NORTHWESTERN UNIVERSITY

Components of a Scalable Distributed Relational Information Service

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Dong Lu

EVANSTON, ILLINOIS

December 2005

Thesis Committee

Peter A. Dinda, Northwestern University, Committee Chair
Fabian E. Bustamante, Northwestern University
Yan Chen, Northwestern University
Ian Foster, Argonne National Laboratory and the University of Chicago

©copyright by Dong Lu 2005
All Rights Reserved

ABSTRACT

Components of a Scalable Distributed Relational Information Service

Dong Lu

An information service stores information about the resources and services within a distributed computing environment and answers queries about it. This dissertation presents the design of several important components of the Relational Grid Information System (RGIS).

The query rewriting component handles the challenges originating from the powerful features of relational algebra - some complex queries may be too expensive for RGIS. We have developed several query rewriting techniques to trade off the query time with the number of results returned. These techniques make it feasible to provide the flexibility and power of a relational data model in a GIS system while controlling the costs.

The topology generation and annotation component helps to evaluate the performance of such information servers. GridG is a synthetic grid generator that can generate annotated Internet topologies including routers, IP links, and end systems. GridG is the first synthetic grid generator that follows the power laws of Internet topology while maintaining a clear hierarchical network structure. We also discovered interesting relationships among the power laws.

The scheduling component helps to enhance system performance by minimizing mean response time. We studied the performance of size-based scheduling policies

as a function of correlation between job sizes and their estimations, and explored several estimators. We have learned that it is feasible to deploy size-based scheduling on RGIS servers.

A centralized relational information RGIS server can not scale with the distributed computing environment. A distributed RGIS can potentially scale by forming an overlay network and sharing load among the servers. Weak consistency needs to be maintained among the RGIS replicas with a content delivery network propagating updates. For efficiency and scalability, we designed a novel overlay multicast protocol that emulates the fat-tree architecture on the WAN [1].

The transfer prediction component is responsible for predicting the update transfer time accurately in a real time manner on the overlay network. We developed a novel TCP flow rate monitoring and prediction framework that can predict serial as well as parallel TCP flow rates accurately with low overhead.

Our next step is to integrate these components into RGIS.

[1]This work was done in collaboration with Stefan Birrer and Fabian Bustamante

Acknowledgments

I am deeply grateful to my advisor Dr. Peter A. Dinda. First, he believed in my potential and gave me the chance to start my career in Computer Science. Second, he taught me how to become a good graduate student and later a good scholar. Third, he has been so open-minded to support and advise me on almost every research topic that I am interested in. Finally, his endless encouragement and guidance throughout the past several years makes my Ph.D. study so exciting and fruitful.

I am also thankful to my thesis committee members– Dr. Fabian E. Bustamante, Dr. Yan Chen and Dr. Ian Foster for their kind help and guidance in developing my research agenda and completing this dissertation.

My thesis work was significantly benefited from the various collaborations with other faculty members and my fellow students. In particular, I would like to thank Dr. Fabian Bustamante, Dr. Yan Chen, Yi Qiao, Stefan Birrer, Huanyuan Sheng, Bin Lin and Jason Skicewicz. In addition, I would like to acknowledge other members of the Northwestern system research group for various discussions and communications. Also, I want to thank my officemate Kevin Livingston, for numerous interesting conversions on research topics outside system area.

I would like to express my earnest gratitude to my great parents Bing Lu and Jihong Liu, my lovely wife Yanping Xu for their love and support. I can't achieve anything without them.

Last but not the least, I would like to thank Andrew Weinreich, who took the lead in developing the RGIS web interfaces, and Jack Lange, who implemented much of the inter-RGIS server protocol.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

An information service stores information about the resources of a distributed computing environment and answers questions about it. One example of Internet-based information service that people use every day is the Domain Name Service (DNS). DNS is a system that stores information about host names and domain names on networks. Most importantly, DNS provides an IP address for each host name, and lists the mail exchange servers accepting e-mail for each domain.

Grid computing [96, 102] aims at providing dependable, consistent, pervasive, unlimited computing power and services to the users, in a manner similar to today's electrical power grid. As grids evolve from cluster to enterprise to global grids – from single departments to multiple departments to outside the firewall – grid computing will provide seamless, transparent, secure access to IT resources such as hardware, software, scientific instruments, and services. While the primary adopters of grid technology today are in compute intensive research of various kinds – the benefits of grid computing have broad appeal.

As the scale and diversity of the resources, applications, and users involved in grid

computing continues to increase quickly, the amount of information needed to keep track of them grows commensurately. Simultaneously, applications need to pose and answer increasingly powerful queries over this information in order to exploit Grid resources well and satisfy users. Grid Information Service (GIS) systems provide this functionality. The possible models and the design space for GIS systems is large. A highly scalable distributed GIS is the target application of this dissertation.

Our view of a GIS is that it is a database (in the generic sense of the word) of information about the entities within a wide area distributed high performance computing environment. Examples of Grid entities include organizations, people, computational resources (workstations, parallel computers, clusters), communications resources (switches, routers, topologies), services, benchmarks, software, event channels, sensors, scientific instruments, and others. A GIS consists of a set of objects that represent these entities, relationships between objects, and systems needed to query and update the objects and relationships. Each object has a unique identifier, a timestamp, and a set of attributes. Updates to the database take the form of additions or deletions of objects and of changes to the attributes of existing objects. The GIS makes updates available to queries as soon as possible. It also manages access to the objects, making sure that they are updated and read only by valid users. It may present different views of the objects to different users. A more detailed description of this view of a GIS is available elsewhere [179].

We are developing a GIS system, RGIS, that is based on the relational data model. Specifically, RGIS servers are implemented on top of the Oracle RDBMS and use SQL as their query language. Oracle is not a requirement of our approach—other RDBMSes could also be used. RGIS focuses on modeling the hardware and software resources of a distributed computing environment. Information streams from dynamic resource monitoring and prediction tools such as RPS [78] are currently

outside of the scope of RGIS, although RGIS does model the existence and location of such tools.

This thesis focuses on the design and implementation of several important components for the RGIS system. These include a synthetic grid information generator, query rewriting techniques, job scheduling, serial and parallel TCP throughput monitoring and prediction, and end-system multicast. These components have value beyond the RGIS system because they are generic research work that can be applied in the design of other distributed systems.

Query rewriting component: A powerful feature of RGIS is that users can write queries in SQL that search for complex compositions of resources, such as groups of hosts and network resources, that meet collective requirements. These queries can be very expensive to execute, however. In response, we have introduced three extensions to the SQL select statement that we call nondeterministic query, scoped query and approximate query. In essence, the three query extensions allow the user (and RGIS) to trade off between the running time of a query (and the load it places on an RGIS server) and the number of results returned. The result set is a random sample of the result set of the deterministic version of the query, which we argue is sufficient and appropriate for a GIS. We implement the queries using a combination of query rewriting, schema extensions, indices, and randomness. No changes to the RDBMS are needed. The three query techniques make it possible to ask complex questions of RGIS and get useful responses quickly.

Generating synthetic grids: Designing and evaluating grid middleware such as RGIS demands realistic workloads. Our query rewriting techniques allow us to trade off between the number of nondeterministically chosen results returned by the query and the amount of work done in support of it. This tradeoff depends strongly on the structure of the grid: the network topology and the characteris-

tics of the hosts, routers, and links within the topology. As the grids are still in their early stages, there is limited available data on the structure of computational grids. We examined the contents of several running GIS systems. The largest dataset we have found contains fewer than one thousand nodes. Given the limited data sets, a synthetic grid generator is a necessity. In response to this need, we built GridG, a synthetic grid generator, which can be downloaded from http://www.cs.northwestern.edu/~urgis/GridG. While developing GridG, we discovered interesting relationship among the four power-laws of Internet topology. GridG is the first topology generator that follows the power-laws while maintaining clear hierarchical network structure. Also, to my best knowledge, GridG is the first topology generator that can annotate the topology with sensible attributes at the end-system level.

Scheduling with inaccurate job size information: Effective query scheduling is essential for the performance of RGIS servers. Although size-based scheduling policies such as SRPT have been studied since 1960s and have been applied in various arenas including packet networks and web server scheduling. SRPT has been proven to be optimal in the sense that it yields—compared to any other conceivable strategy—the smallest mean value of occupancy and therefore also of waiting and delay time. One important prerequisite to applying size-based scheduling is to know the sizes of all jobs in advance, which are unfortunately not always available. No work has been done to study the performance of size-based scheduling policies when only inaccurate scheduling information is available. In the thesis, we study the performance of SRPT and FSP as a function of the correlation coefficient between the actual job sizes and estimated job sizes. We developed a simulator that supports both M/G/1/m and G/G/n/m queuing models. The simulator can be driven by trace data or synthetic data produced by a workload generator we have developed

that allows us to control the correlation. The simulations show that the degree of correlation has a dramatic effect on the performance of SRPT and FSP and that a reasonably good job size estimator will make both SRPT and FSP outperform PS in both mean response time and slowdown. We then explored several job size estimators that support size-based scheduling on web servers and the server side of P2P systems. Our work shows that it is feasible to apply size-based scheduling in RGIS servers.

Centralized VS. Distributed: A centralized information server can't scale with the growth of the distributed system because the server's CPU, memory or disk can potentially become performance bottlenecks. Even if we use a high-performance cluster to host the information service, the outgoing bandwidth can potentially become the bottleneck as the distributed system grows and more and more clients try to get query results back via the outgoing link. Our solution for better scalability of the information service is to replicate and distribute the information servers to geologically different locations on the Internet and keep weak consistency among the replicas. The servers share the query processing load and the network is not likely to become bottleneck. RGIS servers do not talk directly with each other, but indirectly via a content delivery network (CDN), which is based on the publish/subscribe model and is used solely to propagate updates to friendly RGIS servers.

TCP throughput monitoring and prediction component is responsible for predicting the data transfer time accurately in a real time manner on the overlay network. We have prototyped DualPats, a novel TCP flow rate monitoring and prediction framework which can predict serial as well as parallel TCP flow rate accurately with low overhead.

DualPats is designed with two goals in mind. First, it can be used by the RGIS servers to monitor TCP throughput between them, which helps to soft time bound the

consistency control by accurately estimating the updates transferring time. Second, it can be used to monitor the TCP throughput between the distributed computing resources.

In this work, we first exploits the strong correlation between TCP throughput and flow size, and the statistical stability of Internet path characteristics to accurately predict the TCP throughput of large transfers using active probing. We propose additional mechanisms to explain the correlation, and then analyze why traditional TCP benchmarking fails to predict the throughput of large transfers well. We characterize stability and develop a dynamic sampling rate adjustment algorithm so that we probe a path based on its stability.

Parallel TCP flows are broadly used in high performance distributed computing, and can be used by the RGIS servers for update propagation rate adaptation. In this thesis, we address how to predict parallel TCP throughput as a function of the number of flows, as well as how to predict the corresponding impact on cross traffic. To the best of our knowledge, we are the first to answer the following question on behalf of a user: what number of parallel flows will give the highest throughput with less than a $p\,\%$ impact on cross traffic? We term this the maximum nondisruptive throughput. We begin by studying the behavior of parallel TCP in simulation to help derive a model for predicting parallel TCP throughput and its impact on cross traffic. Combining this model with some previous findings we derive a simple, yet effective, online advisor.http://www.cs.northwestern.edu/~urgis/GridG

Our analysis, design, and evaluation is based on a large-scale measurement study.

The multicast component: I contributed to the design of FatNemo, a novel overlay multicast protocol that emulates the fat-tree architecture on the wide area network. The overlay fat-tree architecture helps to prevent the root from becoming bottleneck in the case of multi-source multicast. The end-system multicast protocol

can be integrated into the CDN for better scalability and efficiency.

1.2 Related work

The data model, query language, and implementation of GISes and similar services that store the information about networked resources has been evolving for some time.

Today, many sites that provide externally accessible computing resources make a description of those machines available as web pages. Web search engines are often used to find appropriate resources. This has been aided significantly by the advent of highly discriminating search algorithms for arbitrary documents, such as PageRank [45]. By providing highly structured data, most GIS systems aim to provide more sophisticated queries.

Within the networking community, SLP [223] has been proposed as a standard for discovering services. The DNS name service [20] is universally used. DNS maps a hierarchical name (a path) to a blob of information and is typically used to resolve host names to IP addresses. Protocols for constructing and querying hierarchical distributed databases such as X.500 [121] and LDAP [119] can be viewed as extensions to this idea, although hierarchical databases predate DNS. Each node on an LDAP tree can have multiple typed attributes associated with it. An LDAP query is a traversal of a subtree that returns nodes whose attributes satisfy the query constraints. Each subtree can be serviced by a different LDAP server, making it straightforward to partition responsibility and security over multiple sites. In contrast to these approaches, RGIS builds on a relational data model instead of a hierarchical data model.

Within the distributed systems community, service location and naming services are basic needs. DCE [214], CORBA [168], and Java's Jini Framework [224] in-

clude these services. In DCE and CORBA, the service and the query consists of a type specification for a procedure or object (the interface) and the result is matching instances. Jini uses a more general tuple of attribute-value pairs as the service descriptor, and a tuple of attribute constraints as the query. One strand of recent research [16, 220, 135, 215] has focused on timelines of updates and on how services can push updates to users. Another strand has focused on distributing data throughout the network and then routing queries to likely nodes where matching data may reside using distributed hash tables [207, 192]. In contrast to these systems, RGIS attempts to provide compositional queries (joins) where collections of objects are needed to satisfy the query.

The Grid computing community has seen an explosion of work on GIS systems. The most relevant systems are Globus MDS2 [67], the Condor Matchmaker [185], and R-GMA [91]. MDS2 is based on LDAP and defines a schema (the attribute types) that can be associated with nodes in the tree. In contrast, RGIS uses a relational data model.

In the Condor Matchmaker, *both* resources and queries are collections of attributes and constraints. This enables bilateral matchmaking, where both the resource owner and the querier can constrain which results are returned on a query. Bilateral matching is a very fast process. Condor Matchmaker was later extended to support gang-matching, meaning that a query can be written that requires more than one resource to be satisfied [186]. Gang-matching is implemented using prioritized search with backtracking, which is more expensive than the search for bilateral matchmaking. Recently, Liu and Foster have proposed a matchmaking scheme and developed a system, Redline, in which the language for constraints enables the definition of constraint satisfaction problems (CSPs) [139]. CSPs are NP-Hard and are solved using the heuristic techniques implemented in an underlying CSP solver.

R-GMA [91] is close to our work in that it also proposes a relational data model for GIS systems. RGIS differs from R-GMA in two ways that are relevant to RGIS. First, R-GMA focuses currently on dynamic properties of resources (e.g., load), while RGIS focuses currently on relatively static properties (e.g., memory). Both systems are evolving to unify static and dynamic information, however. Our second difference is RGIS's support for nondeterministic, approximate and scoped queries.

Efforts to define the broad structure of the computational grids [94] and standardize the specifics of interaction among components [93] suggest that there are roles for multiple kinds of GIS systems, and that different systems can and will interoperate.

Traditional replicated transactional databases use strong consistency as a correctness criterion [39], while optimistic systems such as Coda [130] and Bayou [178] greatly favored the availability and performance over strong consistency. Recently, Yu, et al showed that relaxed consistency with a bound of maximum inconsistent access rate is important to many distributed applications. They also proposed numerical error, order error and staleness as three consistency bound metrics and applied anti-entropy [71] in building TACT [235], a middleware layer that enforces adjustable consistency bounds among the replicas. However, the replicas have to be synchronized in real time (in contrast to logical time) to bound the staleness of a replica. Besides, using anti-entropy is not network efficient because anti-entropy is like controlled unicast, while multicast at different levels will improve the efficiency.

To meet the consistency constraint, we need to estimate accurately the TCP throughput on the wide area network so that a RGIS server can finish sending its update within a time bound. The concept of available bandwidth has been of central importance throughout the history of packet networks, and researchers have been trying to create end-to-end measurement algorithms for a long time. From Keshav's packet pair [129], to Crovella's cprobe [53], and the latest work, such as IGI [117],

the purpose is to measure the end-to-end available bandwidth accurately, quickly, and non-intrusively. Today's definition of available bandwidth is "the maximum rate that the path can provide to a flow, without reducing the rate of the rest of the traffic." [117, 122]. Other tools to measure either the bottleneck link capacity or the available bandwidth include nettimer [134], pathchar and pchar [85], pathload [122, 123], NCS and pipechar [127], pathrate [84], spruce [208] and pathchirp [189], and Remos [141]. Most of such tools used the packet pair or packet train techniques to conduct the measurements.

The available bandwidth is different from the TCP throughput that an application can achieve, and that difference can be huge. Lai's Nettimer paper [134] showed many cases where the TCP throughput is much lower than the available bandwidth, while Jain's pathload paper [122] showed the bulk transfer capacity [158] of a path could even be higher than the measured available bandwidth. Additionally, most of these tools take a long time to run, which make them unsuitable to be used in real time for applications and services.

The most widely used TCP throughput prediction tool is Network Weather Service [230] (NWS). NWS applies benchmarking techniques and time series models to measure TCP throughput and provide predictions to applications in real time. NWS has been broadly applied. Allen, et al [24] applied NWS to address the so called Livny and Plank-Beck problems. Swany, et al [210] applied NWS in the grid information service.

Unfortunately, recent work [222, 221] has argued that NWS, and by implication, TCP benchmarking techniques in general, are not good at predicting large file transfers on the high speed Internet. Sudharshan, et al [222] proposed and implemented predicting large file transfers from a log of previous transfers and showed that it can produce reasonable results. However, a pure log-based predictor is updated

at application-chosen times and thus neglects the dynamic nature of the Internet. Hence, when a path changes dramatically, the predictor will be unaware of it until after the application begins to use the path. To take the dynamic changes of Internet into consideration, Sudharshan, et al [221] and Swany, et al [209] separately proposed regression and CDF techniques to combine the log-based predictor with small NWS probes, using the probes to estimate the current load on the path and adjust the log-based predictor. These techniques enhanced the accuracy of log based predictors. However, these combined techniques are limited to those host pairs that have logs of past transfers between them, and due to the dynamic nature of Internet, which only shows certain statistical stabilities, the logs can become invalid after some time. Furthermore, due to the strong correlation between TCP flow size and throughput [239], logs for certain ranges of TCP flow (file) size are not useful for the prediction of different TCP flow sizes. This motivated us to design DualPats, a novel TCP throughput monitoring and prediction framework that can predict TCP throughput in a soft real time manner without such constraints.

Parallel TCP flows are broadly used to enhance end-to-end throughput. For example, GridFTP [21], part of the Globus project [95], supports parallel data transfer and has been widely used in computational grids [22]. RGIS servers can utilize it for data transfer rate adaptation. Hacker et al [108] explained mechanisms parallel TCP flows achieve higher throughput, and proposed a loose upper-bound for parallel TCP throughput. However, to the best of our knowledge, we are the first to answer the following question on behalf of a user: what number of parallel flows will give the highest throughput with less than a p% impact on cross traffic? We term this the maximum nondisruptive throughput.We begin by studying the behavior of parallel TCP in simulation to help derive a model for predicting parallel TCP throughput and its impact on cross traffic. Combining this model with some previous findings

we derive a simple, yet effective, online advisor. We evaluate our advisor through extensive simulations and wide-area experimentation.

Multicast is an efficient way to distribute data among multiple users. End system multicast [64, 124, 30, 57, 187, 170] has recently become an effective alternative for IP multicast which only has very limited support on today's Internet [70, 79]. We designed a fat-tree based end-system multicast protocol that emulates a fat tree on the wide area network. This protocol can be used as an effective alternative for data propagation among the RGIS servers.

1.3 Outline and contributions

The following chapters describes several important components that can be used to build a scalable distributed RGIS system. These chapters also appeared in refereed publications.

1.3.1 Chapter 2: Architecture of the RGIS system

Chapter 2 describes the architecture of the RGIS system. This chapter also appeared as part of our refereed publication [77] in ACM/IEEE Supercomputing 2003.

Contributions: A scalable relational GIS was proposed and its architecture was presented.

1.3.2 Chapter 3: Generating Synthetic Grids

Chapter 3 describes the design and implementation of GridG, our extensible synthetic grids generator. GridG was used to generate synthetic grids information for the evaluation of the RGIS system. GridG consists of two components, namely the topology generator and the annotation generator.

This work was published in proceedings of ACM/IEEE Supercomputing 2003 [145] and Sigmetrics Performance Evaluation Review [143].

Contributions: GridG is the first topology generator that follows the power-laws of Internet topology while retaining a clear hierarchical network structure. Also, to my best knowledge, GridG is the first topology generator that can annotate the topology with sensible attributes from router level to the end-system level. While developing GridG, we also made contributions to the networking theory by revealing interesting relationships among the power-laws of Internet topology. We released the GridG toolkit at http://www.cs.northwestern.edu/~urgis/GridG.

1.3.3 Chapter 4: Query Rewriting Techniques

Chapter 4 describes the design and evaluation of our query rewriting techniques, including nondeterministic queries, scoped and approximate queries.

This work was published in ACM/IEEE Supercomputing 2003 [77] and Grid 2003 [147].

Contributions: These query rewriting techniques trade off the query time with the number of results returned. We show that tradeoffs over many orders of magnitude are possible, and the techniques can also be used to keep query processing time largely independent of query complexity, albeit returning fewer results with more complex queries. Several time-bounded query techniques were also developed based on the three query rewriting techniques to provide soft real time bound for each query.

1.3.4 Chapter 5: Scheduling With Inaccurate Job Size Information

Chapter 5 describes our simulation study on size-based scheduling policies with inaccurate job size information. A few new applications were also briefly discussed. This work can be potentially used by the RGIS servers.

This work was published in IEEE MASCOTS 2004 [154]. Another two papers relating to this topic are in submission.

Contributions: Through simulations, we have evaluated the performance of size-based scheduling policies (SRPT and FSP, with PS for comparison), as a function of the correlation between actual job size and estimated job size. We found that SRPT and FSP's performance strongly depends on the correlation. When provided with weak correlation, SRPT and FSP can actually perform worse than PS, but given a reasonably good job size estimator, they can outperform PS in both mean response time and the slowdown. We also described three new applications of SRPT and FSP.

To generate correlated trace data for the simulation, we introduced a new random number pair generation technique, where each number of the pair is chosen from its required distribution and they are correlated to degree R. This technique can be very useful for simulation related research in this and other areas.

To the best of our knowledge, this work is the first to address the performance of size-based policies with inaccurate scheduling information.

1.3.5 Chapter 6: Characterizing and Predicting TCP Throughput

Chapter 6 describes DualPats, our TCP throughput monitoring and prediction framework that can be used to build the TCP throughput monitoring and prediction component of the RGIS system.

This work was published in ICDCS 2005 [152].

Contributions: We have characterized the behavior of TCP throughput in the wide area environment, providing additional explanations for the correlation of throughput and flow size and demonstrating how this correlation causes erroneous predictions to be made when using simple TCP benchmarking to characterize a path. In response, we proposed and evaluated a new benchmarking approach, dualPacket, from which TCP throughput for different flow sizes can be derived. We developed a novel dynamic sampling rate adjustment algorithm to lower the probing overhead while capturing the dynamism of end-to-end paths. Based on these results, we described and evaluated the performance of a new TCP throughput monitoring and prediction framework, DualPats, and implemented this approach.

1.3.6 Chapter 7: Modeling and Taming Parallel TCP

Chapter 7 describes our techniques in modeling and taming parallel TCP flows which can be used as an adaption mechanism to control the data transfer time between RGIS servers.

This work was published in IPDPS 2005 [150].

Contributions: This chapter shows how to predict both parallel TCP throughput and its impact on cross traffic as a function of the degree of parallelism using only two probes at different parallelism levels. Both predictions are monotonically changing with parallelism levels. Hence, the `TameParallelTCP()` function can be implemented using a simple binary search. To the best of our knowledge, our work is the first to provide a practical parallel TCP throughput prediction tool and to estimate the impact on the cross traffic.

1.3.7 Chapter 8: FatTree Based End-System Multicast

Chapter 8 describes the design and implementation of FatNemo,a fat-tree based end system multicast protocol. This is a highly scalable and efficient data distribution mechanism that can be used by the CDN of the RGIS system.

Initial results were published in WCW 2004 [41] and the final long version is in submission.

Contributions: This chapter makes three main contributions. First, this chapter introduces the use of Leiserson fat-trees for application-layer multi-source multicast, overcoming the inherent bandwidth limitations of normal tree-based overlay structures (Section 8.3). Second, this chapter describes the design and implementation of *FatNemo*, a new application-layer multicast protocol that builds on this idea. Lastly, this chapter evaluate the FatNemo design in simulation, illustrating the benefits of a fat tree approach compared to currently popular approaches to end-system multicast.

1.3.8 Chapter 9: Conclusions and Future Work

Chapter 9 concludes this thesis and describes future work.

1.3.9 Appendix A: Domain-based scheduling on web servers

We proposed and evaluated domain-based scheduling, a simple technique that better estimates connection times by making use of the source IP address of the request. Domain-based scheduling improves SRPT and FSP performance on web servers, bringing the performance benefits of these scheduling polices even to those regimes where the correlation between file size and service time is low. Our paper is in submission.

Contributions: We have demonstrated that the assumption that file size is a good indicator of service time for web servers is unwarranted. File size and service time are only weakly correlated. We have evaluated the performance of SRPT-FS and FSP-FS, SRPT and FSP by running simulations driven by our web server traces. We found that their performance does indeed degrade dramatically with the weak correlation reflected in the trace. In some cases SRPT-FS and FSP-FS can actually perform worse than PS.

We have proposed, implemented, and evaluated a better service time estimator that makes use of the hierarchical nature of routing on the Internet and the history of past requests available on the web server. We refer to SRPT and FSP augmented with our domain-based estimator as SRPT-D and FSP-D. The state size of our estimator is a parameter.

1.3.10 Appendix B: Scheduling the server side of P2P systems

We focused on the problem of scheduling download requests at the server-side of P2P systems with the intent of minimizing the average response time experienced by users.

Early results were published in LCR 2004 [182]. Our final long version is in submission.

Contributions: We characterized P2P server side workload based on extensive trace collection and analysis. We also evaluated the performance and fairness of different scheduling policies through trace-driven simulations. Our results show that average response time can be dramatically reduced by more effectively scheduling the requests on the server-side of P2P systems.

Chapter 2

Architecture of the RGIS system

This chapter describes the architecture of the RGIS system. The following chapters describe the design of the individual components of the system.

Figure 2.1 illustrates the structure of the RGIS system, focusing on a single RGIS server. We expect that each site within a computational grid will run one such server, although multiple servers per site can also be supported. The goal of the RGIS server for a site is to provide a view of the entire computational grid appropriate to that site's users. A site's RGIS server is responsible for all queries issued by users on the site.

