. Rodriguez, N. Gaud, V. Hilaire, S. Galland, and A. Koukam, "An analysis and design concept for self-organization in holonic multi-agent systems," in *Engineering Self-Organising Systems*, ser. Lecture Notes in Computer Science, S. Brueckner, S. Hassas, M. Jelasity, and D. Yamins, Eds. Springer Berlin / Heidelberg, 2007, vol. 4335, pp. 15–27.

[75] M. Rosenblum, E. Bugnion, S. Devine, and S. A. Herrod, "Using the simos machine simulator to study complex computer systems," *ACM Trans. Model. Comput. Simul.*, vol. 7, pp. 78–103, January 1997.

[76] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta, "Complete computer system simulation: The simos approach," *IEEE Parallel Distrib. Technol.*, vol. 3, pp. 34–43, December 1995.

[77] ScaleMP, 2011. [Online]. Available: http://www.scalemp.com/

[78] R. Schaller, "Moore's law: past, present and future," *Spectrum, IEEE*, vol. 34, no. 6, pp. 52 –59, jun 1997.

[79] C. Shallahamer, *Forecasting Oracle Performance.* Berkely, CA, USA: Apress, 2007.

[80] R. Shoup, "ebay's architecture principles," QCON London, March 2008.

[81] H. Simitci, "An analytical throughput model for bulk data transfers over wide area networks," in *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, april 2004, pp. 602 – 609.

[82] M. Steenstrup, M. A. Arbib, and E. G. Manes, "Port automata and the algebra of concurrent processes," *Journal of Computer and System Sciences*, vol. 27, no. 1, pp. 29 – 50, 1983.

[83] C. Stewart and K. Shen, "Performance modeling and system management for multi-component online services," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 71–84.

[84] D. Stewart and P. Khosla, "The chimera methodology: designing dynamically reconfigurable real-time software using port-based objects," in *Object-Oriented Real-Time Dependable Systems, 1994. Proceedings of WORDS 94., First Workshop on*, oct 1994, pp. 46 –53.

[85] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "Analytic modeling of multitier internet applications," *ACM Trans. Web*, vol. 1, May 2007.

[86] E. Varki, "Mean value technique for closed fork-join networks," *SIGMETRICS Perform. Eval. Rev.*, vol. 27, pp. 103–112, May 1999.

[87] D. Villela, P. Pradhan, and D. Rubenstein, "Provisioning servers in the application tier for e-commerce systems," *ACM Trans. Internet Technol.*, vol. 7, February 2007.

[88] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser, "Active messages: a mechanism for integrated communication and computation," *SIGARCH Comput. Archit. News*, vol. 20, pp. 256–266, April 1992.

[89] N. C. Wilhelm, "A general model for the performance of disk systems," *J. ACM*, vol. 24, pp. 14–31, January 1977.

[90] M. Woolridge and M. J. Wooldridge, *Introduction to Multiagent Systems.* New York, NY, USA: John Wiley & Sons, Inc., 2001.

[91] X. Zhang, Z. Wang, N. Gloy, J. B. Chen, and M. D. Smith, "System support for automatic profiling and optimization," *SIGOPS Oper. Syst. Rev.*, vol. 31, pp. 15–26, October 1997.