An RGIS server is built around an RDBMS system. At the present time, we use Oracle 9i Enterprise Edition, but our system could also be based on other RDBMS systems such as DB2, MS SQL Server, Postgres, and MySQL. A early implementation of our work used MySQL. Like most serious database systems, Oracle provides a single front-end interface to multiple back-end implementations. In particular, this provides platform independence (Oracle runs on many operating systems and platforms) and intra-site scalability (Oracle has a variety of implementations, including a Parallel Server product that scales over clusters). RGIS consists of ~18,200 lines of Perl and ~2500 lines of SQL and PL/SQL. ~5000 lines of PL/SQL are automatically

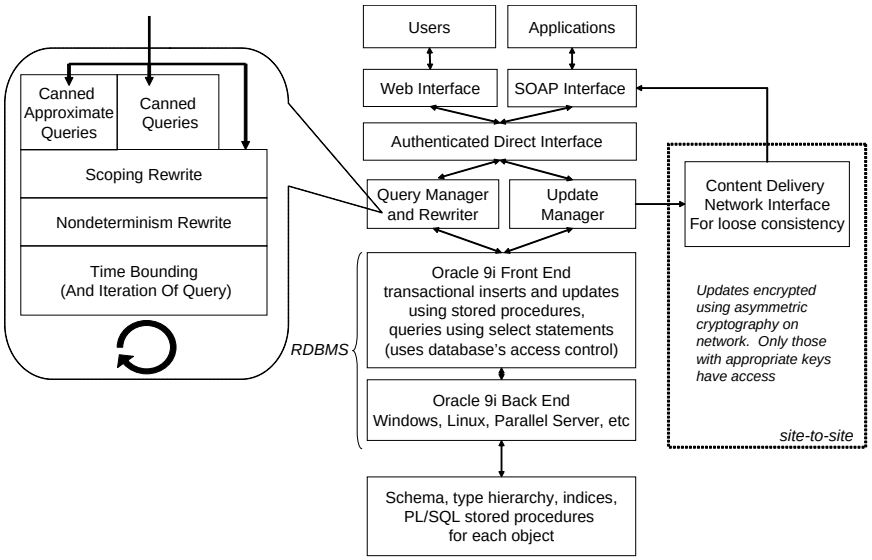


Figure 2.1: RGIS Structure.

generated.

RGIS includes a type system to identify a wide range of components including hosts, routers, switches and hubs at layers 2 and 1, links at layers 3 through 1, paths at layer 3, benchmarks, operating systems, operating system vendors and versions, switches, switch vendors, software modules, running software, and communication endpoints.

Is it feasible to populate a database of such components, particularly the network graphs? Yes. Beyond the observation that network administrators and designers can provide such data for their individual sites, automated network mapping can also be used. Mapping networks at layer 3 is a well established research area that started in the late '90s [60] and has evolved to a considerable degree of sophistication today [118, 206, 83]. The last paper provides an extensive discussion of research and tools in this area. Layer 2 network discovery within a site, in particular automatically mapping large switched Ethernet networks, has also been achieved [142].

Typed objects are inserted into the database by updating one or more tables. An object is also identified in a special table (insertids) by a unique insertion identifier, a timestamp (NTP is assumed), and ancillary information to support our specialized needs, such as our query extensions, and to link virtual resources with physical resources [90]. Every other table that is updated includes this insertion id, hence making it easy to find all elements of an object, no matter what tables it spans.

The RGIS schema includes the sequences, tables, constraints, triggers, and indices that represent our grid modeling efforts. Figure 2.2 shows a high-level view of the RGIS schema, focusing on the representation of a host computer.[1]

Given transactional updates, the constraints and triggers are designed to keep the database in a consistent state. We use Oracle's access control mechanisms to assure that insertions, updates, and deletions occur only via stored procedures that force transactions. For every type of object, there is an associated PL/SQL package (essentially, a class) that provides this functionality. Each package also includes non-transactional versions of the operations which very privileged users can use to batch multiple updates together into a single transaction. These packages are automatically generated from the schema so that the schema can evolve easily. Figure 2.3 shows our SQL representation of a host.

Layered on top of the RDBMS front-end is a query manager/rewriter, and an update manager. Chapter 4 describes the details of our query rewriting techniques while chapter 5 explores the feasibility of applying size-based scheduling policies on the query manager. Together, these two provide the core application interface to an RGIS server. This interface is exported through a layer that provides authentication of the user and checks his capabilities for each request, mapping from an external

[1]The formal schema is available at http://urgis.cs.northwestern.edu/schemas.

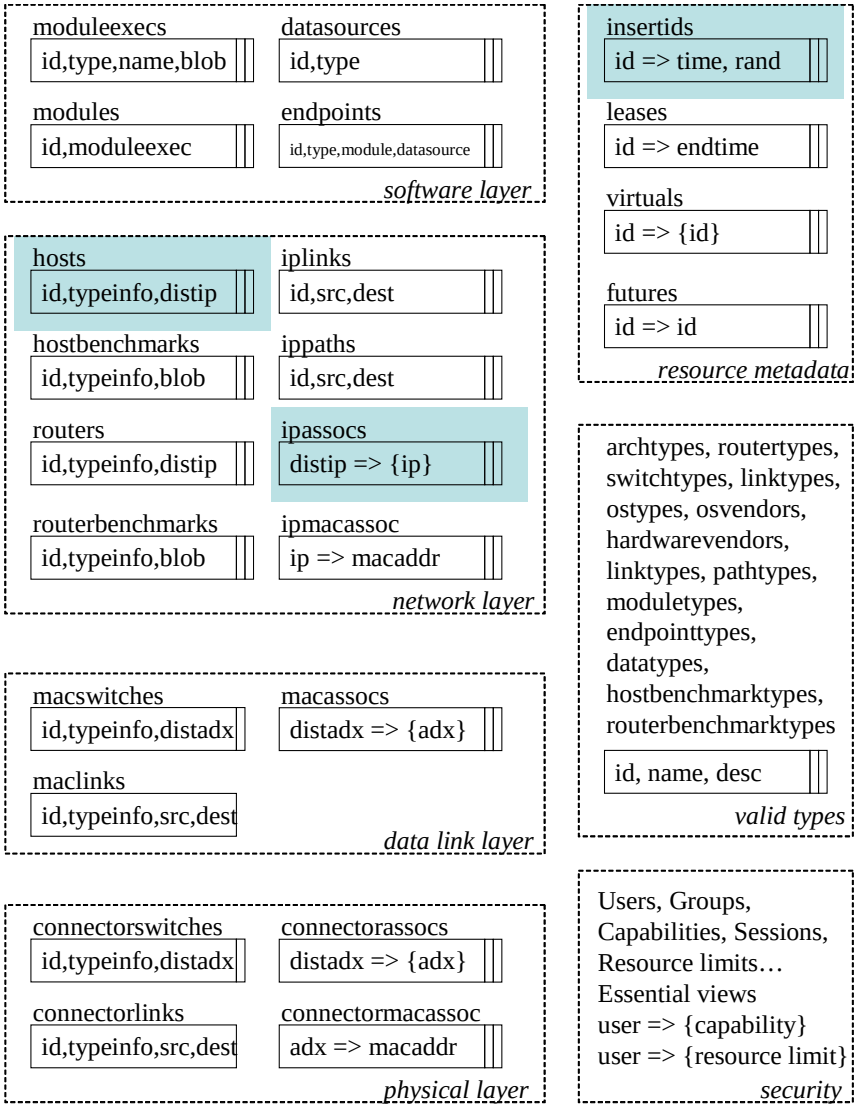


Figure 2.2: Overview of the RGIS Schema. Highlighted are the minimum tables used to represent a host. A host may also be represented in the leases table if it may leave the system, the virtuals table if it is a virtual machine, and the futures table, if it is not yet instantiated.

```
CREATE table hosts (
  distip      varchar2(15)     not null primary key,
  name        varchar2(256)    default('UNKNOWN'),
  numproc     number           default(1),
  mhz         number           default(0),
  constraint good_mhz_hosts check (mhz>=0),
  arch        varchar2(32)     default('UNKNOWN'),
  constraint good_arch_hosts foreign key (arch)
     references archtypes(name),
  hwvendor    VARCHAR2(32)     DEFAULT('UNKNOWN'),
  constraint good_hw_hosts foreign key (hwvendor)
     references hardwarevendors(name),
  os          varchar2(32)     default('UNKNOWN'),
  constraint good_os_hosts foreign key (os)
     references ostypes(name),
  osvendor    varchar2(32)     default('UNKNOWN'),
  constraint good_osv_hosts foreign key (osvendor)
     references osvendors(name),
  osver       varchar2(256)    default('UNKNOWN'),
  kernelver varchar2(256)      default('UNKNOWN'),
  mem_mb      number           default(0),
  constraint good_mem_hosts check (mem_mb>=0),
  vmem_mb     number           default(0),
  constraint good_vmem_hosts check (vmem_mb>=0),
  disk_gb     number           default(0),
  constraint good_disk_hosts check (disk_gb>=0),
  location    varchar2(256)    default('UNKNOWN'),
  owner       varchar2(256)    default('UNKNOWN'),
  description varchar2(256),
  insertid    number           not null unique,
  constraint good_insert_hosts foreign key
     (insertid) references insertids(insertid)
     ON DELETE cascade
);
CREATE table insertids (
  note      varchar2(256),
  time      timestamp  not null,
  insertid  number             not null primary key,
  rand      number             not null
);
CREATE table ipassocs (
  distip      varchar2(15)     not null,
  ip          varchar2(15)     not null primary key,
  insertid    number           not null,
  constraint good_insert_ipassocs foreign key
      (insertid) references insertids(insertid)
       ON DELETE cascade
);
```

Figure 2.3: Specific SQL representation of a host. Definitions of indices elided.

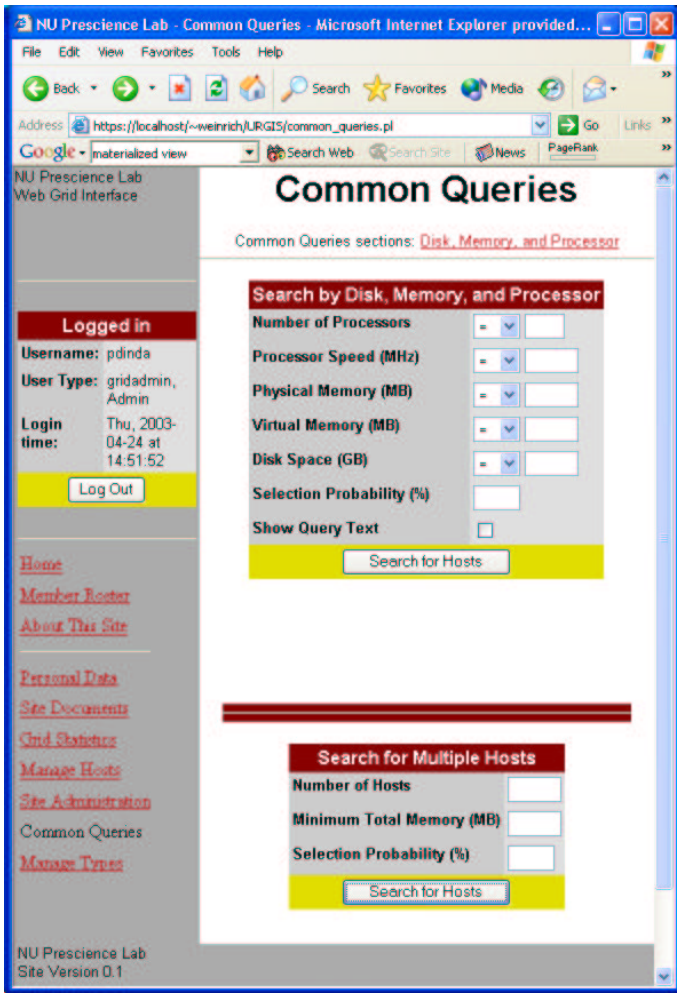


Figure 2.4: RGIS web interface.

notion of user to a database-local notion of user. This interface in turn is made visible to the outside world via a web interface, and a SOAP interface, both running over the encrypted HTTPS protocol. Figure 2.4 shows the current web interface of RGIS.

The update manager aggregates updates (insertions of new objects, deletions of existing objects, and changes to the properties of existing objects) coming from local sources and remote sources and batches them into transactions for the RDBMS. In this role, it can prioritize updates and also control the rate of updates and the update latencies going into the database.

RGIS servers do not talk directly with each other, but indirectly via a content delivery network (CDN), which is based on the publish/subscribe model and is used solely to propagate updates to friendly RGIS servers. There is no implicit notion of trust among RGIS servers. If a site is interested in receiving updates from a remote RGIS server, it must arrange with the remote administrator to create a key pair. Each update to the local RGIS server is then encrypted in a manner similar to PGP multiple destination messages [101], making it readable only to those RGIS servers that hold one of the perhaps many keys used in the encryption process. The CDN is used to send the update to the group of all those holding keys using either unicast or more scalable application-layer multisource multicast. Chapters 6, 7 and 8 are devoted to these data transfer components. An RGIS server combines local updates and remote updates to create a view of the computational grid that corresponds to that which its users have access.

In the limit, each RGIS server could contain data about all the resources on a wide area network, although we expect this will rarely happen. Although this is clearly asymptotically unscalable, it is not unreasonable for computational grids of likely size. Consider a computational grid of one billion hosts and routers (about five times the current size of the Internet). With two kilobytes of information per host, about 2 TB

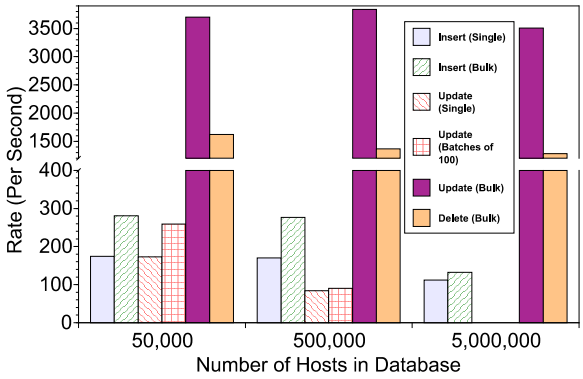


Figure 2.5: Insert, update, and delete rates.

of data storage would be necessary in the RGIS server. In 2004, this requires less than $10,000 of disk storage using a modern direct-attach RAID box, making it clearly within the realm of possibility. Furthermore, the $/MB of disk capacity is shrinking much faster than the Internet is growing. Update rates can be an issue, but three things ameliorate this. First, we can achieve quite high update rates on off-the-shelf RDBMS systems such as Oracle. Figure 2.5 shows the rates for insertions, updates, and deletions in RGIS with different size databases running on our hardware (See Section 4.3 for details about the hardware and software configuration). Here an insert means adding a host to the database, which involves a transactional modification of a sequence and three tables, an update means modifying the memory attribute of a host already in the database, which is a transactional modification of two tables, and delete means to remove a host from the database, transactionally modifying three tables. Second, bandwidth into a site grows with the update rate, since the update rate grows with the number of hosts and routers. Third, RDBMS systems such as Oracle and DB2 can scale over clustered servers to support very high update rates. In effect, we can leverage the existing TPC-C online transaction processing benchmark competition [219] to address the updates.

A site sends queries only to its RGIS server. This ties the resources a site is willing to commit to its RGIS server to the number and kind of queries it wants to make. This is vital because the nature of many RGIS queries is similar to decision support queries (TPC-H [219]) in relational database systems. Such queries can be very expensive to execute and so are unlikely to be welcome on foreign RGIS servers.

The goal of the query manager/rewriter is to shape the query workload so that it can be effectively executed by the RDBMS, by which we mean that the load on the RDBMS is kept below one and individual queries finish quickly. Chapter 4 describes our query rewriting techniques in details. Queries take the form of select statements written in a slightly extended form of SQL. The query manager/rewriter translates queries into the underlying SQL dialect, *modifying the query semantics to balance between the needs of the query and the needs of the system.* In essence, the query manager/rewriter can trade off between the result set size for a query and the resources the query requires to execute.

Query processing is a three stage process, as shown in Figure 2.1. First, for common inquiries, we provide pre-written "canned queries". Some canned queries have been rewritten to an approximate form by hand. Both canned queries and queries constructed on the fly pass through two further automatic rewriting steps, scoping and nondeterminism. The final SQL query is then executed for a specified amount of (wallclock) time and the matching results are returned. The use of nondeterminism and scoping can cause the query to be run repeatedly within the alloted time if necessary, using a different random sample each time.

In industry, the development of relational queries and their optimization is often allotted to a specialist. Similarly, we envision a "grid query developer" who will create canned queries. However, RGIS users will always be able to write their own queries of arbitrary complexity (but constrained running time.)

Chapter 3

Generating Synthetic Grids

This chapter describes the motivation, design and implementation of GridG, our extensible toolkit for generating and annotating synthetic grid topologies. GridG was used to evaluate the RGIS system.

GridG was released at http://www.cs.northwestern.edu/~urgis/GridG.

3.1 Introduction

Designing and evaluating grid middleware demands realistic workloads. The RGIS system is no exception [74, 77]. Using RGIS system, users are able to pose complex compositional queries that resemble decision support queries. A typical query might look for a group of machines that use the same OS, together have a certain amount of memory, and that the subset of the network connecting them have some bisection bandwidth. To make these queries fast, we implement them using stochastic search, allowing us to trade off between the number of nondeterministically chosen results returned by the query and the amount of work done in support of it. This tradeoff depends strongly on the structure of the grid: the network topology and the characteristics of the hosts, routers, and links within the topology.

While Smith, et al [204] studied the update and query processes on such grid

information, there is no extant work and limited available data on the structure of computational grids. We examined the contents of several running GIS systems. The largest dataset we have found, generously provided by Smith, contains fewer than one thousand nodes. Given the limited data sets, a synthetic grid generator is a necessity. Furthermore, even as more data becomes available, it will continue to be useful to have a parametric source of grids. For example, such grids could be used with simulation toolkits such as GridSim [49], Simgrid [55], MicroGrid [205] and Bricks [17]. to study the benefits of different scheduling techniques. Unfortunately, no such generator currently exists. Synthetic grids could also be useful in simulation studies of overlay networks and peer-to-peer systems [193, 31]. Although the correctness of such protocols do not depend on the underlying topology, their efficiency does depend on topology and on the performance of the end-systems.

In response to this need, we have built GridG, a grid generator. Our definition of a synthetic grid is an annotated directed graph in which the nodes represent hosts, routers, switches, and hubs, and the edges represent network links. The graph is thus a network topology that extends to the level of hosts. In addition, each node or edge is annotated with information relevant to its use as a part of a computational grid. A grid generator, such as GridG, produces a grid of a given number of hosts. It must meet the following requirements:

- It must produce a realistic network topology. Much is known about the properties of real network topologies: they are connected, and they have hierarchical structures. Furthermore, wide-area network topologies, including the Internet, have recently been found to follow certain topological power laws [88]. A good generator will provide both structure and follow the power laws [213].

- It must generate realistic annotations for hosts and network components. For

a host, it should at least provide the architecture type, processor type and speed, number of processors, memory size, disk size, hardware vendor, operating system, and available software. For a link, it should provide the hardware bandwidth and latency. For routers and switches, it should specify the aggregate backplane or switching fabric throughput and latency. It should capture correlations between different attributes (for example, we might expect that memory size increases with processor speed with high probability), and between nearby components (for example, a high speed router is unlikely to be connected only to a few slow links).

The networking community has produced a wide range of topology generators, including random Waxman [226], Tiers [80, 50], Inet [125, 229], etc. These generators either meet the structure requirement or they meet the power-law requirement. GridG starts with the output of a structure-oriented topology generator (we currently use Tiers) and adds redundancy to it in such a way as to make it conform to the power laws. As far as we are aware, this makes it the first topology generator that provides structured topologies that obey the power laws.

GridG in fact directly enforces only one power law, the so-called outdegree exponent power law. Its outputs, however, show obedience to all the other laws as well. In studying the unreasonable effectiveness of the outdegree law, we discovered a new fact: it is either the case that the so-called rank exponent power law is not actually a power law, or that it is more significant than the others. We believe that the former is actually the case, but that over the typical range that is considered, a power law is a useful approximation. At this point, we believe the outdegree law is more significant and that the other power laws can be derived from it.

GridG provides mechanisms for annotating each node and edge. These mecha-

nisms are currently based on user-supplied empirical distributions and conditional probability rules. The rules enforce correlations between different attributes on the same graph element and correlations between different graph elements that are close to each other. Distributions and rules can be determined by measurement. For example, we discovered OS concentrations on all the class C IP subnets we probed with nmap—most subnets appear to have a dominant operating system. These kinds of clustered attributes are very important for Grid modeling, resource allocation and scheduling research, because much of the resource allocation and scheduling work that has proved successful depends on clustered homogeneity of such attributes. We added this optional rule to GridG as an example of capturing correlation between attributes on nearby graph elements. We also added optional rules correlating the different attributes of a host that are not measurement based, but reflect the sensible beliefs most people have of how machines are configured.

Unfortunately, very little is known about the characteristics of a grid or network that are represented by the annotations. We point to the information that we think is necessary to develop models of these little studied network and host characteristics. The successful collection of this kind of data, and the models that could be developed from it are a very exciting research opportunity.

In the following, we begin by presenting the overall architecture of GridG in Section 3.2. While GridG can apply any number of transformations to produce a synthetic grid, there are two core steps: topology generation and annotation. In Section 3.3, we describe how topology generation is done and demonstrate that GridG conforms to the power laws of Internet topology. Section 3.4 discusses our insights into those laws, including the apparent contradiction between two of them. Section 3.5 describes the GridG mechanisms for annotation, outlines the requirements of a model for annotation, discusses the OS concentration phenomenon, and de-

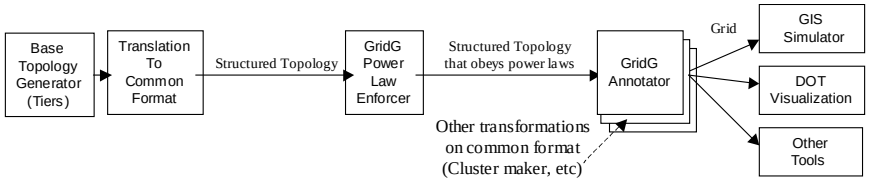


Figure 3.1: GridG Architecture.

scribes the open research questions posed by the need for such a model and how we are attempting to address them. Finally, Section 3.6 concludes this chapter.

3.2 Architecture of GridG

GridG is implemented as a sequence of transformations on a text-based representation of an annotated graph. The transformations are generally written in Perl, although this is not a requirement. Figure 3.1 illustrates how these transformations are composed to generate a grid. Currently, we begin with a structured graph without redundancy that is generated by the Tiers topology generator. The number of networks at each level of the topology is the primary input. This also indirectly specifies the number of hosts. The first transformation enforces the power laws by adding extra links to the graph. The outdegree exponent is main input. The next transformation annotates the graph according to user-defined empirical distributions on and correlations over attributes such as memory and CPU. Additional transformations can be added. For example, we can add clusters to sites on the grid. The final output can then be visualized with DOT, used for GIS evaluation, or for other purposes.

Power Laws	Rank exponent	$d_v \propto r_v^R$
	Outdegree exponent	$f_d \propto d^O$
	Eigen exponent	$\lambda_i \propto i^\varepsilon$
Approximation	Hop-plot exponent	$P(h) \propto h^H$

Figure 3.2: Power laws of Internet topology.

Symbol	Description	Typical Values or Constraints
R	Rank exponent	≈ -0.488 at router level
O	Outdegree exponent	≈ -2.487 at router level
ε	Eigen exponent	≈ -0.177 at router level
H	Hop-plot exponent	≈ 2.84 at router level
d_v	Outdegree of node v	$1 \leq d_v \leq MaxD$
$MaxD$	Theoretical maximum outdegree	$MaxD = e^{-\frac{\log \alpha}{O}}$
r_v	Ranking of node v	Nodes with same outdegree have same ranking
f_d	Frequency of outdegree d	Number of nodes with outdegree d, $f_d \geq 1$
h	Number of hops	
λ_i	The i^{th} biggest eigenvalue of the graph	
$P(h)$	Total number of pairs of nodes within h hops	
N	Total number of nodes in the graph	
β	Constant in equation 3.1	$\approx exp(4.395)$ at router level
α	Constant in equation 3.3	$\approx exp(8.52)$ at router level

Figure 3.3: Symbols used in this chapter.

3.3 Topology

GridG generates topologies comprised of hosts, routers, and IP layer links. In GridG's graphs, nodes in WANs and MANs are routers while nodes in LANs are hosts. Routers have switching capability and several IP interfaces while hosts have computing and storage resources.

Recent research [35, 19, 131] shows that many natural and artificial networks follow the so-called outdegree power law, including such examples as molecules in a cell, the power grid, the World Wide Web, species in an ecosystem, and people in a social group. In particular, recent work [88, 191] show that not only does the Internet topology [88] follow this power law, but also the peer-to-peer sharing overlay network Gnutella [191]. Like the Gnutella network, future grids will be embedded in the Internet topology and thus will likely follow its rules.

3.3.1 Power laws of Internet topology

Faloutsos, et al [88] identified three power laws and one approximation in their influential 1999 paper. Figure 3.2 summarizes these laws. (Figure 3.3 summarizes the symbols used in this chapter.) The rank exponent law says that the outdegree, d_v, of a node v, is proportional to the rank of the node, r_v, raised to the power of a constant R. In our examples and evaluation, we choose to parameterize our power laws according to the router-level data in the Faloutsos paper. The parameterized rank exponent law is

$$d_v = \beta r_v^R = exp(4.395) * r_v^{-0.49} \tag{3.1}$$

The omitted constant term does not affect our results and is commonly dropped [18]. Another useful form of the rank exponent power law is

$$r_v = (\frac{d_v}{\beta})^{\frac{1}{R}} \tag{3.2}$$

The outdegree exponent law says that the frequency, f_d, of an outdegree d, is proportional to the outdegree raised to the power of a constant O. Parameterizing the law using the Faloutsos router-level data, we have

$$f_d = \alpha d^O = exp(8.52) * d^{-2.49} \tag{3.3}$$

A node's ranking is defined in the following way, conforming with the Faloutsos paper. We do a topological sort of the nodes in decreasing order of outdegree. We then assign ranks according to this ordering and the number of nodes in each equivalence class. All n_1 nodes in the class with largest outdegree are assigned rank $r_v = 1$. All n_2 nodes in the class with the second largest outdegree are assigned rank $r_v = 1 + n_1$. This accumulation continues such that all nodes in the class with the

kth largest outdegree are assigned rank $r_v = 1 + n_1 + n_2 + \ldots + n_{k-1}$. For example, if there are 1000 nodes with outdegree larger than 3, and there are 100 nodes with outdegree 3, then the nodes with outdegree 2 will be ranked 1101. All nodes with same outdegree have the same ranking.

The Eigen exponent power law says that the eigenvalues, λ_i, of a graph are proportional to the order, i, raised to the power of a constant ε. Here, the topology graph is represented as an adjacency matrix and its eigenvectors and eigenvalues are found. The eigenvalues represent the contribution of each eigenvector to the graph, in decreasing order.

The hop-plot exponent law is listed as an approximation by Faloutsos, et al. It says that the total number of pairs of nodes, $P(h)$, within h hops, is proportional to the number of hops raised to the power of a constant, H.

3.3.2 Current graph generators

There are mainly three types of topology generators in use: random [226], hierarchical, and degree-based. Debates as to which type is better for Internet graph generation have persisted over a long period of time [213]. Our belief is that a good graph generator should produce a clear hierarchy that also follows the discovered power laws. Hierarchical generators such as Tiers [50, 80] and Transit-Stub [50] can generate a clear hierarchical network, but the graphs don't follow the power laws by nature. The degree-based generators, such as Inet [125, 229], Brite [161], the CMU power law graph generator [172] and PLRG [18], generate graphs that follow the power laws,but have no clear hierarchical structure. The topologies generated by GridG follow the power laws *and* have a clear three-level hierarchy.

3.3.3 Algorithms in topology generation

GridG takes the output of a basic graph generated by Tiers as its input. This input graph has no redundant links. GridG adds links to the graph according to the outdegree power law. Hence, the graphs generated by GridG have a clear three-level hierarchical structure and follow the power laws. The following is a more detailed description.

1. Generate a basic graph without any redundant links using Tiers. Tiers itself has several parameters, specifically the number of nodes and networks at each level of hierarchy. Translate the graph into GridG's native format. This basic graph has three levels of hierarchy: WAN, MAN, and LAN. At each level, the nodes are connected by a minimum spanning tree. Each lower level network is connected to one node on the higher level network. The graph is guaranteed to be connected.

2. Assign each node an outdegree at random using the outdegree power law as the distribution. The probability $P(k)$ that a vertex in the network interacts with k other nodes decays as a power law. This probability is scale-free, meaning that we can extend graphs of any size in this manner [35]. Nodes of outdegree one deviate from the power law as described by Faloutsos, et al [88, Figure 6(b)]. We set $f_1 = f_2$ as this is the case for real router level data. Given outdegree $d = 2, 3 \ldots MaxD$, we calculate the corresponding frequencies according to $f_d = exp(8.9) d_v^{-2.486}$, where 8.9 and -2.486 are the defaults for parameters given a configuration file. $N = \sum_{i=1}^{MaxD} f_i$, where N is the total number of nodes in the graph. We then generate a random number x between 1 and N for each node, if $x \leq f_1$, the node is assigned outdegree 1; if $f_1 < x \leq f_1 + f_2$, the node is assigned outdegree 2; if $f_1 + f_2 < x \leq f_1 + f_2 + f_3$, the node is

	Internet Routers	GridG	Tiers
Rank exponent	-0.49	-0.51	-0.18
R^2		0.94	0.89
Outdegree exponent	-2.49	-2.63	-3.4
R^2		0.97	0.55
Eigen exponent	-0.18	-0.24	-0.23
R^2		0.97	0.97
Hop-plot exponent	2.84	2.88	1.64
R^2		0.99	0.99

Figure 3.4: GridG topology generator evaluation.

assigned outdegree 3, etc.

3. Calculate the remaining outdegree of each node after taking the links of the minimum spanning tree into consideration.

4. Add redundant links to the graphs by randomly choosing pairs of nodes with remaining outdegree > 0. Nodes at higher levels (e.g., WAN) are given priority over nodes at lower levels (e.g., MAN). Continue to add more redundant links until no pairs of nodes with positive outdegree can be found.

3.3.4 Evaluation

In this section, we show that the graphs generated by GridG follow the power laws. For comparison, the basic graph generated by Tiers is also shown in our figures. The basic graph was generated by "Tiers 1 50 10 500 40 5 1 1 1 1 1", meaning one WAN containing 50 MANs, each containing 10 LANs. The WAN contains 500 nodes, while the MANs and LANs contain 40 and 5 nodes, respectively. This is similar to the parameters used in other evaluations [213].

Figure 3.4 shows that the exponents of the topology generated by GridG match router level data from the Faloutsos paper much better than those of the basic Tiers

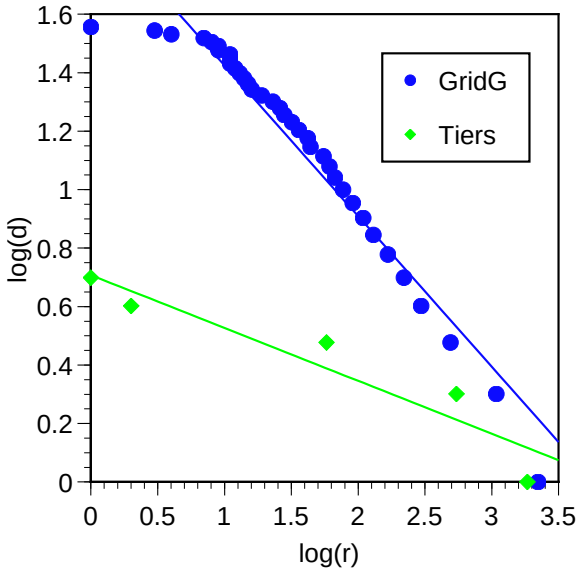


Figure 3.5: Log-log plot of Outdegree vs. Ranking.

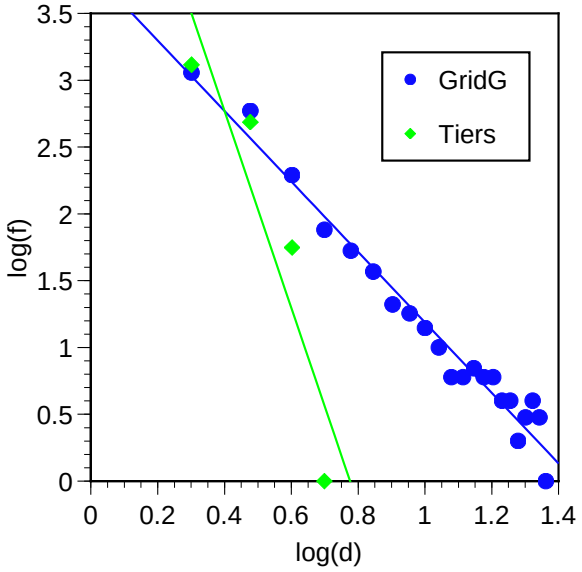


Figure 3.6: Log-log plot of Frequency vs. Outdegree.

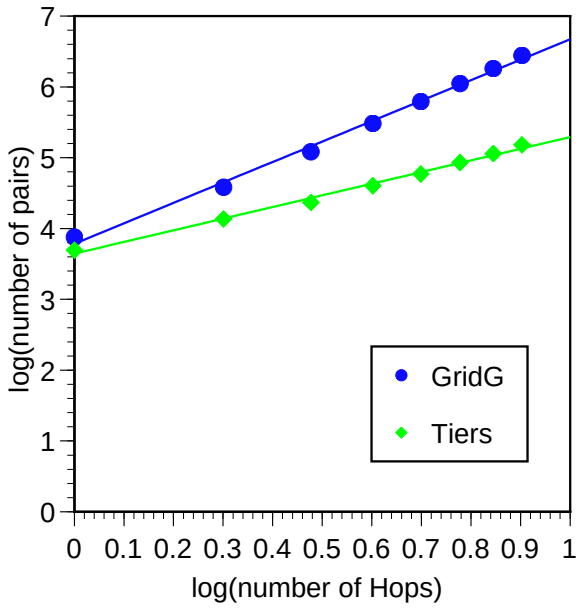


Figure 3.7: Log-log plot of number of pairs of nodes within h hops vs. number of hops h.

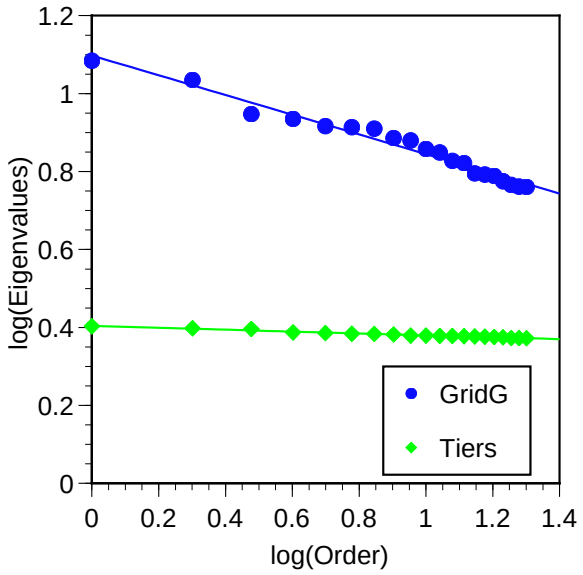


Figure 3.8: Log-log plot of eigenvalues in decreasing order.

graph. The coefficients of determination, R^2, represent how well a power law fits the generated data. We can see that GridG produces R^2 values close to 1, the ideal. The exponents in Figure 3.4 are the slopes in Figures 3.5, 3.6, 3.7, and 3.8.

Figure 3.5 is a log-log plot of outdegree versus ranking. We can see that a linear fit on this graph explains the relationship for GridG's topology very well. The divergence at small ranks is quite interesting and shows up in studies of real topologies including Faloutsos, et al [88, Figure 4(b)] and Medina, et al [162, Figure 6]. Removing the three diverging datapoints from Figure 3.5 increases R^2 to 0.99. In Section 3.4, we will describe a potential new rank law that models this divergence and can be derived from the outdegree law.

Figure 3.6 shows a log-log plot of frequency versus outdegree. GridG follows the outdegree exponent power law very well except when outdegree equals 1, which is not plotted in the graphs. We have already noted that this divergence is intentionally induced to better match real topologies.

Figure 3.7 is a log-log plot of the number of pairs nodes within h hops versus number of hops h. Clearly, GridG's topology conforms to this power law.

Figure 3.8 is a log-log plot of the eigenvalues in decreasing order. We can see that GridG agrees very well with this power law, though our exponents deviate slightly from the data given by Faloutsos, et al [88].

3.4 Relationships among power laws

Several recent graph generators [125, 229, 18, 172] and GridG generate graphs according to the outdegree law only. However, the generated graphs follow all four power laws! Why is this possible? A possible reason is that the power laws (Figure 3.2) are closely interrelated. A recent paper [35] proposed incremental growth

and preferential connectivity to explain the phenomenon and origin of the outdegree law. Medina, et al found that the hop and eigenvalue power laws were followed by all the topologies they considered [162]. Mihail and Papadimitriou have shown that the eigenvalue law follows from the outdegree law [163].

In the following, we show that the outdegree law follows from the rank law. It does not appear, although we can not prove, that the rank law follows from the outdegree law. This suggests one of several possibilities:

- The rank law is strictly more descriptive than the outdegree law.
- The rank law is wrong.
- The outdegree law is wrong.

The evidence against the first and third possibilities is the unreasonable effectiveness of using the outdegree law to generate graphs that appear to follow all of the laws. Furthermore, as we noted earlier, the rank/outdegree relationship diverges from a strict power law in actual topologies at small ranks. Finally, earlier work has shown that the eigenvalue law follows from the outdegree law and that most networks exhibit the eigenvalue and hop-plot laws.

Our belief is that the second possibility is the case. We show that it is possible to derive a power law like rank law from the outdegree law that captures the divergence seen in real topologies and gives the appearance of a power law over much of its range. This also would explain the surprising effectiveness of only using the outdegree law in graph generation. We advocate the following relationship among the laws:

$$\boxed{\text{New rank law} \iff \text{Outdegree law} \implies \text{Eigenvalue law.}}$$

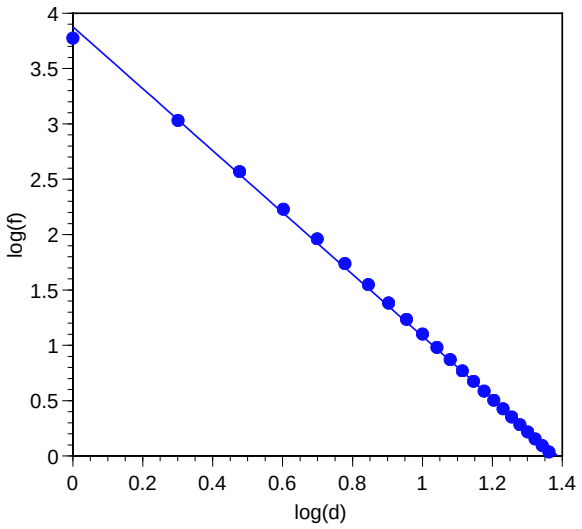


Figure 3.9: Log-log plot of derived f-d law.

3.4.1 Rank law $\Longrightarrow$ outdegree law

Starting from the rank law, we derive a form of the outdegree law. Let f_d be the frequency of nodes with outdegree equal to d, or the number of nodes with outdegree d. Let r_d be the ranking of the nodes with outdegree d. Similarly, let r_{d-1} be the ranking of the nodes with outdegree equal to $d-1$. Given the outdegrees d and $d-1$, and the ranking of nodes with those outdegrees, the frequency of outdegree d is

$$f_d = r_{d-1} - r_d \tag{3.4}$$

Now, substitute for r_{d-1} and r_d their values according to the rank law (Equation 3.2). This gives

$$f_d = (\frac{d_v - 1}{\beta})^{\frac{1}{R}} - (\frac{d_v}{\beta})^{\frac{1}{R}} \tag{3.5}$$

To simplify further,

$$\boxed{f_d = \beta^{-\frac{1}{R}}[(d_v - 1)^{\frac{1}{R}} - d_v^{\frac{1}{R}}]} \tag{3.6}$$

This relationship is itself a power law that associates frequency and outdegree. Figure 3.9 shows the log-log plot of this derived outdegree law (Equation 3.6). We have derived an outdegree power law from the rank power law.

3.4.2 Outdegree law $\Longleftrightarrow$ new rank law

Starting from the outdegree law, we attempted to derive a power law for the rank-outdegree relationship. Our end result is a rank law which is not a power law. If our reasoning is correct, then this shows that the rank law can not be derived from the outdegree law. As discussed earlier, we believe that the new rank law, which we derive from the outdegree law, is more accurate than the original rank law in that it fits actual topology data better.

First, note that

$$\sum_{d=1}^{MaxD} f_d = N \tag{3.7}$$

where N is the total number of nodes in the graph, $MaxD$ is the maximum outdegree and f_d is the number of nodes with outdegree d, or the frequency of outdegree d. The minimum frequency is 1, so we must make sure that $f_d = \alpha d^O \geq 1$ (Equation 3.3). Substituting, we see that

$$MaxD = e^{-\frac{\log \alpha}{O}} \tag{3.8}$$

From the definitions of rank and frequency, we get

$$r_v = 1 + \sum_{d=d_v+1}^{MaxD} f_d \tag{3.9}$$

That is, the rank of the nodes with outdegree v is equal to one plus the total number of nodes with outdegree bigger than d_v. Using the outdegree law (Equation 3.3), we

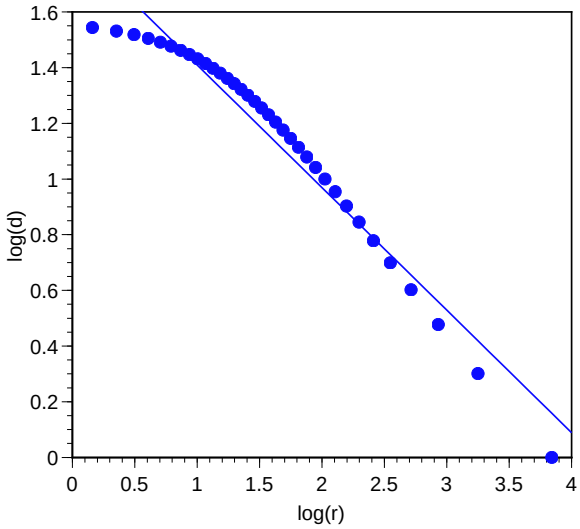


Figure 3.10: Log-log plot of derived d-r law.

get

$$r_v = 1 + \alpha \sum_{d=d_v+1}^{MaxD} d^O = 1 + N - \alpha \sum_{d=1}^{d_v} d^O \tag{3.10}$$

If we assume that O is a negative rational number, as shown in paper [88], we can then derive the following relationship between rank and outdegree:

$$\boxed{r_v = 1 + N - \alpha[\zeta(-O) - \zeta(-O, 1 + d_v)]} \tag{3.11}$$

Here, $\zeta(t) = \sum_{n=1}^{\infty} \frac{1}{n^t}$ is the Riemann Zeta function, and $\zeta(-O, 1+d_v) = \sum_{n=1+d_v}^{\infty} n^O$.

Figure 3.10 is a log-log plot of this derived rank law. Surprisingly, this derived law is not an ideal power law—it is far from a straight line. If our derivation is correct, it is clear that the rank law does not follow from the outdegree law. Furthermore, our derived law is a better fit to the actual observed topologies than the rank law. A close look at Figure 3.10 shows that when rank $r >\approx 37$, the relationship between log rank and log outdegree is nearly a straight line, giving the appearance of a power law relationship. The divergence for $r \leq\approx 37$ is similar to that shown for the actual

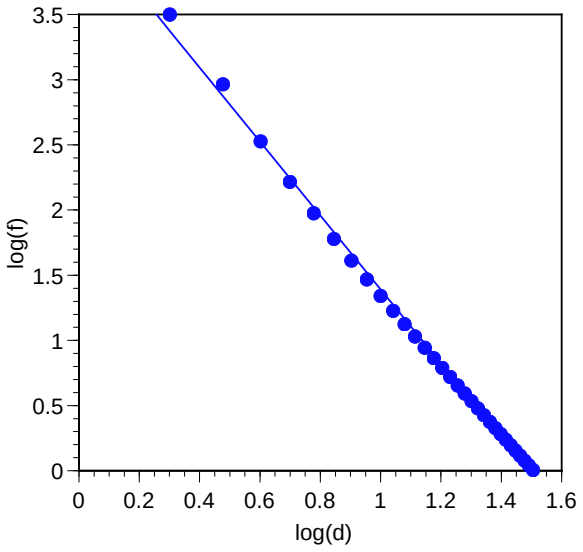


Figure 3.11: Log-log plot of derived d-f law using the new d-r law.

router level topology in Faloutsos, et al [88, Figure 6(b)].

Figure 3.11 shows the log-log plot of the outdegree and frequency relationship that can be derived from the new rank law (Equation 3.11) and Equation 3.4.

3.5 Annotations

In addition to producing a realistic topology that extends to the level of hosts, a grid generator must also annotate the topology with the attributes of its links, routers, switches, and hosts.

As to the network link bandwidth and latency, we can rely on the output of the underlying structure graph generator, leveraging work in the networking community as discussed earlier. As an alternative, we have also built into GridG an optional feature to explicitly generate network link bandwidth and latency. We assume that these distributions are different at the WAN, MAN and LAN levels, and give the user a mechanism to supply them. Similarly, the distributions of the switching bandwidth

Host	CPUS	CPU MHZ	Memory (MB)	Disk (GB)	Arch	OS	OS vendor	Current Load
1	512	1200	256	40	IA32	DUX	Sun	0
2	16	1000	512	800	PARISC	NetBSD	Microsoft	3
3	4	1600	512	160	SPARC32	DUX	RedHat	1
4	1	1800	65536	400	IA32	Solaris	Microsoft	2

Figure 3.12: Silly host configurations generated by the initial GridG annotator.

of routers are specified by the user in a configuration file. GridG then selects randomly based on the supplied distributions.

Host characteristics are considerably more complex. Our initial approximation is to treat each attribute of a host independently. The user supplies an empirical distribution for the attribute, and we select randomly based on that distribution. Host attributes are not independent, however. This oversimplified approach can generate unreasonable results. Figure 3.12 shows examples of silly combinations of host configurations that can result. These hosts appear silly because they violate our expectations about how the attributes of an individual machine are likely to be related. For example, we expect that most buyers will scale the memory of a machine with the number of CPUs, and that software vendors rarely sell their competitor's OS.

There also exist correlations between the attributes of machines that are near each other in the network topology. The obvious example is a cluster, in which tightly coupled machines have identical attributes. While we can support clusters via an additional graph transformation that explicitly creates them, there are other examples: servers in the same machine room are likely to have more in common with each other than with the client hosts that use them.

To capture such intra-host and inter-host attribute correlations, we extended GridG's annotator with a general engine that supports user-supplied conditional probability rules. For example, the user can assert that hosts with four or more processors have at least 4 GB of memory. In the following, we describe the en-

gine, a set of core "common sense" intra-host rules used to prevent silly hosts, an inter-host OS concentration rule for subnets derived from measurement, and show examples of sensible annotations. More rules are necessary and need to be derived from measurement. We describe our efforts to capture sufficient measurements to do so.

3.5.1 Annotation algorithm

Given a model of the distributions and correlations of host characteristics, it would be relatively straightforward to generate realistic host attribute information. However, at the present time we do not have such a complete model because of the difficulties in collecting a sufficient amount of data from which to infer the model. We appear to be the first to need to do this, and we have tried a number of techniques to acquire data, which we discuss later.

In light of the limited data, we have made GridG annotations conform to user-supplied rules and created a set of common-sense rules. In this way, the current generated host attribute information is reasonable on its face, and as new distributions and correlations are discovered, the user can add rules to GridG to make the generated grids conform. In the current implementation, user rules take the form of Perl functions. *Frames* for those functions are given to which the user can simply add their rules. For example, Figure 3.13 shows a function frame governing the correlation among CPU architecture, OS, the number of CPUs and CPU clock rate. Figure 3.14 shows two example rules added to this frame. The first rule says that Intel 32 and 64 bit architecture machines support no more than 4 CPUs, while the second rule says that for any architecture, the maximum number of CPUs per host is 1024. Figure 3.15 presents GridG's default rules in English. These rules can be removed or enhanced by the user and new rules can be added.

```
sub ARCHandOSandCPUrule {
   my ($arch, $OS, $numcpu, $cpuspeed) = @_;
   #add rules here


   #rules end here
   return 1;
}
```

Figure 3.13: Rule frame governing the correlation among CPU architecture, OS, number of CPUs, and CPU clock rate.

```
sub ARCHandOSandCPUrule {
   my ($arch, $OS, $numcpu, $cpuspeed) = @_;
   #add rules here
   #example rule: Intel arch can't support more than 4 CPU/host
   if(($arch eq "IA32") || ($arch eq "IA64")){
      if($numcpu > 4) {
         return 0;
      }
   }
   #example rule: Max num of CPU is 1024
   if($numcpu > 1024) {
      return 0;
   }
   #rules end here
   return 1;
}
```

Figure 3.14: Example rules.

1. One CPU will have at least 64M memory.
2. One CPU will have at most 4G memory.
3. More CPUs, more likely to have bigger memory.
4. One CPU will have at least 10G disk.
5. More CPUs, more likely to have bigger disk.
6. More memory, more likely to have bigger disk.
7. More disk, more likely to have more memory.
8. Intel&Windows box will have at most 4 CPUs, other type may have up to 1024 CPUs.
9. Intel Arch and other Arch have different distributions of CPU MHZ.
10. Host load is not correlated to any other attributes.

Figure 3.15: Default rules.

We believe that there are at least two types of correlations among the host's attributes. The first type is the correlations among the different attributes of an individual host, namely, correlations among the number of CPUs, CPU clock rate, memory size, disk size, etc. We can easily imagine important correlations of this type. For example, we assume a positive correlation between the number of CPUs and the total memory, and that machines with more CPUs are less likely to run a version of Windows. These assumptions appear in our default rules as shown in Figure 3.15. Some of those correlations are deterministic (e.g., a machine with 16 CPUs can't run windows as its operating system), and others are probabilistic (e.g., a machine with 32 CPUs is likely to have more memory per CPU than a 2 CPU machine, but not necessarily.)

The second type is a correlation between the attributes of machines that are near each other in the network topology, namely, correlations such as OS concentration

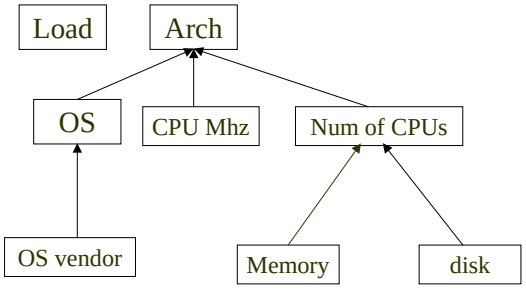


Figure 3.16: Dependence tree.

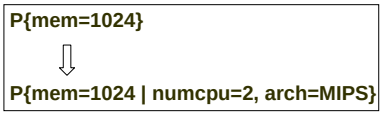


Figure 3.17: Example of conditional probability.

on IP subnets as described in Section 3.5.2 and shown in Figure 3.23. This type of correlation is mostly probabilistic.

To make the GridG annotator conform to the first type of correlations, we assume a dependence tree as shown in Figure 3.16. Host load is independent of other attributes in our model. Architecture is the root of the tree on the assumption that it is the most significant attribute, and that it largely decides what OSes can run on a host and how many CPUs it is likely to have. Clearly the OS vendor depends on the OS. We believe that memory and disk size is likely to grow with the number of CPUs on a machine.

With this dependence tree, we can make GridG conform to correlations by applying conditional probability, choosing the distribution of an attribute based on attributes picked before it. For example, we would first pick the architecture, then the number of CPUs based on that choice, and finally the amount of memory based on those two choices, as shown in Figure 3.17. This approach appears to work well

in practice, generating sensible hosts. In the following, we present in detail the algorithm used to annotate the hosts on a LAN. This process is repeated for each LAN in the topology. To make the description easier to understand, we present the flow chart of the algorithm in Figure 3.18.

1. If the OS concentration feature (Section 3.5.2) is turned on, then, for each IP subnet (generated LAN in our topology) , select the OS concentration percentage P from the user configuration file. Next, select the dominant architecture for the subnet according to the user configuration file. Select the dominant OS according to the dominant architecture for the LAN. There is one dominant architecture and OS for each LAN.

2. For each host, generate architecture for the host according to distribution specified in the user configurable file. If the OS concentration feature is turned on, change the host architecture to the dominant architecture with probability P.

3. Architecture is subdivided into several groups, such as the Intel Architecture family, MIPS, etc. Each group has its own specified distribution for OS, number of CPUs and CPU clock rate in the configure file. Using the distributions, generate the OS according to the architecture type. If the OS concentration feature is turned on and the architecture has been changed to the dominant architecture, also change the generated OS to the dominant OS.

4. Using the distributions, generate CPU clock rate and the number of CPUs according to the architecture type.

5. Apply the user rule governing the correlation among architecture, OS, number of CPUs and CPU clock rate. If the rule is not satisfied, go back to the last step.

6. The number of CPUs is subdivided into several groups, such as the number of CPUs, $n \leq 4$, $4 < n \leq 32$, $32 < n \leq 128$, etc. Each group has its own memory and disk size distribution specified in the configuration file. The configuration is such that a group with a large number of CPUs has distributions for memory and disk size that are of higher mean than those for a group with a small number of CPUs. Within each group, we apply a *promotion probability function* and and a *promotion rate function* so that a machine with larger number of CPUs is more likely to increase its memory and disk. Optionally, we apply a *degradation probability function* and *degradation rate function* so that a machine with smaller number of CPUs is more likely to decrease its memory and disk, which is a strengthening mechanism for promotion probability and rate functions. We discuss these functions in more detail below.

7. Apply the user rule governing the correlation among the number of CPUs, memory size, disk size, and OS, architecture, and CPU clock rate. If the correlations are not satisfied, go back to last step.

8. Generate OS vendor according to its OS.

9. Apply the user rules governing the correlation between OS and OS vendor. If the correlation is not satisfied, go back to last step.

10. Generate a load value for the host according to the specified distribution in user configuration file.

11. Apply the user rule governing the correlation between load and other attributes. If it is not satisfied, go back to last step.

12. Apply the overall user rule governing architecture, OS, number of CPUs, CPU

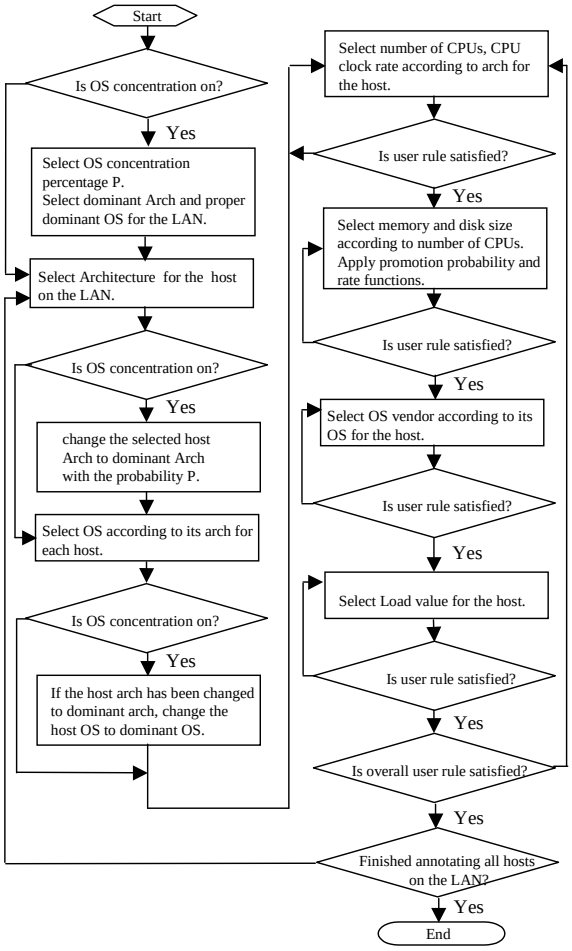


Figure 3.18: Flow chart of the algorithm.

clock rate, memory size, disk size, OS vendor and load of the host, if it is not satisfied, go back to step 4.

13. If all the hosts on the LAN are annotated, go to next step, otherwise, go to step 2.

14. Terminate.

The promotion probability function and the demotion probability function map

```
sub promotionProb
{
   my($numcpu, $maxcpu, $mincpu) = @_;
   my $pp100;
   $pp100=100*($numcpu-$mincpu)/($maxcpu-$mincpu);
   return $pp100;
}
```

Figure 3.19: Linear promotion probability function.

```
sub promotionProb
{
   my($numcpu, $maxcpu, $mincpu) = @_;
   my $pp100;
   $pp100=100*sqrt(($numcpu-$mincpu)/($maxcpu-$mincpu));
   return $pp100;
}
```

Figure 3.20: Power promotion probability function.

from the number of CPUs to a probability PP, the probability that memory and disk size will be increased (promotion) or decreased (demotion) based on a host's number of processors. A larger number of processors leads to an increased promotion and a decreased demotion probability. Paired with these are the promotion and demotion rate functions, which determine the extent to which memory and disk size will be changed. The default rate function doubles/halves the memory and disk size, but user can specify their own rate and promotion/demotion functions. Figure 3.19 shows a linear promotion probability function, while Figure 3.20 shows a power function where the probability of promotion increases faster with the number of CPUs.

Together, the promotion/demotion probability and rate functions control the correlation coefficient between number of CPUs and memory or disk sizes. Shown in Figure 3.21 is the influence of promotion probability and rate functions. If stronger

Promotion function	Correlation coefficient
None	0.69
Linear	0.73
Power	0.76

Figure 3.21: Influence of promotion probability and rate functions on correlation coefficient between number of CPUs and memory or disk.

Host	CPUS	CPU MHZ	Memory (MB)	Disk (GB)	Arch	OS	OS vendor	Current Load
1	512	1200	65536	10240	MIPS	FreeBSD	FreeBSD	9
2	16	1000	8192	800	PARISC	NetBSD	NetBSD	4
3	4	1600	1024	160	SPARC32	Solaris	Sun	1
4	1	1800	512	80	IA32	Win2k	Microsoft	3

Figure 3.22: Sensible hosts generated by current GridG annotator.

correlations are required, both promotion and degradation probability and rate functions are chosen to increase faster with the number of processors.

This algorithm and the default rules work well and generate results that are sensible. More rules can be added as they are discovered. Figure 3.22 shows a few reasonable hosts generated by the current GridG implementation.

3.5.2 OS concentration rule

The nmap port-scanning tool [100] has the ability to determine a host's operating system based on how its TCP/IP implementation behaves. We used nmap to scan 10 different class C IP networks, both at Northwestern and belonging to a company that maintains a popular web site. In each subnet we oberved OS concentration to

Organization	Subnet	Dominant OS	Percentage of concentration	Percentage of recognized OS
CS	1	Windows	79/95 = 83.2%	78%
Department	2	Windows	31/37 = 83.8%	81%
	3	Linux	40/40 = 100%	100%
ECE	4	Solaris	67/76 = 88.2%	65%
Department	5	Solaris	41/41 = 100%	94%
	6	Solaris	21/21 = 100%	96%
	7	FreeBSD	214/214 = 100%	87%
A popular	8	FreeBSD	187/187 = 100%	89%
web site	9	FreeBSD	191/192 = 99%	92%
	10	FreeBSD	72/73 = 99%	91%
Total machines		976		

Figure 3.23: OS concentration observed in IP subnets.

various degrees, that is, there was typically one dominant operating system.

Figure 3.23 shows the data for all the 10 subnets. Nmap couldn't recognize every host's OS from its TCP/IP fingerprints because nmap doesn't have fingerprints for all versions of every OS. However, notice that even if we assume that all the unrecognized OSes were of a type other than the dominant OS, we can still assert that OS concentration exists in all 10 subnets. Subnet 3 is known to be a Linux cluster but there is no cluster in subnet 4, 5 and 6.

People are very sensitive to the port scanning and therefore we have had to limit our activity. However, given this sample of 976 machines and 10 subnets, it seems likely that such OS concentrations occur on many subnets. It also appears sensible given that subnets are owned by individual organizations, and many organizations have standardized operating systems and even machines. Another factor is the existence of clusters in which every node has identical hardware and software. We have added this OS concentration behavior to the GridG rulebase as an optional feature.

3.6 Conclusions

We have presented GridG, a tool for generating synthetic computational grids. GridG produces structured network topologies that obey the power laws of Internet topology. While developing GridG's topology generator, we found that two of the power laws (rank and outdegree exponent laws) are in conflict. We derived a new rank law from the outdegree law that conforms well with published data on actual topologies and has a power law like range. We speculate that this new law is a better approximation of rank behavior.

The topology annotation component of GridG can annotate the network according to user supplied empirical distributations and user conditional probability rules. Two

kinds of correlations among hosts attributes are considered and built into GridG: correlations between an individual host's attributes and correlations between attributes on nearby hosts. We developed a basic set of rules that capture common sense intra-host correlations. Through nmap based measurement, we observed OS concentrations in all the IP subnets we probed. We added this as an inter-host rule.

Developing additional rules will require more host measurement data to analyze. We have tried a number of techniques for acquiring such data and have been largely unsuccessful. We continue to explore ways to acquire a rich set of measurements from which to derive a larger rulebase for GridG annotations.

GridG was used to evaluate the RGIS system and several other systems that require synthetic annotated topologies [41, 140].

Chapter 4

Query Rewriting Techniques

This chapter describes the motivation, design and implementation of our query rewritting techniques that enable fast processing of complex queries.

4.1 Introduction

A powerful feature of RGIS is that users can write queries in SQL that search for complex compositions of resources, such as groups of hosts and network resources, that meet collective requirements. These queries, as they are expressed using joins, can be very expensive to execute, however. In response, we have introduced four extensions to the SQL select statement:

- Nondeterminism: the result set is a random sample of the result set of the full query. This extension is processed automatically.
- Scoping: the query is limited to a region of the modeled network, giving a result set that is a deterministic subset of the full query. This extension is processed automatically.
- Approximation: the query's joins are eliminated and new constraints are added, giving a result set that is a deterministic subset of the full query. This extension

requires humans to process it and is used in "canned queries", pre-written queries for commonplace inquiries.

- Time bounding: the query (or the RGIS server) can place limit on the wallclock time of the query. This extension is processed automatically.

The goal of these extensions, which can be used orthogonally, is to make it possible for the user (and RGIS) to trade off between the running time of a query (and the load it places on an RGIS server) and the number of results returned. The extensions require no changes to the RDBMS. We argue that it is sufficient and appropriate for a GIS to provide partial results, particularly a random sample.

In this chapter, we describe our query extensions, and their implementation. We also present a performance evaluation of our implementation that shows that a meaningful tradeoff between query processing time and result set size is possible using our techniques, that we can use that tradeoff to keep query running time largely independent of query complexity, and that we can bound the response time of a query. These results demonstrate that we can deliver powerful relational queries to users, which is a necessary support for our general argument that GIS systems should be built on a relational data model.

4.2 Addressing the query problem

Parallel and distributed applications are not interested in individual resources per se, but rather in compositions of them. For example, suppose a data parallel program has been compiled to run on four processors. At startup, it will want to ask questions such as "find me a set of four unique hosts which in total have between 0.5 and 1 GB of memory and which are connected by network paths that can provide at least 2 MB/s of bandwidth with no more than 100 milliseconds of latency." Such questions can be

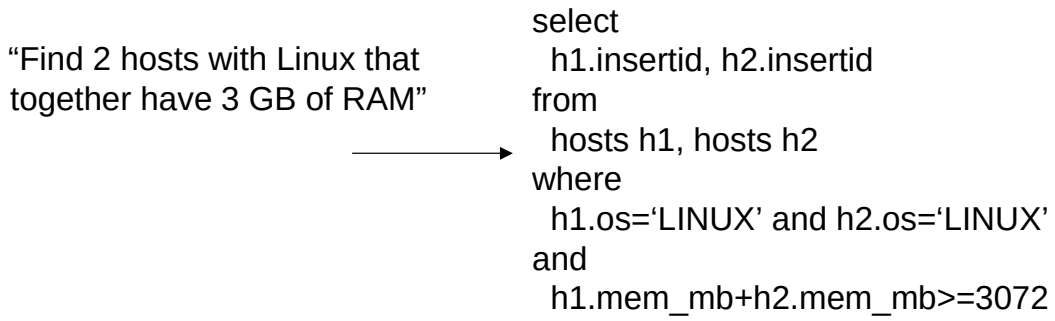


Figure 4.1: An RGIS Query.

readily posed to RGIS using the SQL language. Indeed, SQL lets the application or user combine multiple resources in arbitrary ways. Figure 4.1 shows a simple RGIS query that is searching for all pairs of hosts that both run the Linux operating system and together have at least 3 GB of RAM. For clarity in this example query, we omit the constraint that the two machines be distinct. In our evaluation, where we use similar queries, we introduce this constraint.

Because the query is declarative, there is significant room for the query optimizer in the RDBMS to make the query efficient. It also means that the query is independent of the underlying RDBMS implementation that is being used. The same query may run today on a basic Windows implementation of Oracle, while tomorrow it may be run by a parallel implementation of Oracle on a cluster or SMP. The query is also independent of the indices created by the database or by the database administrator. Hence, if this form of query becomes common, the administrator can create indices to speed it up. Finally, if queries are written in ANSI standard SQL, the underlying RDBMS can be changed without changing the query. Common queries can also be

provided as materialized (i.e., precomputed) views on the database.

Unfortunately, queries such as the one in Figure 4.1 can be very expensive to execute, especially as the number of joins (number of hosts in the query in this example) grows. In the worst case, the query cost can grow exponentially with the number of joins, even with our carefully chosen indices. Not only must individual queries not take long periods of time to execute, an RGIS server must also be able to handle the query workload of a whole site. If we supported such queries directly, we would very soon begin disappointing users and overloading the RGIS server.

4.2.1 Deficiencies of limited deterministic queries

One approach to reducing the work involved in answering a query is to limit the size of the result set that is returned (using "rownum<N" as part of the where clause in Oracle, or MySQL's "limit" clause, for example). The query would then only run until the specified number or rows was returned. We'll refer to this as a limited deterministic query. It is intuitive why a limited deterministic query would be reasonable from an application's perspective. The application making the query of Figure 4.1 is not interested in *all* pairs of hosts that meet its requirements. It is merely trying to find *some* pairs that do.

Limiting result set size has two serious problems, however. First, the computational time for the query is not directly proportional to the result set size—it depends on the data distribution in the input tables. Continuing the example, the rarer that pairs of hosts that meet the requirements are, the longer a limited query will run. In the worst case, the RDBMS may have to scan the cross product of the hosts table to the very end to find a single match, making this query as expensive as one without limits. The other problem with the limited deterministic query is that the query returns exactly the same results each time it is run. Suppose there are 10

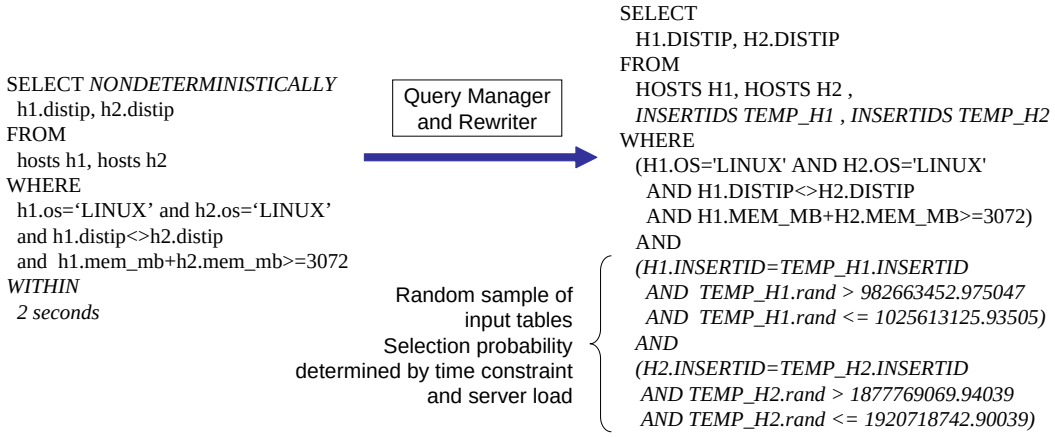


Figure 4.2: Nondeterministic query and its implementation.

pairs of hosts that are appropriate, but the query is limited to one pair. Different applications making the same query would end up choosing the same pair, leading to contention. In general, limited deterministic queries can lead to certain resources suffering contention hotspots merely due to where they happen to be placed in the database.

4.2.2 Nondeterministic queries

The first query optimization technique we use to limit query running time (and load) is nondeterminism. This technique also helps to avoid contention. The left-hand side of Figure 4.2 shows a nondeterministic, time-bounded version of the earlier query. The additions to the query are shown in italics. A nondeterministic query returns a random subset of the full set of query results. The computational cost of the query is controlled by the *selection probability*, which is derived from the time limit of the "within" clause and the current load on the RGIS server, as described in Section 4.4. The selection probability is the probability that a row of an input table will be included in the join. Intuitively, as the load increases or the time limit shrinks, the

selection probability shrinks. As the selection probability shrinks, so does the amount of work needed to perform the query and the expected number of rows returned by it. Each time the query is run, the rows returned are different while the computational cost of getting them stays roughly the same.

We implement nondeterministic queries using a combination of query rewriting, schema extensions, indices, and randomness. No changes to the RDBMS are needed. When a nondeterministic query is posed to the query manager/rewriter, it determines a selection probability, p, for the query. Associated with each object inserted into the database (in the insertids table), at insert time, is a random number, ranging from R_{min} to R_{max}, for a range, $R = R_{max} - R_{min}$. We translate the p into a subrange, $r = pR$. Next, for each input table T_i in the query, we add a where clause that constrains rows in that table to have associated random numbers in the range $[s_i, s_i + r)$, where s_i is chosen from a uniform random distribution over $[R_{min}, R_{max}]$ at query translation time. Notice that in implementing this selection, each input table is joined with the insertids table to recover the random number. While these equijoins are likely to be fast given the RGIS indices, it does double the number of joins in the query. If the underlying RDBMS provides a sampling operator, we can use it to avoid this doubling. Oracle has recently added such an operator, but we do not use it in our evaluation in this chapter.

It is important to note that this approach has a grouping effect that should ideally not occur in an implementation of random sampling. Two objects inserted into the database may be assigned nearby random numbers. Hence, if one is chosen, there is a greater likelihood that the other will also be chosen. With small selection probabilities, the effect is negligible. However, to ameliorate it, we regularly "reshuffle" the insertids table, assigning new random numbers to objects.

In addition to the random numbers associated with each object in the database,

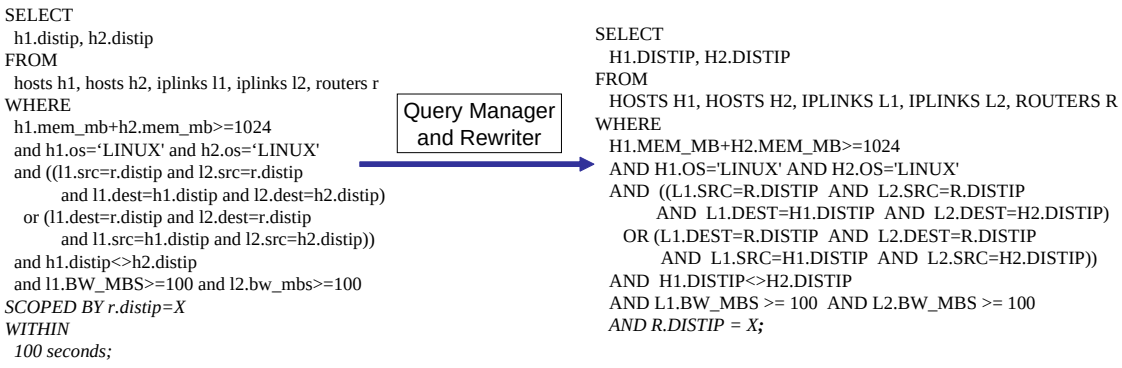


Figure 4.3: Scoped query and its implementation.

the database also includes indices on these random numbers and on their associations with other attributes. These indices help to make the random sampling fast.

4.2.3 Scoped queries

It is a common misconception that, unlike hierarchical data models, it is impossible to scope queries in the relational data model. In actuality, it is schema-dependent. Because the RGIS schema models the network, it is possible to scope queries with respect to the network of the grid. Scoping can dramatically lower the cost of a query.

Can a user scope to a network component? Yes. It is as straightforward as scoping to a host or a site, which is common in hierarchical GIS systems. If the user chooses a host, he can issue a quick RGIS query to determine its first hop router (or layer 2 switch). If the user has access to the host, he does not even have to use RGIS—the gateway router is in the routing table of the host. Scoping to a site is done by using the site's readily available CIDR block as a prefix match against host IP addresses, an operation that turns into a fast range scan on the database.

Figure 4.3 shows a scoped version of our running example query along with its

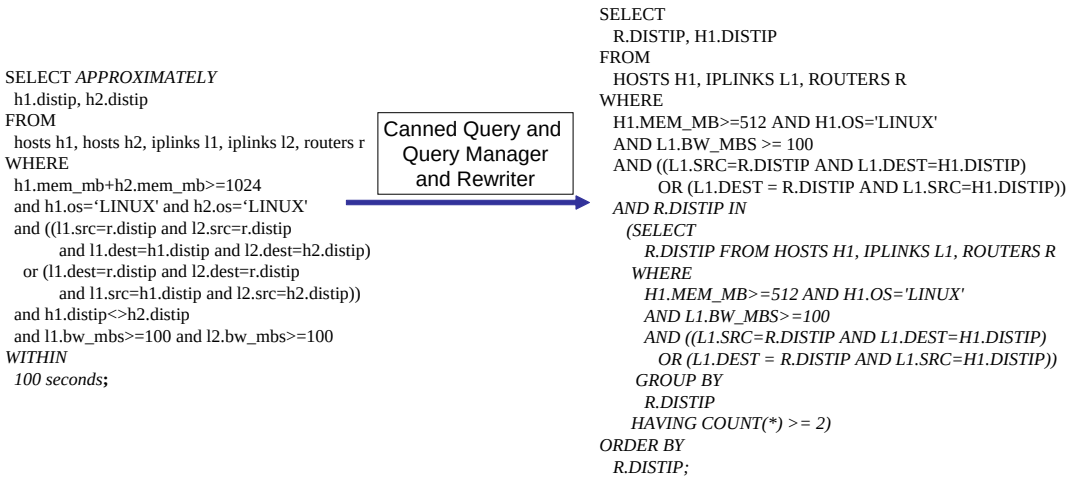


Figure 4.4: Approximate query and its implementation.

implementation. Here, we have extended the previous query to include the network: we are asking for pairs of hosts that are connected to the same router. Scoping is achieved by constraining the choice of router to one particular router. A scoped query may return no results if the router does not have a set of machines that meets the constraints. The user may issue the query iteratively with a list of routers known to him, or have the system issue it repeatedly with randomly chosen routers until a match is found or the time limit is exceeded.

4.2.4 Approximate queries

The main idea behind approximate queries is to minimize the number of joins by rewriting a SQL query with tighter constraints that returns a result set that is a deterministic subset of the full result set. Consider an example query in which we seek to find N machines with total memory $\geq B$ MB. In this simple example the original SQL query will execute an N way join. A valid approximation of this query is to find N machines each with memory $\geq B/N$ MB. The approximation is valid in

that those N machines must also be a row of the result set of the original query. It is an approximation in that the constraints are stronger. A solution in which $N/2$ machines have $4B/3N$ MB and $N/2$ have $2B/3N$ would be found by the original query but not by the approximate query. The upshot is that we can quickly compute a result set that is a deterministic subset of that of the original query using a *fixed* number of join operations.

Note that we must know the semantics of the particular resource (hosts and their memory) to know whether the transformed query makes sense. For this reason, the approximation rewrite is done by hand and supplied to users in the form of canned queries.

Figure 4.4 shows an approximate version of our running example (again, including the network) along with its implementation. Notice that the number of tables joined is three (with an additional 3-way join in the sub select). This is fixed regardless of the number of hosts being searched for. Such a "cluster finder" query that looks for 64 machines would also involve the same joins. In the query, the sub select returns those routers that have at least $N = 2$ appropriate machines attached to them, and the main select returns all those machines ordered by their router.

4.2.5 Combining query techniques

Scoping, approximation, and nondeterminism are orthogonal and can be combined. Combining techniques often leads to a simplification of the ultimate implementation. For example, combining scoping and approximation leads to very efficient and very small queries. Figure 4.5 illustrates this form of query and its implementation. Here, the entire sub select of the approximate query of the previous section is replaced by the router to which the query is scoped. The query now involves only a three-way join. Instead of looking for two machines with total memory of 1GB attached

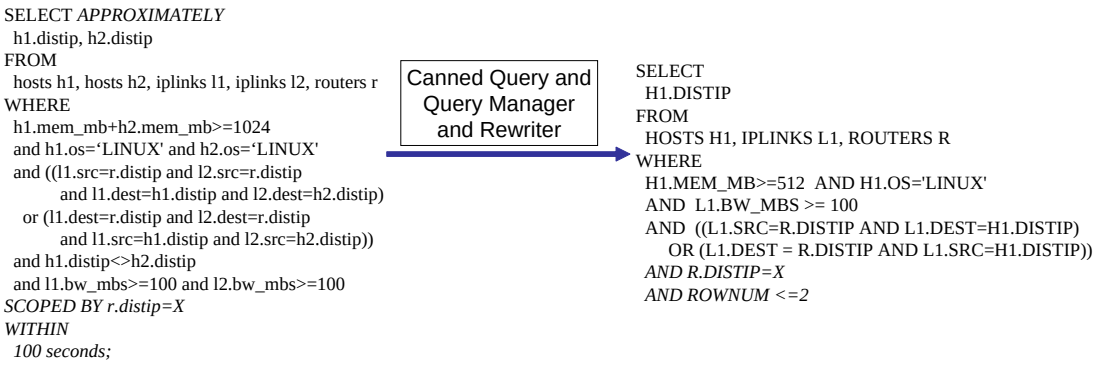


Figure 4.5: Scoped approximate query and its implementation.

to a common router, the approximate query looks for two machines that each have memory ≥ 0.5 GB, both attached to a common router, which is a deterministic subset of the original query results. This query can be extended to N host clusters in the following way.

To find a cluster of N qualified machines that are connected to one router, we first randomly choose a router (or iterate through a list provided by the user). We then perform the three-way join query, limiting the number of rows returned to N, to find possible combinations of hosts in the LAN. This process repeats until we find a router with N appropriate hosts attached, we run out of routers with no results found, or we run out of time. Notice that for any arbitrary N, the query remains a fixed three-way join. With the original SQL query, we would need a $2N + 1$-way join to find a N machine cluster, which is of course extremely expensive. If random routers are chosen, then we essentially randomly sample the result set of the approximate query (Figure 4.4), which is itself a deterministic subset of the full result set.

4.3 Evaluation

We evaluated our query techniques using versions of the queries described in the previous section, studying the tradeoffs possible between query run time and result set size. Similar to earlier GIS evaluation work [237], our query run time metric is the average response time from the users perspective. Ultimately, RGIS's goal is to use these techniques to return at least one valid result within the time limit specified by the query or the system.[1]

4.3.1 Experimental setup

Our experimental infrastructure is based on Oracle 9i Enterprise Edition running on Red Hat Linux 7.1 on a dedicated Dell PowerEdge 4400 server. The server has two 1 GHz Xeon processors, 2 GB of main memory, and a PERC3DI RAID controller producing about 240 GB of RAID 5 storage over eight 36 GB U3 SCSI disks.

To evaluate queries, we must first populate our database. We did this using GridG [145], which was described in details in chapter 3.

Host memory sizes and network properties are most critical for the types of queries we use in our evaluation. For memory sizes, we studied (anonymized) data dumps from the MDS servers running on several large grids, and data provided by the BOINC project at Berkeley. The largest dataset was one collected by Smith, et al [204]. We extracted memory sizes from the Smith dataset and then configured GridG to generate hosts with the same memory size distribution.

We generated networks using specific parameter values found by Faloutsos, et al, in their dataset on routers in the Internet [89]. GridG network topologies are

[1]The original and rewritten queries of this section are available from http://urgis.cs.northwestern.edu/example_queries.

configured using eight parameters. Six determine the hierarchical structure of the generated grid (these are passed to the underlying Tiers generator [82]) while the remaining two determine the parameters of outdegree power law of Internet topology (our extension). The eight parameters are: α (outdegree law constant), O (outdegree exponent), NW (maximum number of WANs), NM (maximum number of MANs per WAN), NL (maximum number of LANs per MAN), SW (maximum number of nodes per WAN), SM (maximum number of nodes per MAN), and SL (maximum number of nodes per LAN). We provide the actual parameter values in the relevant sections.

4.3.2 Nondeterministic queries

We evaluated nondeterministic queries by studying how the query run time and the result set size depends on the database size, the selection probability, and the complexity of the query. We use two different queries. The first looks for groups of hosts that together have a given amount of memory. The second looks for two directly connected hosts running the same operating system.

"Find groups of N distinct Linux hosts with at least B total memory"

This query our running example from Section 4.2.2, generalized to find N different hosts. Note that while such queries can become quite complex as the number of hosts grows, they can be automatically generated very easily.

Using the memory size distribution, we generated grids of 50,000, 500,000, and 5,000,000 hosts with GridG. For each grid, we evaluated 2, 4, 8, and 16 host versions (using 2, 4, 8, and 16-way joins) of the query, varying selection probabilities. We ran each query five times, measuring the query running time and the number of rows returned by the query. We report the mean, minimum, and maximum of the five runs. Figure 4.6 shows all the performance data, which we elaborate on below.

			Number of rows selected			Query Time (seconds)		
Hosts	Joins	Selection Probability	Average	Min	Max	Average	Min	Max
50K	2 way	deterministic	-	-	-	> 1 hour	> 1 hour	> 1 hour
50K	2 way	limited deterministic 1 or 10 rows	1,10	1,10	1,10	0.13	-	-
50K	2 way	0.001	562	261	933	0.42	0.376	0.463
		0.01	55729	50093	66803	17.8	17.3	18.7
	4 way	0.0001	527	111	1343	0.35	0.3	0.45
		0.0005	131791	69357	181139	26.3	12.1	34.2
	8 way	0.00005	1156	0	3103	0.72	0.53	0.99
		0.0001	178853	1920	597911	29.3	0.83	102
	16 way	0.00001	0	0	0	6.67	6.64	6.7
		0.00005	298598	0	1492992	81.3	6.64	380
500K	2 way	0.0001	566	299	802	0.81	0.53	0.94
		0.0005	13048	10620	16168	5.34	4.42	6.73
		0.001	57524	13048	62340	18.3	16.3	19.43
		0.002	216382	210290	220030	73.0	70.0	76.0
	4 way	0.00001	541	0	1293	0.36	0.26	0.48
		0.00005	143226	62853	219366	26	18.2	32.9
	8 way	0.000005	1231	0	6848	0.69	0.53	1.3
		0.00001	54127	9368	130971	9.1	2.2	19.9
	16 way	0.000001	0	0	0	6.65	6.63	6.7
		0.000005	804533	0	3930540	115.6	6.63	523.6
5000K	2 way	0.00001	507	380	635	1.1	0.96	1.13
		0.0001	60315	52707	70613	22.4	20.9	23.9
	4 way	0.000001	235	20	624	0.55	0.46	0.65
		0.000005	189920	109533	322668	23.2	17.5	35.4
	8 way	0.0000005	551	138	1296	0.77	0.71	0.87
		0.000001	272704	110614	674554	28.9	13.9	68.2
	16 way	0.0000001	0	0	0	6.7	6.69	6.71
		0.0000005	121473	0	330884	31	6.7	78.1

Figure 4.6: Performance of nondeterministic queries with different sizes of grid, different numbers of hosts, and different selection probabilities.

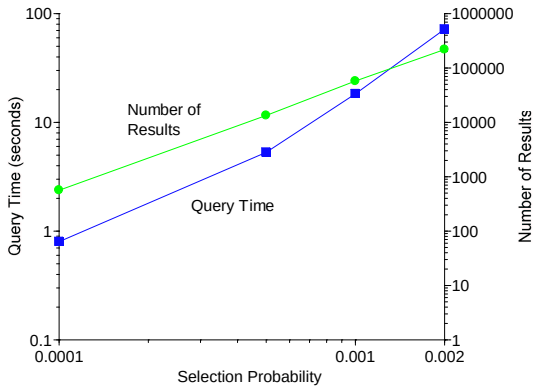


Figure 4.7: Query time and number of selected rows versus selection probability.

In addition to the nondeterministic queries, we also evaluated the performance of deterministic and limited deterministic versions of the simplest query (2 way join) on the smallest number of hosts (50K), data that occupies the first two rows of Figure 4.6. Notice that the purely deterministic query, which will eventually return all possible results, requires over an hour of running time. Oracle's query plan makes full use of the available indices.

The limited deterministic query, which returns the first result, or the first 10 results, finished very quickly (0.13 s), but always returned the same results. The nondeterministic version of the same query executes more slowly, taking about twice as long even with a very low selection probability. However, following the discussion of Section 4.2.2, we now get a different set of results each time we run the query, and the query will do a fixed amount of work each time it is run. It is slower because there are two additional equijoins with the insertids table, as can be seen in Figure 4.2. This overhead is the cost for implementing random sampling above the database. A database engine that supports random sampling will not pay this penalty.

To better illustrate our results for nondeterministic queries, we show two slices

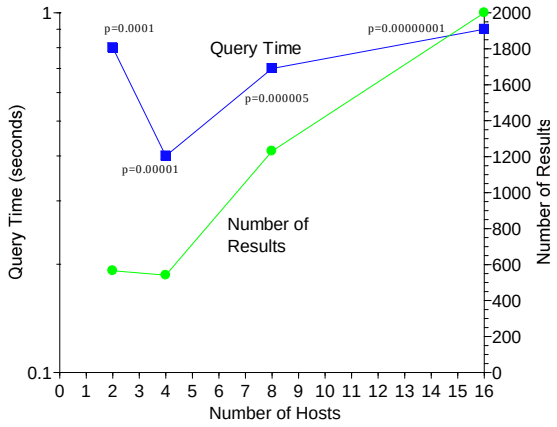


Figure 4.8: Query time and number of selected rows versus number of hosts.

through the table. We use the 500,000 host grid. Figure 4.7 shows the average number of results and the average query time for the two host version of the query as a function of the selection probability. The left hand scale corresponds to the query time, while the right hand scale shows the number of results. Note that all scales are logarithmic. These results show that it is possible to meaningfully trade off between query processing time and result set size. The point here is that it is possible to vary the query time and the result set size over several orders of magnitude by modulating the selection probability.

Figure 4.8 shows a second slice through our data. Once again, we have used the 500,000 host grid and show the average query time and result set size, but here we vary the complexity of query (the number of hosts asked for) and choose selection probabilities to try to keep the query time as constant as possible. The point here is that it is possible to use the selection probability to control the query time largely independent of query complexity.

Hosts	α	O	NW	NM	NL	SW	SM	SL
10K	8.915	-2.49	1	20	10	10	10	50
50K	8.915	-2.49	1	100	10	10	10	50
100K	8.915	-2.49	1	100	20	10	10	50

Figure 4.9: Parameters passed to GridG to generate grids for nondeterministic query evaluation.

"Find pairs of directly attached Linux hosts with at least total memory B"

Because RGIS also stores information about the network topology of a grid, we can include topology in our queries. For example, we can find the shortest paths between a pair of hosts, or all pairs shortest paths, or a group of hosts that are tightly connected.

Our query here tries to find all pairs of hosts that are directly attached (at layer 3). The machines must have the same operating system and must have a total memory of at least 1 GB. We generated different networks, ranging from 10,000 to 100,000 hosts for this query. The network parameters given to GridG are summarized in Figure 4.9.

Figure 4.10 shows the average query time versus selection probability for different size grids. Figure 4.11 shows the average result set size for the same parameters. For this range of selection probabilities, query time increases approximately linearly with selection probability, while the result set size increases slightly faster. These results provide more evidence that it is possible to use selection probability to trade off between result set size and query time for queries about grids with typical topological and memory size distributions.

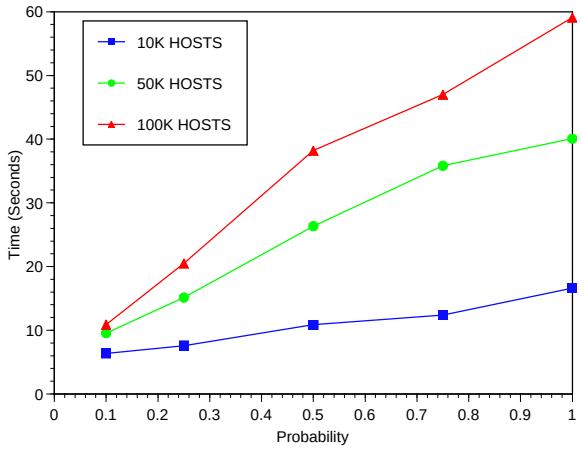


Figure 4.10: Query running time versus selection probability for nondeterministic network query.

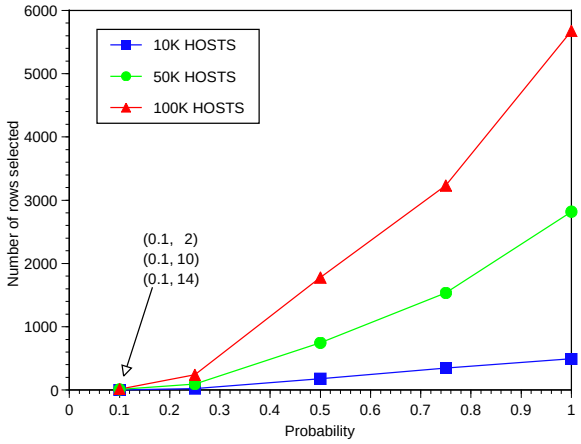


Figure 4.11: Number of selected rows versus selection probability for nondeterministic network query.

Hosts	α	O	NW	NM	NL	SW	SM	SL
9,800	8.915	-2.49	1	7	7	40	30	200
101,200	8.915	-2.49	1	23	22	40	30	200
980,000	8.915	-2.49	1	70	70	40	30	200

Figure 4.12: Parameters passed to GridG for generating grids to evaluate scoped, approximate, and scoped approximate network queries.

4.3.3 Scoped, approximate, and scoped approximate queries

We evaluated how the performance of scoped, approximate, and scoped approximate queries on RGIS depends on the database size, and the complexity of the query. The queries we use are nearly identical to those of the previous section. The first looks for sets of hosts that meet a total memory requirement and total disk requirement, the latter requirement being the difference from the previous section. The second looks for groups of hosts that additionally are tightly coupled on the network (i.e., clusters.) Here, the coupling is not direct attachment, but sharing a common first hop router.

The network parameters of the synthetic grids used in this section are described in Figure 4.12. The memory distributions are as before.

"Find groups of N distinct Linux hosts with at least total memory B and disk D"

Here we evaluated the performance of original and scoped approximate versions of a query that searchs for N hosts with a total memory $\geq 512N$ MB and total disk $\geq 80N$. Figure 4.13 shows the average response time of both versions of the query for the largest grid size. In all the different sized grids, we see that the scoped approximate query time is independent of the number of hosts in the group, while the deterministic query time grows very fast with the number of hosts. In all cases, results

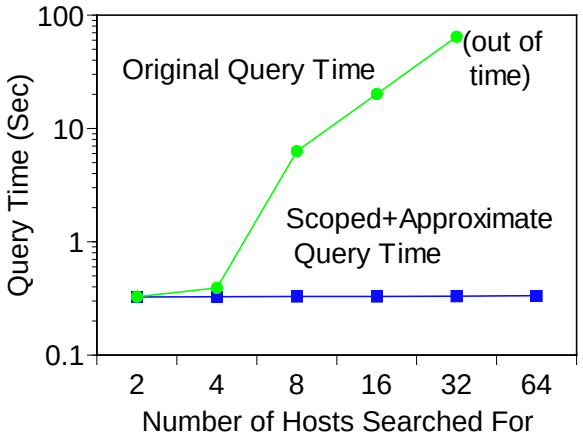


Figure 4.13: Host group query response time vs. number of hosts with 980,000 hosts (scoped approximate query).

were returned. Clearly, human query creation (the approximate form of the query) combined with automated transformation (scoping) can result in query performance that scales very well with problem size.

"Find groups of N distinct Linux hosts with link bandwidth L and total memory B attached to a common router."

This query searches for an N host Linux cluster with link bandwidths ≥ 100 Mbps, and total memory $\geq 512N$. Here, we measured the response time of original, scoped, approximate, and scoped approximate versions of this "cluster finder" query as a function of database size and query complexity (number of hosts). Because of time limitations, our data compares all versions on the small grid (9,800 hosts), but only the original (slowest) and scoped approximate (fastest) versions on the larger grids.

Figure 4.14 shows a table of the average response times for each of the different versions of the query run on the small grid. When going from original SQL queries to scoping, the decrease in response time is small due to the presence of N-way

Cluster Size	*Technique* Original SQL query	Scoped query	Approx query	Scoped + Approx query
2	21.44	2.27	7.62	1.16
4	>7200	2047.93	7.48	1.32
8	>9000	>3600	7.46	1.43
16	N/A	>3600	7.51	1.45
32	N/A	>3600	7.65	5.96
64	N/A	>3600	>120	9.58

Figure 4.14: Cluster finder query times in seconds for the four query techniques for a database populated with 9,800 hosts. In the figure, *N/A* represents those tests that were not run due to expected extremely long query times.

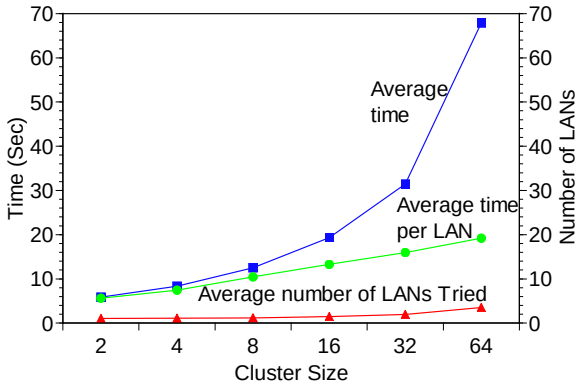


Figure 4.15: Cluster finder query time vs. cluster size with 980,000 hosts (scoped approximate query).

joins in each. Moving to approximate queries results in much better scaling with N. Only after the $N = 32$ does the response time become unreasonable. The scoped approximate queries have dramatically lower response times, and scale nicely with increasing cluster sizes.

Figure 4.15 provides more detail for the scoped approximate query running on the largest grid (980,000 hosts). Here we use randomized scoping, repeating the query with a different random router until a match is found. Because the scoped

approximate queries always perform a three-way join, regardless of N, the larger that N is, the more benefit they provide. As the grid grows in size, the expected number of LANs that must be searched decreases. In addition, the average request time per LAN scales relatively linearly as the cluster size increases. In the worst case we studied (64 node cluster, 980,000 hosts) the average time of the request is less than 70 seconds, which is remarkably faster than the >> 2000 second times seen by the original query. Again, automated query transformations can dramatically improve performance, with human intervention providing even more gains.

4.3.4 Queries under load

GIS systems must scale in the presence of multiple concurrent users and under update load [180]. To study RGIS's performance in this respect, we used the scoped approximate queries described in Section 4.3.3 running on the small grid (9,800 hosts). The update load consisted of a process continuously doing transactional updates of the disk size of randomly selected hosts.

In Figure 4.16, multiple users are iteratively issuing scoped approximate queries that are each searching for Linux clusters with 64 nodes, link bandwidths of ≥ 100 Mbps, and a total memory of 512×64 MB. We plot their average response time as a function of the number of concurrent users. Figure 4.17 is identical except here we run the host group query trying to find N hosts with a total memory $\geq 64 \times 512$ MB. We observe that the updating process can slow down the information server, but the effect is small and no worse than 10% in most circumstances. How well multiple queries scale depends very much on their nature. RGIS effectively leverages the existing mechanisms in the RDBMS to handle a mixed query/update workload.

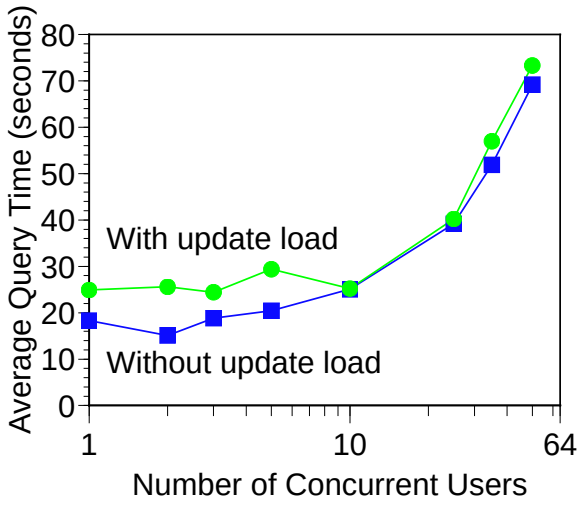


Figure 4.16: Cluster finder with multiple concurrent users and update load (scoped approximate).

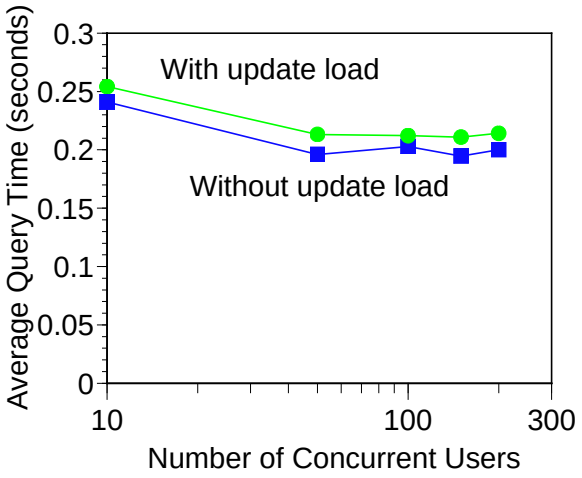


Figure 4.17: Host group query with multiple concurrent users and update load (scoped approximate).

4.4 Time-bounded queries

The evaluation of the previous section showed that our query techniques can trade off between the running time of the query and the number of results returned over many orders of magnitude. In particular, nondeterminism provides very fine grain control, the selection probability. However, as described in Section 4.2, queries in RGIS are also time-bounded. RGIS implements these deadlines using three techniques.

The first technique is *hard-limiting*. The query manager/rewriter starts the query as a child process or thread. The child is then allowed to run until the deadline is exceeded. If it completes before that time, it returns the result set to the parent which returns them to the caller. If it runs out of time, it is killed and no result set is returned. Hard-limiting can be used with every query in our system. Nondeterministic queries and randomly scoped queries are repeated during the allotted time.

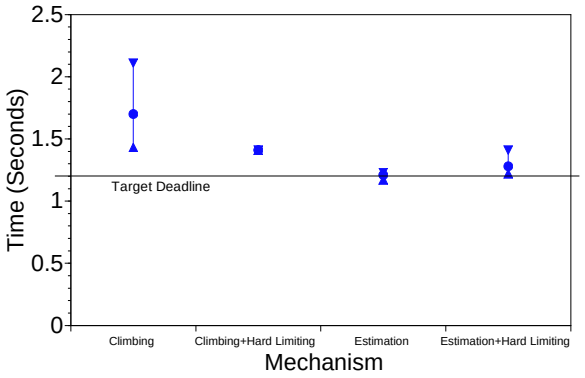
The second technique is *climbing*, which is limited to nondeterministic queries. Here, we initially run the query with a very small selection probability. If no results are returned, the probability is doubled and the query is run again. This happens iteratively until either the deadline is exceeded or a non-null result set is available. Notice that because climbing always issues another query if there is time left, it may overshoot the deadline.

The third technique is *estimation*, which is also limited to nondeterministic queries. Estimation is similar to climbing except that we predict the next query time from the previous query times and then only issue the next query if there is sufficient time remaining. Hence, it is far less likely to overshoot the deadline. Surprisingly, predicting query time from previous instances of a nondeterministic query run with lower selection probabilities appears to be straightforward.

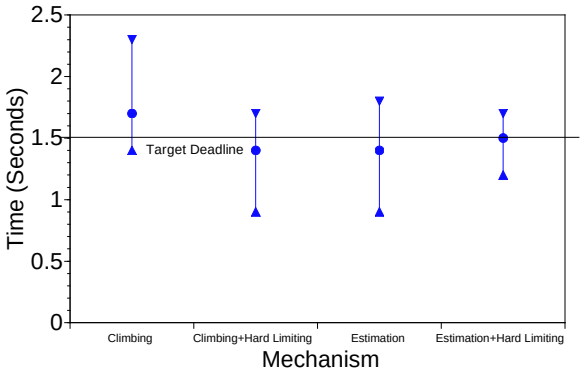
We studied several functions (linear, power, polynomial, exponential) for mapping from selection probability to running time. Degree two polynomials worked best for the queries described in the previous section. In our implementation, we monitor each query's time and selection probability. After the first query, we estimate the second query time to be the same as the first. After the first two queries, we do a degree one Lagrange interpolation to estimate the third query time. For the fourth and further queries, we estimate the next query time by applying a degree two Lagrange interpolation polynomial to the previous three query times. Hence, after the first query, we have some model that maps from selection probability to query time. We then use that model to predict if we still have enough time to do the next query or must terminate.

Because selection probability grows exponentially, climbing and estimation are largely insensitive to the initial selection probability and we set it very low.

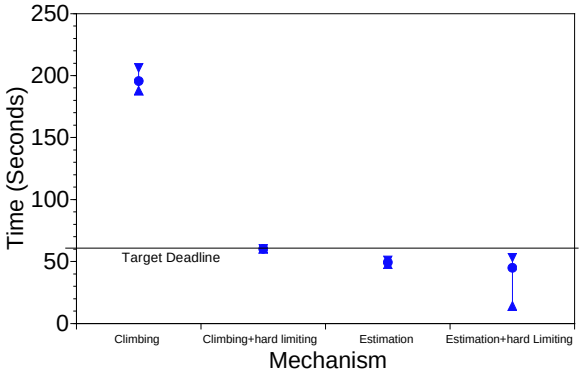
Figure 4.18 illustrates the performance of these different techniques for a sample nondeterministic query. The query looks for two hosts with a combined total of 600 GB of main memory in a 50,000 host database. Such a combination is very rare, but possible, hence the running time would be quite high for a deterministic version of the query. The only difference between (a), (b), and (c) is the deadline, 1.2, 1.5, and 60 seconds, respectively. Each query is run five times. The figure illustrates the average, minimum, and maximum running time. Clearly, it is possible to keep the running time close to the deadline using the three techniques. This is also the case for queries that are allowed to run longer, and for queries involving a larger number of hosts.



(a) 1.2 second deadline



(b) 1.5 second deadline



(c) 60 second deadline

Figure 4.18: Time-bounding nondeterministic queries.

4.5 Conclusions

In this chapter, we have described our SQL query rewritting techniques, inluding nondeterministic, scoping and approximation, three techniques to reduce the running time of queries in the RGIS system. Nondeterministic query conducts joins on randomly selected subset, scoping adds additional constraints to queries relative to the network topology and other relationships in the schema, while approximation replaces joins with tighter constraints on individual objects. These techniques are implemented using a combination of query rewriting, schema extensions, indices, and randomness. No changes to the RDBMS are needed. In all the three cases there is a tradeoff: a subset of the full result set is quickly returned. Response time and server load can be dramatically reduced.

We evaluated the performance of our implementation, populating our database with networks as large as five million hosts. The evaluation showed that a meaningful tradeoff between query processing time and result set size is possible using the three techniques, and that we can use that tradeoff to keep query running time largely independent of query complexity. We then discussed three techniques that we use to time-bound nondeterministic queries and evaluated their performance.

These techniques are complementary to each other. The three techniques help to make a relational approach to GIS feasible.

Chapter 5

Scheduling With Inaccurate Information

This chapter describes our research on the performance of size-based scheduling policies when only limited job size information is available. This is of important practical value for the RGIS system because the RGIS servers typically have to schedule queries using estimated job sizes and updates. Research in this chapter helps to answer questions like "can we use size-based scheduling on RGIS?" and "how to use size-based schedulers on RGIS?".

5.1 Introduction

In a queuing system, job requests continuously arrive to be serviced by one or several servers or stations. A request requires a certain service time to be completed. A request is queued when it arrives and remains in the system until it is complete, with the total time from arrival to completion being called the sojourn time or response time. Scheduling policies determine which requests in the queue are serviced at any

point in time, how much time is spent on each, and what happens when a new request arrives. Common goals of the scheduling policy are to minimize the mean sojourn time (response time of the request), the average slowdown (the ratio of its response time to its size), and to act fairly to all requests.

Many policies are widely used due to their simplicity or perceived fairness. First Come First Served (FCFS) is a non-preemptive policy in which the requests are run to completion in the order in which they were received. A more common policy is Processor Sharing (PS), which is preemptive. In PS all requests in the queue are given an equal share of the server's resources and all requests share roughly same slowdown, thus PS is considered to be fair. Generalized Processor Sharing (GPS) generalizes PS with priorities. Often, FCFS can be combined with PS or GPS, with FCFS dispatching of requests from the queue to a pool of processes or threads that are collectively scheduled using PS or GPS. These polices ignore the service time or job size of the requests.

Sized-based scheduling policies, such as Shortest Remaining Processing Time (SRPT) and the Fair Sojourn Protocol(FSP) incorporate the service time or the job size of the request into their scheduling decisions, and thus can achieve shorter mean response time than the scheduling policies that ignore the job size information, such as FCFS and PS.

The primary concern with SRPT is the fear that large jobs may starve under SRPT [212], that the average performance improvements of SRPT over other policies stem from SRPT unfairly penalizing large jobs in order to help small jobs. Recent research [33, 103] has shown that the performance gains of SRPT over PS in fact do not usually come at the expense of large jobs.

These results make size-based scheduling more practical. However, size-based scheduling policies require a priori knowledge of job sizes, which is not always avail-

Scheduling Policy	Description
PS	Processor Sharing scheduling policy.
SRPT	Ideal Shortest Remaining Processing Time scheduler, job sizes are known accurately a priori. The scheduler always choose the job with the shortest remaining size to serve first.
SRPT-E	Shortest Remaining Processing Time scheduler that uses estimated job sizes as scheduling information. The scheduler always chooses a job with the estimated shortest remaining size to serve first.
FSP	Ideal Fair Sojourn Protocol
FSP-E	Fair Sojourn Protocol that uses estimated job sizes as scheduling information.

Figure 5.1: Scheduling policies used in the chapter.

able. This is another reason for the lack of broad application of these policies [33]. All previous research work has targeted ideal size-based policies where the job sizes are assumed to be accurately known in advance. As a result, the behavior of size-based scheduling policies with inaccurate scheduling information is largely unknown. This dissertation is the first work to address the question.

The rest of this chapter is organized as follows. We talk about related work in Section 5.2. In Section 5.3, we describe our simulation setup and introduce a useful random number generator algorithm that allows us to control the correlation between two random number series. Next, we show simulation results on performance in Section 5.4, and fairness results in Section 5.5. We conclude with two new applications of size-based scheduling policies in Section 5.6 that would be made possible given reasonably accurate job size estimators.

5.2 Related work

SRPT has been studied since the 1960s. Schrage first derived the expression for the response time in an M/G/1 queue [197]. For a general queuing system (G/G/1) Schrage proved in 1968 that SRPT is optimal in the sense that it yields—compared to any other conceivable strategy—the smallest mean value of occupancy and therefore also of waiting and delay time [196]. Schassberger obtained the steady state appearance of the M/G/1 queue with SRPT policy in 1990. Perera studied the variance of delay time in $M/G/1/SRPT$ queuing systems and concluded that the variance is lower than FIFO and LIFO [177]. Bux introduced the SRPT principle into packet networks [48] in 1983.

Recently, SRPT [33, 199, 112, 103, 98] and FSP [98] have received much attention in the context of connection scheduling at web servers. Bansal, et al proved theoretically that the degree of unfairness under SRPT is surprisingly small assuming an M/G/1 queuing model and heavy-tailed job size distribution [33]. Gong, et al further investigated the fairness issues of SRPT through simulation [103] and confirmed the theoretical results regarding the asymptotic convergence of scheduling policies with respect to slowdown [113]. Harchol-Balter, et al prototyped SRPT scheduling on Apache web server and their evaluation showed the superiority of SRPT over PS [112] in terms of mean response time. To further improve the fairness of SRPT scheduling, Friedman, et al proposed Fair Sojourn Protocol (FSP) that combined SRPT with PS to trade off fairness with performance [98]. They concluded that FSP is both efficient in a strong sense (similar to SRPT), and fair, in the sense of guaranteeing that it weakly outperforms processor sharing (PS) for every job on any sample path.

All the previous research on size-based scheduling assumes accurate knowledge of

job sizes a priori, which is not obtainable in many cases. To the best of our knowledge, no work has been done to characterize size-based policies with inaccurate scheduling information. Furthermore, most theoretical work assumes the M/G/1 queuing model due to its analytical simplicity. However, many computer systems are better modeled with a G/G/n/m queuing model, where both job arrival and job size distribution are not Poisson, and there are n servers serving a queue with limited capacity m. A web server is an example. Previous research [175, 72] has shown that Poisson processes are valid only for modeling the arrival of user-initiated TCP sessions such as the arrival of TELNET connections and FTP connections. HTTP arrivals are not Poisson. Previous work [72] pointed out that the aggregated interarrival times of HTTP requests can be modeled with a heavy-tailed Weibull distribution.

There has been significant work on the G/G/n queuing model, and only the most closely related papers are listed here. Tabet-Aouel, et al gave analytic approximations for the mean sojourn time of P ($P \geq 2$) priority classes in a stable G/G/c/PR queue with general class interarrival and service time distributions and c ($c \geq 2$) parallel servers under pre-emptive resume (PR) scheduling [211]. Boxma, et al considered a G/G/1 queue in which the service time distribution and/or the interarrival time distribution has a heavy tail, i.e., a tail behavior like t^{-v} with $1 \leq v \leq 2$, such that the mean is finite but the variance is infinite. Depending on whether the service time distribution is heavier than that of the interarrival time distribution, they concluded that the stationary waiting time can be modeled as either a Kovalenko distribution or a negative exponential distribution [44]. However, we are unaware of any analytical results on G/G/n/m for SRPT or FSP scheduling in regimes where interarrival times and service times are heavy-tailed.

To characterize size-based policies with inaccurate scheduling information under more realistic queuing models such as G/G/n/m, we developed a simulator that

Queuing Model	Description
$M/G/1/m$	Poisson arrival process; General job size distribution (Pareto and Weibull); Single server ; Limited queue capacity m.
$G/G/n/m$	General arrival process (Pareto and Weibull); General job size distribution (Pareto); n servers ; Limited queue capacity m.

Figure 5.2: Queuing models studied in the chapter.

can support PS, FSP, and SRPT in both M/G/1/m and G/G/n/m. The simulator operates on a trace of request arrivals, which can come either from real world traces or from a trace generator. The trace contains the request arrivals, the actual job sizes, and the estimated job sizes. Our trace generator allows us to control the correlation coefficient R between actual job size and estimated job size in a trace. Using the simulator and the trace generator, we study the mean response time and slowdown for SRPT and FSP scheduling policies with estimated job size information. Our simulation experiments with generated traces show that the performance of size-based policies is strongly related to the degree of correlation (R) between estimated job size and actual job size. For low values of R, these scheduling policies perform *worse* than PS, but given a reasonably good job size estimator, SRPT and FSP can outperform PS in both mean response time and slowdown.

5.3 Simulation setup

In this section, we describe our performance metrics, simulator validation, synthetic trace generation and simulation parameters.

Throughout this chapter, we refer to the scheduling policies as listed in Figure 5.1, and the queuing models used as listed in Figure 5.2.

We set up the simulations driven by our synthetic traces to investigate how the degree of the correlation (R) between actual job size and estimated job size affects the performance of SRPT-E and FSP-E, where estimated job size is used as scheduling information, and compare them with a size-oblivious policy (PS), ideal SRPT, and FSP where actual job sizes are assumed to be known a priori.

Using the standard definition [197], we define the *load* on the queuing system as mean arrival rate divided by mean service rate throughout the rest of this chapter. Unless otherwise stated, we fixed the load to be 0.9, making the queuing system reasonably heavily loaded.

5.3.1 Performance metrics

Our performance metrics are the mean response time and slowdown.

- Mean response time: Response time refers to the time span between a job's arrival at and departure from the server. It is also known as sojourn time or turn around time. Mean response time has been used as a primary performance metric in queuing theory [197, 33, 103].

- Slowdown: Using the definition introduced by Bansal and Harchol-Balter [33], we define slowdown of a job as the ratio of its response time to its size (or service time). Slowdown is also referred to as normalized response time [33]. This metric is important because it reflects how long a job waits in the system relative to its size, thus helps to evaluate unfairness. Under a fair policy like PS, all jobs experience the same slowdown.

5.3.2 Simulator

Our simulator supports both M/G/1/m and G/G/n/m queuing systems. It is driven by a trace in which each request contains the arrival time, actual job size, and estimated job size. We use synthetic traces generated with interarrival times from exponential and bounded Pareto distributions, actual job sizes from bounded Pareto distributions, and estimated job sizes also from bounded Paretos. In the synthetic traces, we directly control the correlation, R, between actual size and estimated size, as described later. Throughout the rest of the chapter each simulation is repeated 20 times and we present the average.

Similar to Bansal and Harchol-Balter's work [33], we concentrate on M/G/1/m queuing model for the simulations presented in this chapter. Simulations with G/G/n/m queuing model show similar trends and the details can be found in our technical report [153]. Figure 5.4 shows the parameters of the bounded Pareto distributions used for the simulations shown in the rest of this chapter. We used identical bounded Pareto distributions for both estimated job size and actual job size distributions as shown in Figure 5.2.

We validated our simulator by:

- Assuring that Little's law is never violated on each run, using effective arrival rate as appropriate for limited queue capacity.
- Repeating the simulations described in Friedman and Henderson's FSP paper [98]. We got nearly identical results under FSP, SRPT, and PS policies.
- Comparing our simulation results with the analytic results of Bansal and Harchol-Balter's SRPT fairness paper [33]. Our simulation results are qualitatively consistent with theirs.

5.3.3 Controlling R in synthetic traces

Given some parametric distribution, e.g. exponential, and a target correlation coefficient R, we generate pairs of random numbers where each number of the pair is chosen from its required distribution and where the two numbers of the pair are correlated to degree R. To do this, we use a simplified Normal-To-Anything (NORTA) method. The basic ideas and proofs behind NORTA were developed by Cario and Nelson [52]. Given the distributions $dis_{estimatedsize}$ and $dis_{actualsize}$, our target correlation coefficient R and our sample size N, the following algorithm generates N pairs:

1. Set $\rho = R$
2. Generate two independent random numbers $x_1, x_2 \sim N(0,1)$.
3. let $y_1 = x_1, y_2 = \rho \times x_1 + \sqrt{(1-\rho^2)} \times x_2$
4. let $u_1 = NormCDF(y_1, 0, 1)$ and $u_2 = NormCDF(y_2, 0, 1)$, where $NormCDF(y_i, 0, 1)$ is the CDF value of a standard normal distribution at y_i for $i = 1, 2$. It can be shown that $u_i \sim U[0,1], i = 1, 2$
5. let $estimatedsize = F^{-1}_{dis_{estimatedsize}}(u_1)$, $actualsize = F^{-1}_{dis_{actualsize}}(u_2)$, where $F_{dis_{estimatedsize}}$, $F_{dis_{actualsize}}$ are the CDFs of our desired distributions for estimated size and actual size respectively. F^{-1}_{dis} is the inverse of F_{dis}.
6. Repeat steps 2-5 N times generating N pairs $\{(estimatedsize_j, actualsize_j)\}$. $\{estimatedsize_j, j = 1, \ldots, N\}$ and $\{actualsize_j, j = 1, \ldots, N\}$ are two correlated random numbers each following their own distributions.
7. Compute the correlation coefficient of $\{estimatedsize_j\}$, $\{actualsize_j\}$ and call

it ρ_{temp}. If $\rho_{temp} > R$, then decrease ρ and go to step 2. If $\rho_{temp} < R$, then increase ρ and go to step 2. If $\rho_{temp} \approx R$ then stop.

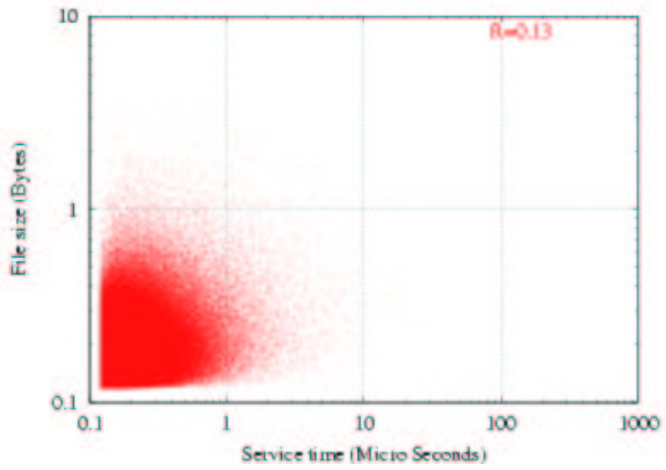
Figure 5.3 gives some examples of estimated size/actual size pairs generated for different values of R.

To show the correctness of this algorithm, we can try following analysis: First, it is easy to see that $y_1, y_2 \sim N(0,1)$ and $u_1, u_2 \sim U[0,1]$, thus $estimatedsize \sim dis_{estimatedsize}$ and $actualsize \sim dis_{actualsize}$. Second, it can be shown that y_1 is correlated with y_2 and thus so is u_1 with u_2. Intuitively it follows that $\{estimatedsize_j\}$ and $\{actualsize_j\}$ are correlated as well. Cario and Nelson showed that (1) ρ_{temp} is a nondecreasing continuous function of ρ, and (2) ρ_{temp} and ρ share the same sign. These properties guarantee the termination of the above simplified NORTA algorithm and let us bound the values of R that can be achieved by NORTA. If we sample ρ from 0 to 1, we can estimate the range of ρ_{temp}, producing a set of sets of pairs, ordered with increasing R as a side effect. This is exactly how we generated correlated random pairs of file size and service time. Depending on the structures of different distributions, ρ_{temp} may not always take a full range of $[0,1]$, which is why some of the results we show here have a restricted range of R.

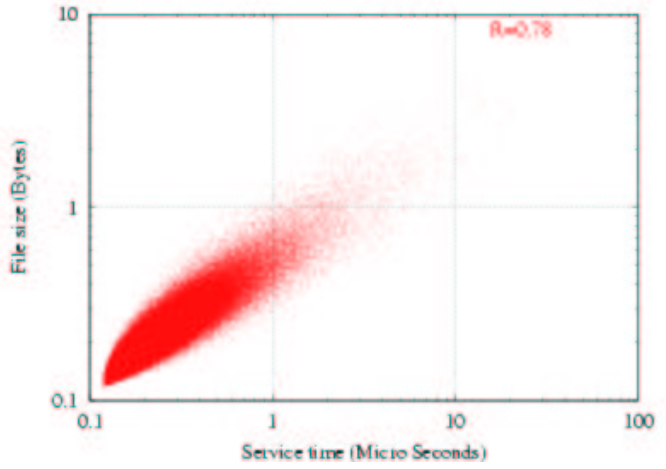
In Section 5.4, we show the simulation results on mean response time. We study slowdown as a function of job size and correlation coefficient R in Section 5.5.

5.4 Simulation results on mean response time

To study the effects of the correlation R between actual job size and estimated job size on the performance of SRPT-E and FSP-E, we generated traces with controlled correlation as described in the previous section. We used bounded Pareto distributions for both actual job size and estimated job size. For the arrival process, we consider



(a) R=0.13



(b) R=0.78

Figure 5.3: Examples of generated estimated size/actual size pairs.

	α	Lower bound	Upper bound
Job size	2.0	0.1188	10^5

Figure 5.4: Parameters for Bounded Pareto Distribution.

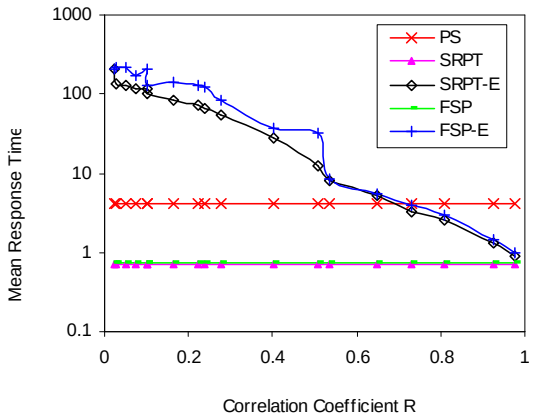


Figure 5.5: Mean sojourn time versus R, synthetic traces, M/G/1/m, Pareto service times, Poisson arrivals.

Poisson arrivals (exponential interarrival times), heavy tailed Pareto arrivals, and heavy tailed Weibull arrivals. For all the simulations of this section, the load (mean arrival rate divided by mean service rate) is 0.9, and the queue capacity is 5000. A single server is assumed. Multiple servers are similar to the single server case, hence the results are not shown here.

The scheduling policies used (SRPT, SRPT-E, FSP, FSP-E and PS) are described in Figure 5.1. Each graph data point represents the average of 20 simulations, each of which has processed 0.5 million job requests.

Figure 5.5 shows the effects of R on the mean response time of different scheduling polices with a Poisson arrival process, corresponding to the M/G/1/m queuing model. The interarrival mean used to generate Poisson process is 0.264. Note that the Y axis is in log scale. Heavy-tailed Pareto and Weibull arrival process with the same

job size distributions, corresponding to the G/G/1/m queuing model, show similar results and thus are omitted here.

As shown in Figure 5.5, SRPT-E results in lower mean response time than FSP-E does in most cases. The performance of SRPT-E and FSP-E increases quickly with increasing R. When R is very small, SRPT-E and FSP-E essentially behaves like a random scheduling policy, and it is worse than PS in mean response time. When R exceeds a threshold, SRPT-E and FSP-E performance exceed that of PS in both M/G/1/m and G/G/1/m. The threshold is about 0.7 in our simulations. We believe this threshold is a function of R and both distributions of job size and estimated job size. Beyond this point, SRPT-E and FSP-E's performance increases quickly with increasing R. The figure clearly shows that SRPT performance is strongly tied to R, even at high values of R. Improvements in estimating actual job size can dramatically improve SRPT and FSP for a wide range of R.

The lack of accurate job size information has been an important reason why SRPT is not widely deployed [33, 198]. Our simulations show that a reasonably good job size estimator is required for SRPT-E and FSP-E to outperform PS in terms of mean response time.

5.5 Simulation results on slowdown

Fairness is one major concern when applying size-based scheduling policies such as SRPT in practice. Theoretical [33] and simulation [103] work has shown that the degree of unfairness under SRPT is very small. However, no work has been done to study the fairness issue when the scheduler doesn't have accurate job size information. Like previous work [33, 103, 98], we use slowdown as fairness metric.

We evaluate the slowdown of SRPT-E and FSP-E as a function of job size and

the correlation coefficient R. Figures 5.6 through 5.11 show the slowdown versus the percentile of the job size distribution, with R increasing from 0.0224 to 0.9778. Note that the Y axis is in log scale in all these figures. The job sizes are categorized into 100 bins with each bin containing one percentile of the job size distribution. Again the load of system is 0.9.

From Figures 5.6 and 5.7, we can clearly see that both SRPT-E and FSP-E perform very poorly compared to PS when correlation is weak. This comes as no surprise because poor estimation of job sizes would render these policies almost equivalent to random scheduling. Much longer delays are imposed on jobs across the board. However, as the estimates improve, i.e. the increase of R values, the SRPT-E curve moves downward. It begins to outperform PS at $R = 0.4022$ for small jobs. For SRPT-E at the level of $R = 0.5366$, jobs below the 30 percentile have lower slowdown than with PS. For FSP-E, at the level of $R = 0.5366$, the slowdown is close to that caused by PS.

When R increases to 0.7322, both SRPT-E and FSP-E perform better than PS in general. For SRPT-E, close to 93% of the jobs have slowdown smaller than that of PS, while for FSP-E, it appears that all jobs have lower slowdown than that of PS. When estimated size is highly correlated with actual job size, SPRT-E and FSP-E's performance closely resembles that of ideal SRPT and FSP (Figure 5.11). Note that in most cases, the performance of ideal SRPT and FSP is very close and not clearly distinguished in the figures.

Based on the simulation results in Section 5.4 and this section, it is clear that both the mean response time and slowdown of size-based policies heavily depend on the correlation between actual job sizes and estimated job sizes. Although SRPT-E has a lower mean response time than FSP-E, it causes larger slowdowns when the correlation coefficient is larger than about 0.7.

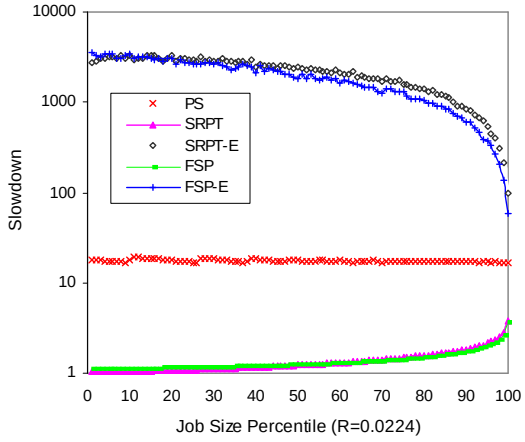


Figure 5.6: Slowdown as a function of the percentile of the job size distribution. Synthetic traces, R = 0.0224, M/G/1/m, Pareto service times, Poisson arrivals.

The simulations have clearly shown that high correlation between actual job sizes and estimated job sizes not only helps SRPT-E and FSP-E to reduce the mean response time, but also helps to achieve smaller slowdowns across various job sizes. More important than an accurate prediction for each job size is the order of the jobs in the queue—they need only be ordered by their size. A reasonably good estimator will enable SRPT-E and FSP-E to outperform PS in both mean response time and slowdown.

5.6 New applications

In this section, we describe three applications where the size-based policies could be successfully applied with effective job size estimators. We also show that effective job size estimators can be achieved at low overhead.

Appendix A describes our work on Domain-based scheduling on web servers. Appendix B describes our work on P2P server side scheduling.

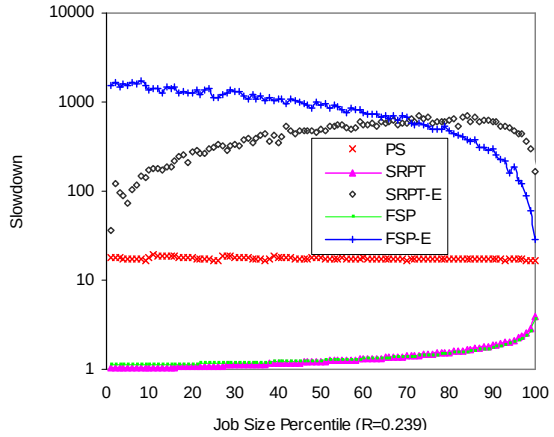


Figure 5.7: Slowdown as a function of the percentile of the job size distribution. Synthetic traces, R = 0.239, M/G/1/m, Pareto service times, Poisson arrivals.

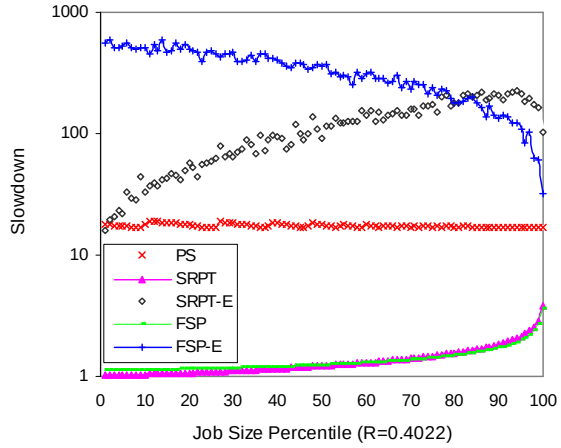


Figure 5.8: Slowdown as a function of the percentile of the job size distribution. Synthetic traces, R = 0.4022, M/G/1/m, Pareto service times, Poisson arrivals.

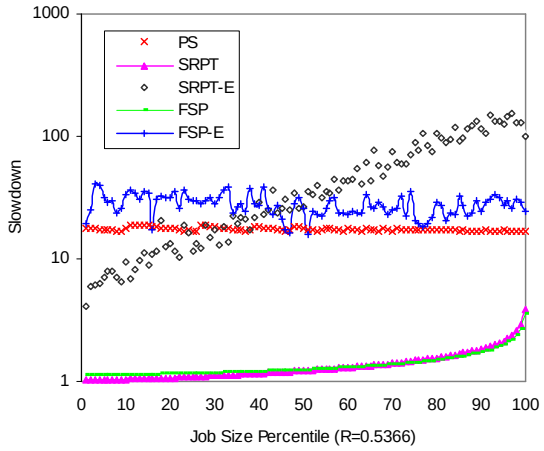


Figure 5.9: Slowdown as a function of the percentile of the job size distribution. Synthetic traces, R = 0.5366, M/G/1/m, Pareto service times, Poisson arrivals.

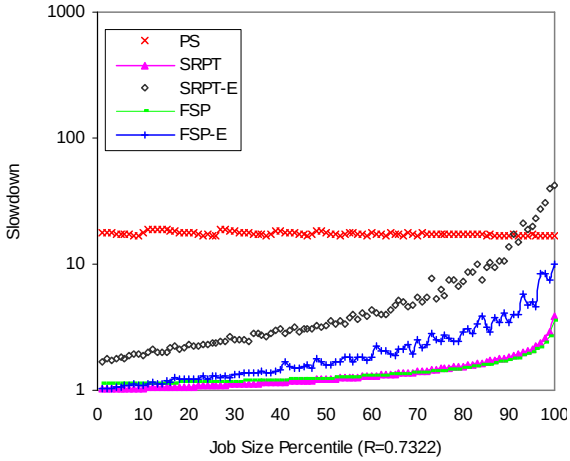


Figure 5.10: Slowdown as a function of the percentile of the job size distribution. Synthetic traces, R = 0.7322, M/G/1/m, Pareto service times, Poisson arrivals.

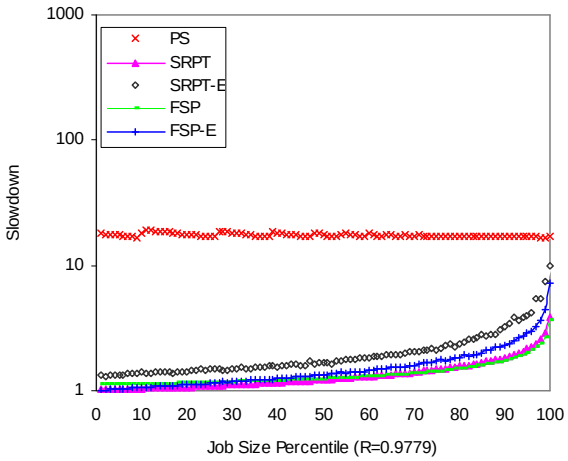


Figure 5.11: Slowdown as a function of the percentile of the job size distribution. Synthetic traces, R = 0.9779, M/G/1/m, Pareto service times, Poisson arrivals.

5.6.1 Domain-based scheduling on web servers

Previous research has tried to apply SRPT scheduling on web servers [111, 112] on the assumption that the service time is the size of the file being served, as this is very easy to discover when the request enters the system. More broadly, the assumption is that the service time is strongly correlated to the file size. In our own work [153], we found that this assumption is not accurate, and consequently developed Domain-Based scheduling to effectively apply size-based scheduling on web servers.

The idea of domain-based scheduling is to use web log to classify the Internet into domains, and use the web log to estimate throughput from different domains to the web server. Our assumption is that hosts within a same domain have similar throughput to the remote web server. The domain in our definiton is the high order k bits of the client IP address, which is based on the Classless Inter Domain Routing (CIDR) [114]. Our study [153] shows that this job size estimator is very effective with a low overhead.

5.6.2 P2P server side scheduling

Peer-to-Peer systems have grown significantly in popularity over the last few years. In the context of peer-to-peer file sharing, research efforts have focused on routing, search, incentives, and a few other topics. But no one has looked at the server side scheduling problem except our own ongoing work [181].

The server side of current file sharing P2P applications such as Kazaa [9] is similar to a web server in that they both accept requests for files and send back the requested files. But there are significant differences between them from a scheduler's point of view. Requests to P2P nodes are typically a small chunk of the complete file, and the amount of data actually served is often a fraction of the request size. Therefore, the job sizes cannot be known a priori for the P2P application. Furthermore, while web servers can reasonably assume full control over resources, P2P applications are commonly configured with quite conservative upper bounds for each thread's resource consumption to minimize their impact on other applications.

To apply the SRPT-E or FSP-E scheduling policies in current P2P file sharing applications, we have to use estimated job sizes as scheduling information. Our work [181] shows that by using the requested data chunk size as the scheduling metric, we can get a mean response time that is only 30% of that of FCFS and 50% of that of PS. We are currently working on better job size estimators to further enhance the performance of the P2P applications.

5.6.3 Network backup system scheduling

Backups protect file systems form user errors, disk or other hardware failures, software errors that may corrupt the file system. The most common uses of backups are to restore files accidentally deleted by users and to recover from disk failures. As

more and more data needs reliable and efficient backup, the backup techniques are becoming increasingly important.

Chervenak, et al [59] has shown that as the capacities of new storage devices continue to increase at a rate that is much faster than the speed of disk and type access, it will take increasing long to read the contents of a disk drive and write them to a backup device. Therefore, it is very important to improve the efficiency of backup systems.

Most current backup systems such as Amanda [69, 68] allow concurrent backups in a networked computing environment, where multiple file systems are backed up in parallel to one or more backup systems. A network backup system is similar to a web server in that they both transfer a large number of files between multiple machines via network. However, unlike a web server, a network backup system doesn't know the job sizes a priori because of the use of the incremental backup scheme, which copies only those files that have been created or modified since a previous backup. Furthermore, incremental backup schemes that compute and store file differences are the extreme case, where job size is not known until after completion.

If SRPT or FSP scheduling can be applied on network backup systems, the mean response time could be lowered. However, because the job sizes cannot be obtained in advance, estimated job sizes have to be used for scheduling. We speculate that history-based time series predictors can be applied to estimate the backup size for each machine, and job priority can be assigned accordingly. We are currently studying this possibility.

5.7 Conclusions and future work

Through simulations, we have evaluated the performance of size-based scheduling policies (SRPT and FSP, with PS for comparison), as a function of the correlation between actual job size and estimated job size. We found that SRPT and FSP's performance strongly depends on the correlation. When provided with weak correlation, SRPT and FSP can actually perform worse than PS, but given a reasonably good job size estimator, they can outperform PS in both mean response time and the slowdown. We also described three new applications of SRPT and FSP supported by our job size estimators.

To generate correlated trace data for the simulation, we introduced a new random number pair generation technique, where each number of the pair is chosen from its required distribution and they are correlated to degree R. This technique can be very useful for simulation related research in this and other areas.

To the best of our knowledge, this work is the first to address the performance of size-based policies with inaccurate scheduling information. However, we believe that we have by no means completely addressed this area. More simulations and theoretical work are necessary for a better understanding. We are very interested in analytical results for the impact of inaccurate job size information, and simulation results for a wider range of workload characteristics.

To effectively apply size-based scheduling on the RGIS server, we need to have an effective job size estimator. Our previous work has shown that such effective estimators can be built with the support of extensive traces analysis and characterization. As grids evolve into production stage and more performance traces become available, we can develop new job size estimator based on the characterizations of the queries. Then we can apply size-based scheduling on RGIS servers.

Chapter 6

Predicting TCP Throughput

RGIS servers rely on a Content Distribution Network (CDN) to propogate their updates to other peers. The CDN can use several mechanisms to do its job, for example, Unicast, Multicast or the Publish/Subscribe model. This dissertation discusses the unicast and multicast mechanism in chapter 6, 7 and 8.

This chapter describes DualPats, our TCP throughput monitoring and prediction framework, which can be used by the CDN to soft time bound the updates propagation.

6.1 Introduction

Application developers often pose an age-old question: "what is the *TCP throughput* of this path?" The question is more subtle than it appears. This chapter motivates, describes, and evaluates DualPats, an algorithm and system for answering it. We define TCP throughput as $\frac{D}{T}$ where D is the flow size and T is the flow duration, starting at connection establishment and ending at teardown. For a file transfer [21, 234], D is the file size.

The *available bandwidth* of a path—the maximum rate of a new flow that will not reduce the rate of existing flows [117, 122]—has been thoroughly investigated. Keshav's packet pair [129], Crovella's cprobe [53], IGI [117], and spruce [208] attempt to measure the available bandwidth accurately, quickly, and non-intrusively. Other tools, such as nettimer [134], pathchar and pchar [85], pathload [122], NCS and pipechar [127], pathrate [84] and delphi [190] measure either the bottleneck link capacity or the available bandwidth.

The available bandwidth is different from the TCP throughput that an application can achieve, and that difference can be significant. For example, Jain's pathload paper [122] showed the bulk transfer capacity [158] of a path is higher than the measured available bandwidth, while Lai's Nettimer paper [134] showed many cases where the TCP throughput is much lower than their measurement. Jin, et al confirmed that available bandwidth is not an indicator of what an application can actually obtain [126]. Tools for estimating available bandwidth have a further issue: their slow convergence, on the order of at at least 10s of seconds, makes them too slow for many applications.

The most widely used real time TCP throughput prediction framework is the Network Weather Service [230] (NWS). NWS applies benchmarking techniques and time series models to measure TCP throughput and provide predictions to applications. NWS is widely used in grid computing and other application contexts.

Unfortunately, recent work [222, 221] has shown that NWS, and by implication, current TCP benchmarking techniques in general, have difficulty predicting the throughput of large file transfers on the high speed Internet. Sudharshan, et al [222] showed that NWS was predicting less than 1/10 of the actual TCP throughput achieved by GridFTP. In response, they proposed using a log of large file transfers to predict future file transfers. A key problem with this idea is that the log is up-

dated only at application-chosen times, and thus changes in TCP throughput are only noticed after the application uses the path.

Taking dynamic changes into consideration, Sudharshan, et al [221] and Swany, et al [209] separately proposed regression and CDF-matching techniques to combine the log-based predictor with small NWS probes, using the probes to estimate the current load on the path and adjust the log-based predictor accordingly. These techniques enhanced the accuracy of log based predictors. However, they remain limited to those host pairs that have logs of past transfers between them, and the logs must still be kept fresh. Zhang, et al [238] showed it is misleading to use history older than one hour. Furthermore, due to the strong correlation between TCP flow size and throughput [239], a log for one TCP flow size is not directly useful for predicting throughput for another.

The passive measurement approach [42, 127, 236] avoids the overhead of active probes by observing existing network traffic. Unfortunately, similar to the log-based prediction techniques, the passive approach is limited to periods of time when there is traffic on the network between the hosts of interest. Also, these systems measure available bandwidth, not TCP throughput.

There is an extensive literature on analytic TCP throughput models [160, 169, 26]. However, these models are limited in practice due to the difficulty in obtaining accurate model parameters such as TCP loss rate and RTT. Goyal et al [105] concluded that it is not easy to obtain accurate estimates of network loss rates as observed by TCP flows using probing methods, and that polling SNMP MIBs on the routers can do much better. However, because these statistics are aggregated and it is well known that TCP has a bias against connections with high RTT [183], this approach is limited to paths where the bottleneck router provides common loss rates, such as with RED. Furthermore, this approach has to determine the bottleneck router on the

end-to-end path (a difficult problem) and have SNMP access to it (rarely available today).

DualPats follows from these questions, which we address via a large-scale measurement study:

- How can we explain the strong correlation between TCP flow size and throughput, and what are its implications for predicting TCP throughput?
- How can we characterize the statistical stability of the Internet and TCP throughput, and what are its implications for predicting TCP throughput?
- How can we predict the TCP throughput with different TCP flow sizes without being intrusive?

The main contributions of this chapter are:

- Additional causes for the observed strong correlation between TCP flow size and throughput [238],
- A characterization of TCP throughput stability and statistics,
- A novel yet simple TCP benchmark mechanism,
- A dynamic sampling rate adjustment algorithm to lower active probing overhead, and
- DualPats and its evaluation.

6.2 Experimental Setup

Name	Statistics	Main Purpose	Hosts, Paths, Repetitions	Messages, Software, procedure
Distribution Set	1,620,000 TCP transfers	To evaluate TCP throughput stability and transient distributions	40 PlanetLab nodes in North America[a], Europe, Asia, and Australia. Repeat random pairing 3 times, 60 distinctive paths total	Client/Server: 100 KB, 200 KB, 400 KB , 600 KB, 800 KB, 1 MB, 2 MB, 4 MB, 10 MB. Server sends data with specific size to client continuously for 3,000 times and then start to sendata of different size.
Correlation Set	2,430,000 TCP transfers; 270,000 runs	To study correlation between TCP throughput and flow size, and evaluate proposed TCP benchmark mechanism.	40 PlanetLab nodes in North America[a], Europe, Asia, and Australia. Repeat random pairing 3 times, 60 distinctive paths total	Client/Server: 100 KB, 200 KB, 400 KB , 600 KB, 800 KB, 1 MB, 2 MB, 4 MB, 10 MB. Server sends a sequence of data with increasing sizes in order, start over after each run.
Verification Set	4,800 TCP transfers; 300 runs	To test proposed TCP throughput benchmark mechanism; To strengthen Distribution Set and Correlation Set with large TCP flow sizes and different applications	20 PlanetLab nodes in North America[a], Europe, Asia, and Australia, one node at Northwestern, one node at ANL, 30 distinctive paths total	GridFTP, scp: 5 KB to 1GB. Server sends a sequence of files with increasing sizes in order, start over after each run.
Online Evaluation Set	14000 test cases	To evaluate the DualPats TCP throughput prediction service.	50 PlanetLab nodes in North America[a], Europe, Asia, and Australia, random paring 2 times, 50 distinctive paths total	GridFTP, scp: randomly send a file of size 8MB or 40 MB or 160 MB. About 10 days long

Figure 6.1: Summary of experiments.

Our experimental testbed includes PlanetLab and several additional machines located at Northwestern University and Argonne National Laboratory (ANL). PlanetLab [6] is an open platform for developing, deploying, and accessing planetary-scale services. It currently consists of more than 400 computers located at about 200 sites around the world.

We conducted four sets of experiments: Distribution Set, Correlation Set, Verification Set, and Online Evaluation Set. Each experiment set involves PlanetLab nodes from North America, Europe, Asia, and Australia. We set the TCP buffer size in a range from 1 to 3 MegaBytes in all the experiments with GridFTP, while we used default TCP buffer in other experiments. In the 2.4 Linux kernel, which we use, the socket buffer size is automatically tuned to approximately twice the estimated bandwidth delay product. More details of the experimental setup can be found elsewhere [149].

For each of Distribution Set and Correlation Set, we chose 40 nodes on PlanetLab spread across North America, Europe, Asia, and Australia. We randomly grouped those nodes into 20 pairs, each containing a client node and a server node and the Internet path between the two. A server listens and accepts incoming TCP connection requests from its client counterpart and transfers data of a particular size to the client through the established TCP connection. The client repeatedly connects to its server, requests some data, records the transfer time, and then closes the connection. To evaluate more Internet paths, we randomly changed pairings 3 times, resulting in 60 different paths for Distribution Set and another 60 for Correlation Set.

Distribution Set serves as a basis for evaluating TCP throughput stability and distributions. Since here we want to see how TCP throughput with a specific flow size varies with time and its distribution within each stable epoch, we transfer data of a particular size between pairs of clients and servers continuously for at least 3,000

times, then move on to perform the same operation on another data transfer with a different size until we finish the set of flow sizes as listed in the Distribution Set entry in Figure 6.1. The trace data used in Distribution Set is mainly used in the discussion of TCP throughput stability and distributions in Section 6.4.

Correlation Set serves to study the strong correlation between TCP flow size and throughput, and to verify our TCP throughput benchmark mechanism, as discussed in Section 6.3. We define a *run* in Correlation Set and Verification Set as a procedure conducting a sequence of TCP transfers with increasing flow sizes between two hosts. For example, in Correlation Set, the client first requests 100 KB data, followed by a 200 KB request, then 400 KB, etc, up to 10 MB; this sequence forms a run. This lets us evaluate our TCP benchmark mechanism by predicting the transfer time of larger TCP transfers based on the transfer time of two smaller TCP flows and then comparing the predicted time with the actual transfer time of the larger transfers in the run. To guarantee fair evaluation, runs were repeated approximately 4,500 times between each pair of nodes, yielding the same number of TCP throughput predictions scattered at different times during our experiments. In total we have $\sim 270,000$ runs on ~ 60 paths.

Verification Set was done to further verify the correctness of our proposed TCP benchmark mechanism, and to strengthen analysis based on Distribution Set and Correlation Set with larger TCP flow sizes. Verification Set was conducted on twenty PlanetLab nodes, one node on Northwestern University campus and one node at ANL. We used GridFTP and scp in this set because both applications require authentication before transferring effective data. Our script transfered a series of files ranging from 5 KBytes to 1 GBytes in sequence and recorded each transfer time as the flow duration.

Online Evaluation Set serves to evaluate our DualPats real time TCP throughput

prediction framework. We randomly choose fifty PlanetLab nodes, and do random pairing twice, resulting in 50 distinctive paths. We use DualPats to monitor the 50 paths for a duration of about 10 days. During the experiment we randomly send a file of size 8MB or 40MB or 160MB using scp as a test case to compare with the prediction result. Online Evaluation Set contains 14000 predictions.

6.3 Exploiting size / throughput correlation

A surprising finding in recent TCP connection characterization is that TCP flow size and throughput are strongly correlated. This section explains the phenomenon, provides new additional explanations for it, explains why it can lead to inaccurate TCP throughput predictions, and outlines a new prediction approach.

6.3.1 Phenomenon

Zhang, et al [239] analyzed the correlations between the TCP flow characteristics of interest, including flow duration and throughput, flow duration and size, and flow size and throughput. They pointed out that these correlations are fairly consistent across all their traces, and show a slight negative correlation between duration and throughput, a slight positive correlation between size and duration, and a strong correlation between throughput and flow size. They argue that the strong correlation between flow size and throughput is the most interesting one and explained it in the following ways.

Slow start: TCP slow start could cause some correlation between flow size and flow rate [239]. The distribution of TCP flow sizes follows a power law, which, in part, tells us that the majority of flows are short. Balakrishnan, et al [27] showed that 85% of the web-related TCP packets were transfered during slow start. This implies

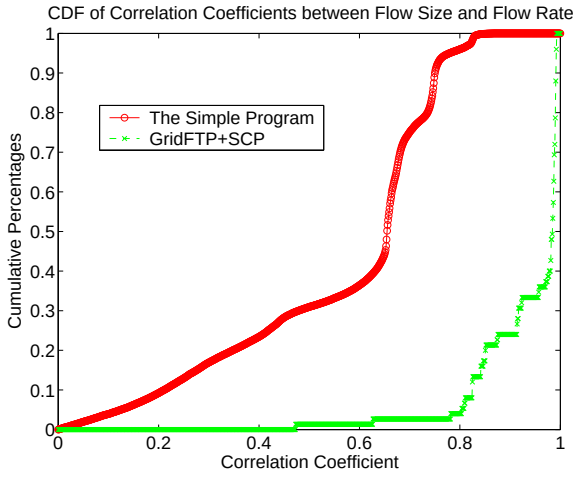


Figure 6.2: CDF of correlation coefficients R between flow sizes and throughput in experiments Correlation Set and Verification Set.

that most web-related flows ended in slow start, before TCP had fully opened its congestion window, leading to throughput much lower than would be possible with a fully open window. However, after eliminating the first one second of all the flows, they found that the strong correlation between flow size and throughput remained strong.

User effect: The users are estimating the underlying bandwidth, and thus transferring big files only when the estimated bandwidth is correspondingly large [239].

These are two valid reasons, but they may be insufficient. We claim that most users do not estimate the available bandwidth before transferring data. Furthermore, that the correlation persists even when initial slow start is removed suggests that there must be some other mechanisms at work.

Let's consider the correlation between flow size and throughput in our experiments. Figure 6.2 gives the cumulative distribution functions (CDFs) of the correlation coefficient (Pearson's R) [1], where each individual R value is calculated from

[1]Both Pearson's Correlation Coefficient R and Coefficient of Determination R^2 are used in the

one run of Correlation Set or Verification Set. The correlation between flow size and transfer time is large for the majority of transfers using our simple test program, and even larger for GridFTP and scp transfers. For our simple TCP test program in Correlation Set, over 80% of all runs demonstrate strong or medium Rs between flow sizes and flow rates. Further, 64% of all runs have $R > 0.6$. For the GridFTP and scp results in Verification Set: $> 98\%$ of the runs shows strong correlation, $> 95\%$ show $R > 0.8$.

6.3.2 Further explanations

Now we consider additional explanations for the surprising correlation between flow size and transfer time.

Non-negligible startup overheads: Most applications have an initial message exchange. For example, GridFTP and scp require certificate or public key authentication before starting to send or receive data.

Figure 6.3 shows the TCP throughput as a function of TCP flow size, for transfers using GridFTP between Northwestern university and ANL. The dotted line is the asymptotic TCP throughput. We tried linear, logarithmic, order 2 polynomial, power, and exponential curve fitting, but none of them fit well.

We next considered the relationship between TCP flow duration (transfer time) and flow size (file size). Figure 6.4 shows that this relationship can be well modeled with a simple linear model with R^2 close to 1. The majority of the data-points missed by the linear model are located at the very beginning of the curve, which we refer to

analysis of the paper. R^2 represents the percent of the variation that can be explained by the regression equation, therefore we use it to show how good a curve fitting is. R is widely used to measure the strength of a (linear) relationship, therefore we use R to show how strong two random variables are linearly correlated.

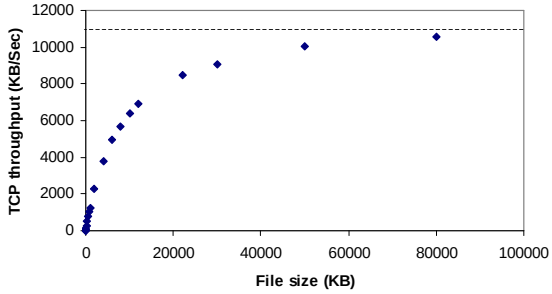


Figure 6.3: TCP throughput versus flow size (file size) with GridFTP. Transfers are between Northwestern University and Argonne National Lab. Single TCP flow with TCP buffer set. We made similar observations on all the other paths we studied.

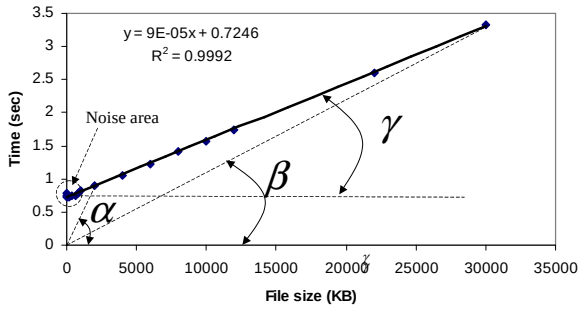


Figure 6.4: Transfer time versus TCP flow size with GridFTP. Transfers are between Northwestern University and Argonne National Lab. Single TCP flow with TCP buffer set. We made similar observations on all the other paths we studied.

as the *noise area* in the figure. The noise area is due to startup costs and the residual slow start effect, described below. The linear model may not hold in the noise area.

A closer look at Figure 6.4 shows that the total TCP flow duration or file transfer time can be divided into two parts: the startup overhead and the effective data transfer time. We represent this as

$$T = A \times x + B \tag{6.1}$$

where T is the TCP flow duration, including both startup overhead and data transfer time, x is the TCP flow size or file size, and B is the startup overhead, which includes authentication time and the residual slow start effect as described below. $\frac{1}{A}$ is the steady state asymptotic TCP throughput in Figure 6.3.

Given Equation 6.1, we can easily deduce the expression for the TCP throughput in Figure 6.3 as

$$TP = \frac{x}{T} = \frac{x}{A \times x + B} \tag{6.2}$$

where TP is the TCP throughput, and x, A, B are the same as in Equation 6.1.

Residual slow start effect: Mathis, et al [160] pointed out that it takes TCP some time before its throughput reaches equilibrium. Assuming selective acknowledgments (SACK), TCP will send roughly $\frac{1}{p} \times log_2 \frac{1}{C\sqrt{p}}$ packets in the unstable phase, where p is the loss rate and C is a constant $\approx \sqrt{3/2}$. This number can be significant given a low loss rate p. This happens because with SACK, slow start will overshoot and drive up the loss rate or run out of receiver window. Zhang, et al [238] showed that the mean loss rate in their traces is between 0.006 and 0.0087. Assuming the loss rate is 0.006, and each packet is 1.5 KB, roughly 800KB data has to be sent before TCP throughput reaches equilibrium.

We examined hundreds of Linux machines on the Northwestern University campus

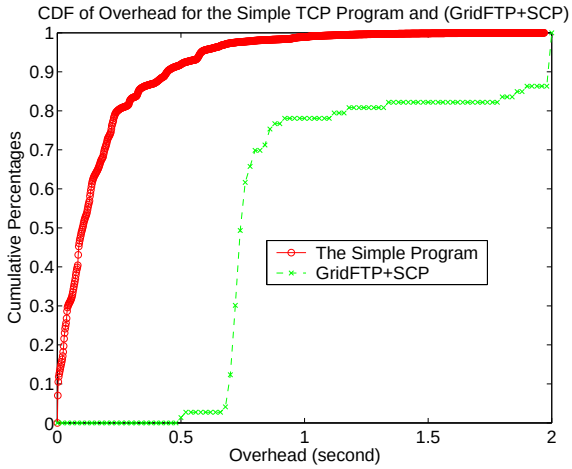


Figure 6.5: CDF of B, the startup overhead. Even for the simple client/server there is startup overhead likely caused by the residual slow start effect. The startup overheads of scp and GridFTP are much larger.

and on PlanetLab and found that all of them were using SACK. Therefore, it is likely that most TCP connections experience this slow start overshoot effect, and because TCP in slow start doesn't use bandwidth well, this residual slow start effect can be treated as another kind of startup overhead, incorporated in B as above. This can also explain why in Figure 6.2 the Rs for the scp and GridFTP traces are much stronger than that of the simple program.

To verify that this is the case in general for the simple applications without other startup overheads, we used the data collected in Correlation Set. We did least square linear curve fitting and calculated B for each set of data. Figure 6.5 shows the CDF for these Bs. The effect of residual slow start is obvious in the CDF, where we see over 50% simple TCP transfers has a B value equal or larger than 0.1. For comparison purpose, we also plot the CDF of B for applications that require authentication in the same Figure, namely GridFTP and SCP. As the CDF indicates, a typical B for such applications is much larger than that of the simple application.

6.3.3 Why simple TCP benchmarking fails

Now we can explain why current TCP benchmarking approaches, such as implemented in NWS, have difficulty predicting the performance of large transfers such as GridFTP tests [221]:

- The default probe used by NWS is too small. It will likely end up in the noise area as shown in Figure 6.4.
- The TCP throughput that the probe measures is only useful to TCP flows of similar size because of the strong correlation between throughput and flow size.

  Given Equation 6.2, it is clear that $cotangent(\alpha)$ is the TCP throughput for the flow size 2000KB, $cotangent(\beta)$ is the TCP throughput for the flow size 30000KB and $cotangent(\gamma)$ is the steady state TCP throughput. As file size increases α decreases, and when the file size is approaching infinity, the throughput will approach $cotangent(\gamma)$.
- The TCP buffer is not set for NWS probes while the GridFTP tests were done with adjusted buffer sizes.
- The usage of parallel TCP flows in GridFTP increases its aggregated throughput.

To verify that the linear model is valid for most Internet paths, Figure 6.6 shows the R^2 of the linear curve fitting for the data in Correlation Set and Verification Set. It is clear the model holds for both our simple client and server, and applications such as scp and GridFTP that require authentication.

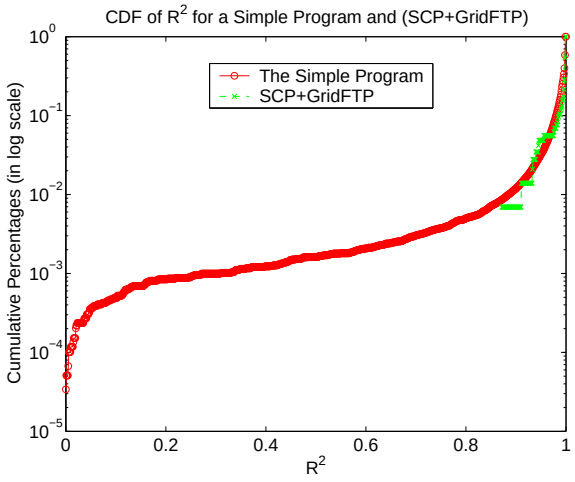


Figure 6.6: CDF of R^2 for linear model of Figure 6.4. Each R^2 is from a independent test. Both simple client/server and applications that require authentication show a strong linear property. Note that the Y axis is in log scale to show detail. Over 99% of the runs had $R^2 > 0.95$.

6.3.4 A new TCP throughput benchmark mechanism

Based on the above observations, we developed a new simple TCP benchmark mechanism. Instead of using probes with the same size, we use two probes with different sizes, chosen to be beyond the noise area. We then fit a line between the two measurements, as shown in Figure 6.4. Using Equations 6.1 and 6.2, we can then calculate the TCP throughput for other flow sizes (file sizes).

To verify that the new technique works, we used the trace data in Correlation Set. We chose a small probe with size 400KB and a bigger probe with size 800KB, and predicted the throughput of the other TCP transfers in the trace. Figure 6.7 shows the CDF of relative prediction error for our results by flow size. $> 80\%$ of the prediction errors are below 20%.

The CDFs suggest that the relative prediction error may follow the normal distribution, so we used quantile-quantile plots to test this. Figure reffig:qqnormal shows

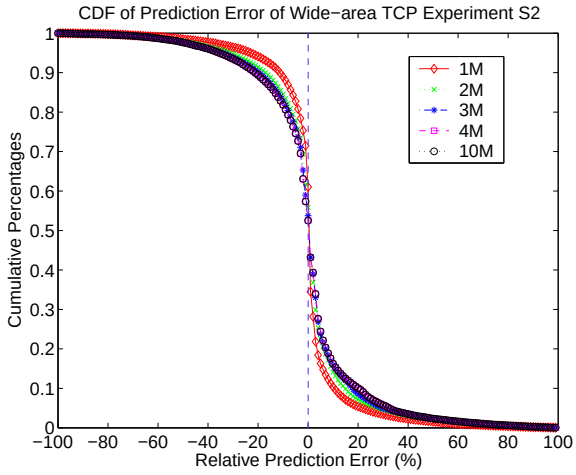


Figure 6.7: CDF of relative prediction error for TCP throughput with different flow sizes.

one example for the qqplot. In almost all cases, we can fit a straight line to these plots with $R^2 \approx 1$, which tells us that our relative error is almost always normal. Normality of prediction errors here is both surprising and extremely useful. In particular, we can simply estimate the variance of the relative prediction error as we measure and predict, and then use this information straightforwardly to create confidence intervals for our predictions. Being able to compute accurate confidence intervals is vital to using predictions in applications [76]. Time series based predictors can be applied to enhance the prediction accuracy, as covered in Section 6.5.

In practice, we don't have to send two probes. Instead, we can send the larger one of the two probes, record its starting time, the time when as much data as the size of the small probe was sent and full probe's finishing time. We call such a probe a *dualPacket*[1].

For most paths, the larger the flow size is, the bigger the standard deviation

[1]It is possible to use more than two probes, e.g. at 200K, 400K, 600K and 800K, and then apply linear regression.

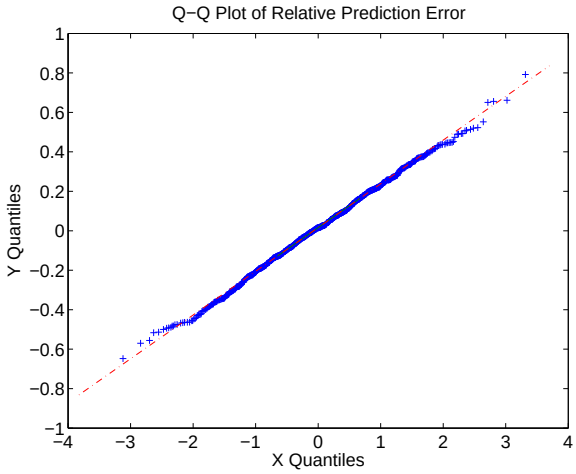


Figure 6.8: Quantile-quantile plot of relative prediction error with flow size 2MB against standard normal distribution. qqplot of relative prediction error for flows with different sizes looks almost identical to this figure.

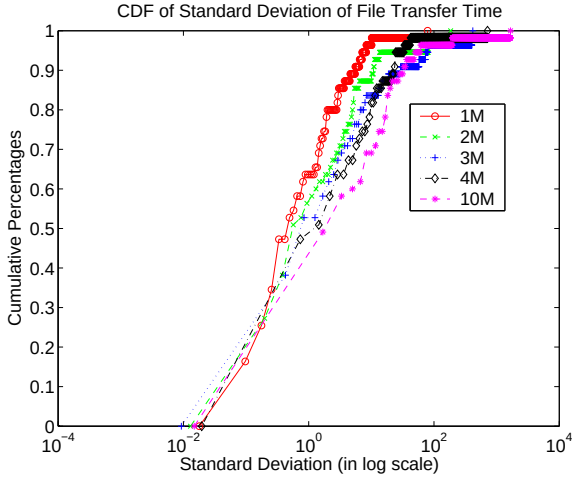


Figure 6.9: CDF of standard deviation of transfer time at all Internet paths for 5 different flow sizes.

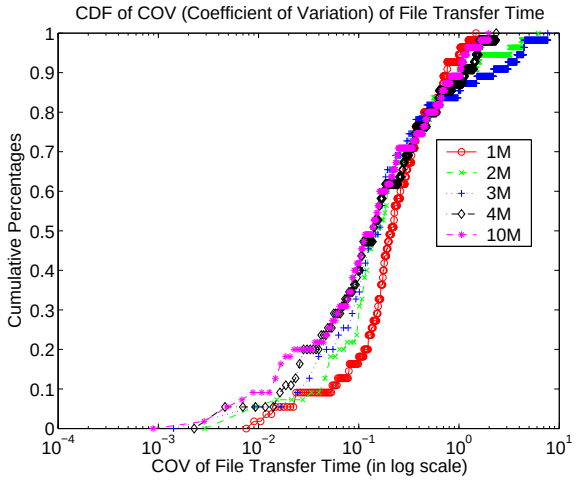


Figure 6.10: CDF of COV (coefficient of variation $COV = \frac{std}{mean}$) of transfer Time at all Internet paths for 5 different flow sizes.

of the transfer time. However, consider Figure 6.10, which shows the CDFs of the coefficient of variation (COV) of transfer time. Here the conclusion is the reverse: for most paths, the bigger the file is, the smaller the COV of the transfer time. This essentially means that in *relative* terms, the variance of transfer time of flows with larger sizes are actually smaller.

As explained in Section 6.3.2 and 6.3.4, we need two probes with different sizes to determine the steady state TCP throughput. Inevitably, fluctuations of flow transfer time happen on the dynamic Internet, and have shown themselves in the standard deviation we have just seen. These fluctuations are the main cause of the estimation error of steady state TCP throughput. Since flows with larger sizes actually have less variance in *relative* terms, estimating steady state throughput using larger flows will certainly be more accurate. On the other hand, probes with larger flows are more expensive. This leads us to the selection of two probe sizes of 400 KBytes and 800 KBytes as default probes, which we feel is a reasonable trade-off between estimation accuracy and probing cost.

6.4 Statistical stability of the Internet

Statistical stability or consistency is one of the most important characteristics of the Internet and is the basis that makes it possible to predict TCP throughput on the wide area network. A good understanding of stability will also help us to make decisions about prediction strategies, such as the frequency of active probing and optimal time series predictors.

6.4.1 Routing stability

Paxson [174] proposed two metrics for route stability, prevalence and persistency. Prevalence, which is of particular interest to us here, is the probability of observing a given route over time. If a route is prevalent, than the observation of it allows us to predict that it will be used again. Persistency is the frequency of route changes. The two metrics are not closely correlated. Paxson's conclusions are that Internet paths are heavily dominated by a single route, but that the time periods over which routes persist show wide variation, ranging from seconds to days. However, 2/3 of the Internet paths Paxson studied had routes that persisted for days to weeks. Chinoy found that route changes tend to concentrate at the edges of the network, not in its "backbone" [61]. Routing stability is the basis of other stabilities or consistency. If the route is changing frequently and quickly, then no other stabilities will hold.

6.4.2 Locality of TCP throughput

Balakrishnan, et al analyzed statistical models for the observed end-to-end network performance based on extensive packet-level traces collected from the primary web site for the Atlanta Summer Olympic Games in 1996. They concluded that nearby

Internet hosts often have almost identical distributions of observed throughput. Although the size of the clusters for which the performance is identical varies as a function of their location on the Internet, cluster sizes in the range of 2 to 4 hops work well for many regions. They also found that end-to-end throughput to hosts often varied by less than a factor of two over timescales on the order of many tens of minutes, and that the throughput was piecewise stationary over timescales of similar magnitude [28]. Myers, et al examined performance from a wide range of clients to a wide range of servers and found that bandwidth to the servers and server rankings from the point of view of a client were remarkably stable over time [167]. Seshan, et al applied these findings in the development of the Shared Passive Network Performance Discovery (SPAND) system [200], which collected server performance information from the point of view of a pool of clients and used that history to predict the performance of new requests.

Zhang, et al [238] experimented by sending 1 MB files every minute between pairs of hosts, and proposed an effective way to evaluate the temporal locality of end-to-end TCP throughput of those flows. He looks at the length of the period where the ratio between the maximum and minimum observed TCP throughput is less than a constant factor ρ. This is referred to as an Operational Constancy Region (OCR). Instead of using OCR, we define a *Statistically Stable Region* (SSR) as the length of the period where the ratio between the maximum and minimum *estimated* steady state TCP throughput is less than a constant factor ρ. The difference between OCR and SSR is important because OCR is only characterizing the throughput for flows with a specific size, while SSR characterizes the steady state throughput for all flows with different sizes. We used traces from Correlation Set to characterize the SSR with steady-state TCP throughput. That is, instead of looking at the TCP throughput of a specific flow size, we estimated steady-state TCP throughput of the path using

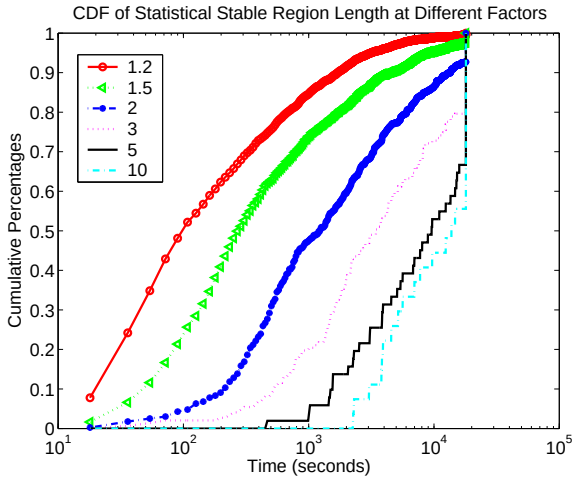


Figure 6.11: CDF of statistically stable region (SSR) for steady-state TCP throughput with different ρ.

Equation 6.1.

Figure 6.11 gives the CDF of length of all SSRs modeled by steady-state TCP throughput from Correlation Set. Each curve in the plot corresponds to a particular value of the constant factor ρ. Under all different values of ρ, some degree of temporal locality is exhibited. As we expected, the larger ρ is, the longer the SSRs tend to be.

For comparison purposes, we also calculated the CDF of OCR with data from Distribution Set. The comparison between ours and Zhang's results [238] suggests that the temporal locality in our test environment is much weaker. For instance, Zhang found that $\approx 60\%$ of OCRs are longer than 1 hour when $\rho = 2$ and $> 80\%$ of all OCRs exceed 3 hours when $\rho = 10$. In our results, the two corresponding numbers drop to 2% and 10% respectively. TCP throughput in our testbed appears to be less stable. We suspect that this difference may largely due to the fact that PlanetLab nodes often become CPU or bandwidth saturated, causing great fluctuations of TCP throughput. It is challenging to predict TCP throughput under a highly dynamic environment.

6.4.3 End-to-end TCP throughput distribution

An important question an application often poses is how the TCP throughput varies, and, beyond that, whether an analytical distribution model can be applied to characterize its distribution. Balakrishnan, et al [28] studied aggregated TCP throughput distribution across all different flow sizes between each pair of Internet hosts. Their statistical analysis suggests that end-to-end TCP throughput can be well modeled as a log-normal distribution. Zhang, et al verified this finding in [239].

Since we have already seen earlier that there exists strong correlation between TCP throughput and flow size, we are therefore more interested in studying the TCP throughput distribution of a particular flow size than in getting an aggregated throughput distribution across all different flow sizes. The data from Distribution Set lets us do this analysis.

Recall that in Distribution Set, for each client/server pair, we repeated the transfer of each file 3,000 times. We histogramed the throughput data for each flow size/path tuple. Almost in every case, the throughput histogram demonstrates a multimodal distribution. This suggests that it is probably not feasible to model long time TCP throughput using simple distributions.

Because the collection of data for each client/server pair lasted several hours or even longer, we suspect that the multimodal feature may be partially due to the change in network conditions during the measurement period. To verify this hypothesis, we try to study throughput distribution using subsets of each dataset. A subset contains much less data and covers shorter measurement length. In other words, we hoped to find "subregions" in each dataset in which the network conditions are relatively stable and the throughput data can be better modeled unimodally.

It is very hard to predefine an optimal length or data size for such "subregions"

in the throughput data; in fact, the appropriate length may vary from time to time. Therefore, we believe it is necessary to adaptively change the subregion length over time as we acquire data (or walk the dataset offline). The purpose is to segment the whole dataset into multiple subregions (or identify segment boundaries online). For each segment, we fit the data with several analytical distributions, and evaluate the goodness of fit using R^2.

Our offline distribution fitting algorithm for TCP throughput has the following steps:

1. Select a trace of TCP throughput (sequence of measurements for a particular flow size on a particular Internet path).

2. Initialize the subregion length, and set the start and end point of the subregion to 1 and 100, respectively.

3. Fit the subregion data with an analytical distribution, and calculate the value of R^2.

4. Increase the subregion length by 100, that is, keep the start point as from the previous step, but increase the end point by 100. For this new subregion, fit the data with the analytical distribution model again, get a new value of R^2. The adjustment granularity can also be changed.

5. Compare the new R^2 with the previous one. If the new one is larger, repeat step 4, otherwise, we have found that previous subregion has the optimal length.

6. Log the start point, end point, and value of R^2 from previous subregion. Reset the subregion length to be 100, and set the start point of the subregion to be one larger than the end point of the previous subregion.

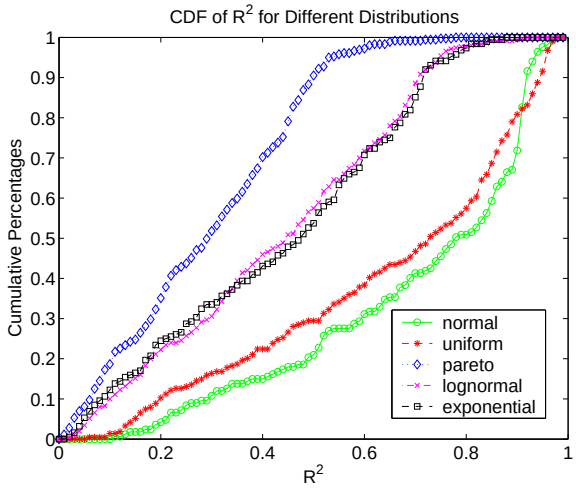


Figure 6.12: CDF of R^2 for five common distributions for TCP throughput characterization on segmented traces. The size of the file is 10 MBytes. Other flow sizes show similar results.

7. Go to step 3 and repeat above procedure, until all data points in the datasets are examined.

We segmented and model-fitted each path/flow size trace in Distribution Set using this algorithm. We then considered the R^2 distribution for each of flow size and analytical distribution. The CDFs of the values of R^2 for each flow size and analytical distribution are shown in Figure 6.12. It is clear that for the five distributions we compared, the normal distribution best fits the TCP throughput data. However, throughput is *nonstationary*, so a given normal distribution holds for only a period of time before it changes to another one. This nonstationary behavior is remarkably similar to the "epochal behavior" pattern of load on hosts that we observed in earlier work [75].

6.5 TCP throughput in real time

Based on our study and previous research, we have developed and evaluated DualPats, a prototype real time TCP throughput prediction service for distributed applications. DualPats actively sends out dualPackets to benchmark a path. It automatically adjusts its rate to capture the SSR on the path and therefore to minimize intrusiveness without losing sampling accuracy. The benchmarking technique is described in Section 6.3.

DualPats is different from all the previous available bandwidth estimation tools such as PathLoad [122]. First, instead of estimating current available bandwidth, DualPats predicts TCP throughput in the next short period of time. Secondly, DualPats monitors the paths and thus can return a prediction immediately. It takes DualPats less than 1 ms to give a prediction on a Pentium III machine, while it takes PathLoad tens of seconds to do one estimation for a path. DualPats don't send probes upon a prediction request, instead it monitors the path and sends out dualPackets according to the dynamic sampling rate adjustment algorithm as described below.

6.5.1 System architecture

DualPats consists of two components, a network sensor and a TCP throughput predictor.

Figure 6.13 illustrates the two components and their relationship with applications and the underlying operating system. The whole system works at application level.

The network sensor sends out dualPackets at a self-adjusting rate as described in Section 6.5.2. It records the sizes and transfer times of each probe in each dualPacket. When monitoring N different TCP connections, N series of probe records are maintained.

The TCP throughput predictor interfaces with both the network sensor and applications. Whenever an application needs a prediction, it sends a query to the TCP throughput predictor, and the predictor executes the following:

1. Parse the query from the application and get parameters including the destination and file size.

2. Fetch the dualPacket data series for the destination from underlying network sensor. If no series exists, a probing process for the destination is started.

3. Apply a prediction model, such as moving average or EWMA, to predict the current transfer times for each of the two probes in the dualPacket.

4. Fit a linear curve as described in Equation 6.1 and calculate the TCP throughput for the given file size using Equation 6.2. (Optionally, compute a confidence interval using normality assumptions).

5. Return the estimated TCP throughput for the transfer time to the application.

We tested several prediction models for step 3, including interval-aware moving average (IAMA), exponential weighted moving average (EWMA) and simply using the last value. An IAMA is similar to a moving average except that the IAMA computes its average over only previous probe values with the same sampling interval. IAMA with window size 20 works best on average in our experiments. We believe that this is so because during each SSR, the end-to-end TCP throughput is best modeled with a normal distribution. For a normal distribution with no serial correlation, the mean is the best predictor possible, and IAMA estimates this.

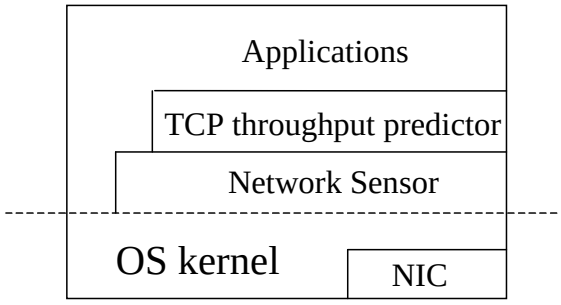


Figure 6.13: System architecture of DualPats.

6.5.2 Dynamic sampling rate adjustment algorithm

There are two ways to decrease the overhead caused by the dualPacket probes: decrease the sampling rate or decrease the size of the dualPacket.

As we discussed in Section 6.4, each Internet path shows statistical stability in TCP throughput. However, each path is different in the length of its SSR. Therefore, instead of using the periodic sampling algorithm used by previous TCP throughput monitoring frameworks such as NWS [230], we designed a simple algorithm to dynamically adjust the sampling rate to the path's SSR. For stable paths with longer SSR, we send fewer probes, while for unstable paths with shorter SSR, we adapt to its dynamics by sampling the path more frequently. The algorithm is:

1. Set an upper bound U and a lower bound L for the sampling interval. They were set as 20 and 1200 seconds in our tests.

2. Set another two relative changing bounds, $B1$, $B2$, in units of percentage. After sending each dualPacket, estimate the current steady-state TCP throughput. If it has changed less than $B1$, increases the sampling interval by a step of S seconds; if it changes between $B1$ and $B2$, keep the current interval; otherwise decrease the interval. In Online Evaluation Set, $B1$, $B2$ were set to be 5% and

15%.

3. The interval must be between L and U.

We also want to minimize the size of dualPacket on the condition that none of them will fall into the noise area as shown in Figure 6.4. However, the noise area is different for each Internet path, as discussed in Section 6.3. It is a function of loss rate and underlying bandwidth. Our algorithm for determining it is:

1. Set a default initial size for the dualPackets. In Online Evaluation Set, we used 400KB and 800KB. Also set an upper bound size U_S for the dualPacket.

2. If M continuous prediction errors are bigger than a threshold T_H, and with the same sign, we increase the probe size by 100KB each.

3. The size of dualPacket must be $\leq U_S$.

6.5.3 Evaluation

Our primary metric is the relative error:

$$err = \frac{PredValue - RealValue}{RealValue} \tag{6.3}$$

DualPats ran $\approx$ 14000 predictions on 50 monitored end-to-end paths during about 10 days. Test cases are randomly chosen 8MB, 40MB, or 160MB files. Details of the evaluation experiments can be found in the Section 6.2 discussion of Online Evaluation Set.

Our detailed results for each path are available elsewhere [149]. To summarize them for this chapter, we use the following metrics. Mean error is calculated by averaging all of the relative errors. For an unbiased predictor, this value should

be close to zero given enough test cases. We can see that in our evaluation it is quite small in most cases, and we see an roughly equal proportion of positive and negative mean errors. The mean $abs(err)$ is the average of the absolute value of the relative error. We consider it the most important metric in evaluating the predictions. Mean stderr is the standard deviation of relative error while mean $abs(stderr)$ is the standard deviation of the absolute value of relative error.

DualPats is accurate: Figure 6.15 shows the CDF of the mean error and mean $abs(err)$ of our results. Figure 6.17 shows the CDF of the standard deviation for relative errors. Over 70% of the predictions have mean error within [-0.1, 0.1], and all of them are within [-0.21, 0.22]. About 90% of the predictions have mean stderr smaller than 0.2. About 30% of the predictions have mean $abs(err)$ within 0.1, while over 90% of the predictions has mean $abs(err)$ below 0.2, and about 95% of them are below 0.25. Over 90% of the predictions have mean $abs(stderr)$ smaller than 0.35.

We studied the correlation among the prediction errors and several known path properties. The results are shown in Figure 6.19. We define ($|R| > 0.3$) as being weakly correlated, ($0.3 \leq |R| \leq 0.8$) being medium correlated, and ($|R| > 0.8$) being strongly correlated. Clearly, the mean error is not related to any others, which further suggests that the predictions given by DualPats are unbiased. However, if the path is very dynamic it is hard to predict. Figure 6.19 shows that R between the mean absolute error and the sampling interval length (and, indirectly, the SSR) is negatively and very weakly correlated. This implies that our algorithm captured the path dynamics and effectively adjusted to its changes. The mean interval and mean standard deviation of error show the strongest correlation in Figure 6.19. This is because both longer mean interval and smaller mean standard deviation of error imply a stable path. Also, we can see that number of hops is weakly correlated with mean RTT.

Path	Router Hops	Mean RTT	Mean error	Mean stderr	Mean abs(error)	Mean abs(stderr)	Mean Interval
1	20	55	-0.0073	0.11	0.069	0.13	641.97
2	18	60	0.10	0.17	0.17	0.18	29.44
3	17	33	-0.21	0.23	0.25	0.51	132.2
4	11	27.5	-0.03	0.19	0.13	0.25	71.56
5	13	31	-0.04	0.20	0.16	0.28	48.76
6	16	138	-0.079	0.19	0.14	0.29	58.18
7	16	120	0.048	0.355	0.28	0.42	21.87
8	14	51	0.021	0.12	0.095	0.168	512.64
9	18	207	-0.14	0.17	0.18	0.36	51.50
10	14	29	-0.11	0.19	0.14	0.31	180.17
11	19	110	-0.036	0.18	0.11	0.24	28.57
12	15	36	-0.038	0.14	0.078	0.18	258.16
13	17	59	0.035	0.208	0.16	0.24	32.23
14	12	23.5	-0.012	0.060	0.042	0.082	320.97
15	13	28	-0.095	0.186	0.14	0.31	511.33
16	18	100	-0.028	0.16	0.11	0.21	543.75
17	19	70	-0.083	0.030	0.083	0.17	543.63
18	14	81	-0.076	0.025	0.076	0.154	522.20
19	19	72	0.21	0.38	0.29	0.39	48.39
20	17	50	0.11	0.12	0.14	0.12	97.25
21	13	31	-0.068	0.082	0.082	0.17	246.53
22	16	135	0.022	0.18	0.14	0.21	23.93
23	14	36	-0.12	0.16	0.13	0.29	194.04
24	14	51	-0.15	0.14	0.15	0.33	542.34
25	12	24	-0.10	0.10	0.12	0.24	59.41
26	18	148	0.0023	0.18	0.13	0.22	26.74
27	12	48	-0.068	0.11	0.092	0.19	165.86
28	15	113	-0.020	0.10	0.089	0.15	55.85
29	9	181	-0.076	0.060	0.079	0.17	652.53
30	18	208	-0.062	0.095	0.088	0.18	40.50
31	7	2	-0.018	0.12	0.094	0.17	29.09
32	10	25	-0.061	0.034	0.061	0.13	521.16
33	13	33	0.22	0.25	0.25	0.25	423.75
34	16	141	-0.041	0.12	0.11	0.19	26.00
35	14	51	-0.14	0.17	0.15	0.34	600.65
36	16	24	-0.049	0.14	0.10	0.20	481.57
37	14	35	0.025	0.12	0.089	0.14	45.85
38	19	117	-0.059	0.042	0.061	0.13	77.08
39	13	34	-0.057	0.045	0.058	0.12	643.84
40	13	58	-0.13	0.015	0.12	0.27	73.33
41	18	208	-0.075	0.11	0.099	0.21	27.97
42	11	51	-0.063	0.034	0.064	0.13	810.53
43	11	175	-0.052	0.082	0.073	0.15	25.71
44	18	60	-0.095	0.078	0.095	0.21	575.81
45	15	58	0.14	0.20	0.18	0.21	27.64
46	11	50	-0.075	0.038	0.075	0.16	848.39
47	14	36	0.13	0.17	0.10	0.28	148.86
48	15	24	-0.099	0.24	0.19	0.38	24.92
49	21	86	-0.087	0.15	0.11	0.25	30.28
50	19	47	-0.042	0.17	0.12	0.23	24.80

Figure 6.14: Prediction error statistics for Online Evaluation Set. RTT is the round trip time between the two sites in milliseconds, and Mean Interval is the average interval time between dualPackets in seconds. Mean error is the average relative error while mean abs(error) is the average of the absolute value of relative error. Mean stderr is the standard deviation of relative error while mean abs(stderr) is the standard deviation of the absolute value of relative error.

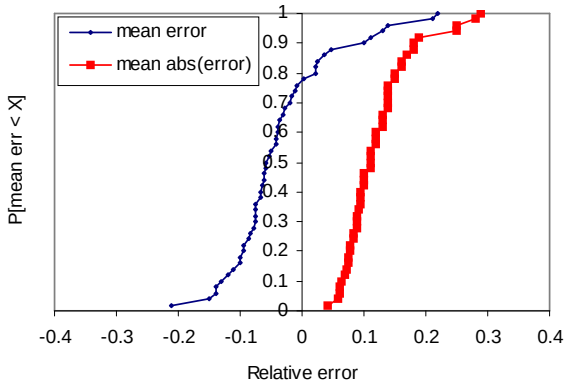


Figure 6.15: CDF of relative errors.

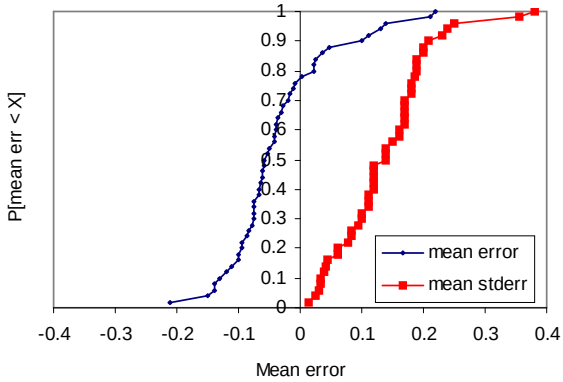


Figure 6.16: CDF of mean and standard deviation of relative error.

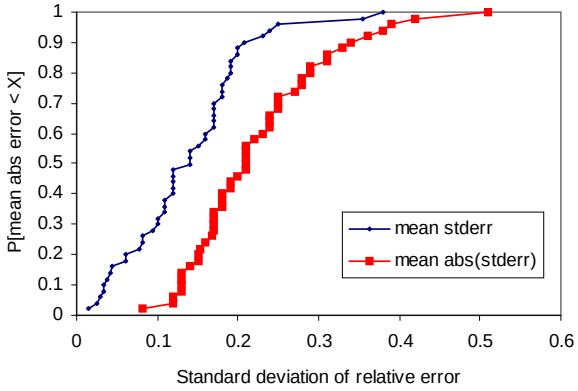


Figure 6.17: CDF of standard deviation of relative errors.

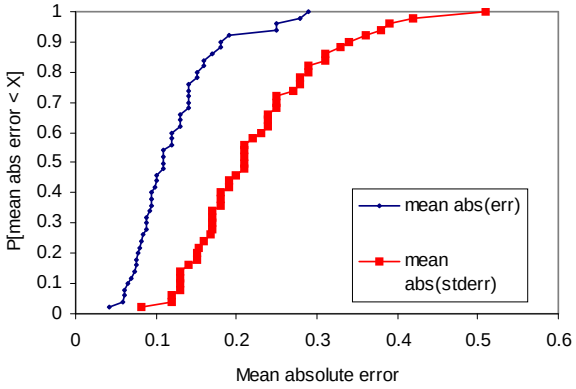


Figure 6.18: CDF of mean and standard deviation of absolute relative error.

	Router Hops	Mean RTT	Mean Interval
Mean abs(err)	0.24	-0.024	-0.36
Mean abs(stderr)	0.19	-0.018	-0.28
Mean err	0.10	-0.081	-0.20
Mean stderr	0.30	-0.031	-0.44
Router Hops	1.00	0.34	-0.27
Mean RTT	0.34	1.00	-0.25
Mean Interval	-0.27	-0.25	1.00

Figure 6.19: Correlation coefficient R among prediction error and path properties.

Recall from Section 6.1 that Sudharshan, et al [222, 221] showed that NWS was predicting less than 1/10 of the actual TCP throughput achieved by GridFTP with large file transfers. Our evaluations show that DualPats does an effective job of predicting TCP throughput for large transfers. Also recall that log-based prediction techniques [209, 221] work only for the host pairs that have recent data exchange history, and the data to be sent is of similar sizes as in the log. Our conclusion is that DualPats achieves comparable or even better performance without such constraints.

Our evaluation is conservative: Jin, et al showed that end-system capability can have significant effects on the network bandwidth estimation algorithms [128]. They showed that resolution of the timer, the time to perform a system call, the

interrupt delay and the system I/O bandwidth all can affect network bandwidth estimation. They compared packet pair dispersion against packet train based algorithm and concluded that packet train based algorithms are less sensitive to the resolution of the system timer and less affected by I/O interrupt delays. This implies that high system load has negative effects on bandwidth estimation algorithms.

We ran the experiments of Online Evaluation Set on PlanetLab, which is typically heavy loaded. Using data available from the CoDeeN project web site [3], we found that CPU load averages were very high on the machines we used. 50% of the machines had Unix load averages that exceeded 5.0. Recall that in Section 6.4 we compared our measured OCR with that shown by Zhang, et al [238], and found that TCP throughput on PlanetLab is more dynamic than Zhang found. This suggests that our prediction would probably do better in a more typical, lightly loaded environment, and that DualPats is robust in the face of high load conditions. We speculate that the robustness of DualPats is related to the TCP benchmarking approach being used: we are measuring the application-to-application transfer time, and thus our prediction must necessarily incorporate the end-system behavior as well.

To get a sense of the CPU load average on the PlanetLab nodes, we took the CPU load average statistics data from the CoDeeN project web site [3] and summarized the data in Figure 6.20 as a CDF. The data is a snapshot for when our evaluation experiments (Online Evaluation Set) were running. The CDF shows the load average of 85 nodes. We can see that more than 90% of the nodes had a load average above 1, about 55% of the nodes had load average above 5, and about 15% of the nodes had load average above 10.

DualPats overhead is low: The overhead of DualPats mainly comes from the dualPackets sent. In our current implementation, we use the default 800KB probe sent at the rate controlled by our dynamic sampling rate adjustment algorithm

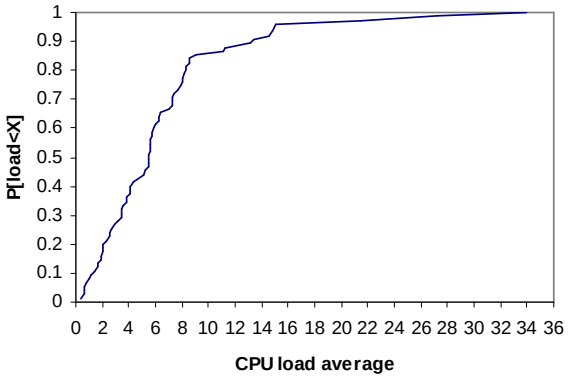


Figure 6.20: CDF of CPU load average on a subset of PlanetLab nodes.

resulting in an overhead on the network of 800KB / (mean Interval). In a highly dynamic testbed like PlanetLab, close to 30% of the mean intervals achieved by DualPats are longer than 500 seconds (800KB/500Sec=1.6KB/Sec), close to 50% are longer than 180 seconds (800KB/180Sec=4.4KB/Sec), and about 90% are longer than 30 seconds (800KB/30Sec=26.7KB/Sec). The interval is bounded by our protocol to limit the maximum overhead on the network.

As DualPats is designed mainly for today's high speed networks, we believe the overhead of DualPats is reasonably small. For less dynamic paths, the overhead of DualPats will be further reduced because the dynamic sampling rate adjustment algorithm will automatically increase the sampling interval. In contrast, Strauss, et al [208] reported that Pathload generates between 2.5 and 10 MB of probe traffic per measurement, which is much larger than that of DualPats. The amount of data sent by DualPats is comparable or smaller than current available bandwidth estimators.

6.6 Conclusions and future work

We have characterized the behavior of TCP throughput in the wide area environment, providing additional explanations for the correlation of throughput and flow size and demonstrating how this correlation causes erroneous predictions to be made when using simple TCP benchmarking to characterize a path. In response, we proposed and evaluated a new benchmarking approach, dualPacket, from which TCP throughput for different flow sizes can be derived. We described and evaluated the performance of a new TCP throughput monitoring and prediction framework, DualPats, and implemented this approach. We have recently extended our work to support throughput prediction for parallel TCP [150].

Like all benchmarking-based systems, our approach has scalability limits. We have addressed this to some extent with our dynamic sample rate adjustment algorithm. However, we are also considering combining our techniques with passive monitoring as in Wren [236], and hierarchical decomposition as in Remos [141] and NWS Clique [231]. Our evaluation of DualPats was in a conservative, heavily loaded environment, but its performance and robustness in the next generation high speed Internet are yet to be explored.

DualPats can be used by RGIS servers to monitor the TCP throughput between the servers.

Chapter 7

Modeling and Taming Parallel TCP

This chapter describes our technique in modeling and predicting the throughput of parallel TCP flows and their influence on the background traffic. Parallel TCP flows can be used to tune the throughput between the RGIS servers without distrubing other data traffic.

7.1 Introduction

Data intensive computing applications require efficient management and transfer of terabytes of data over wide area networks. For example, the Large Hadron Collider (LHC) at the European physics center CERN is predicted to generate several petabytes of raw and derived data per year for approximately 15 years starting from 2005 [22]. Data grids aim to provide the essential infrastructure and services for these applications, and a reliable, high-speed data transfer service is a fundamental and critical component.

```
struct ParallelTCPChar {
  int    num_flows;
  double max_nondisruptive_thru ;
  double cross_traffic_impact ;
};
ParallelTCPChar *
TameParallelTCP(Address  dest,
            double  maximpact);
```

Figure 7.1: The TameParallelTCP() function.

Recent research has demonstrated that the actual TCP throughput achieved by applications is, persistently, significantly smaller than the physical bandwidth "available" according to the end-to-end structural and load characteristics of the network [202]. Here, we define *TCP throughput* as the ratio of effective data over its transfer time, also called *goodput* [183].

Parallel TCP flows have been widely used to increase throughput. For example, GridFTP [21], part of the Globus project [95], supports parallel data transfer and has been widely used in computational grids [22].

A key challenge in using parallel TCP is determining the number of flows to use for a particular transfer. This number affects both the throughput that the transfer will achieve and the impact that it will have on other traffic sharing links with these data flows. While there has been significant previous work on the understanding of parallel TCP performance, no practical parallel TCP throughput prediction techniques exist and there is no analysis work or system that can support the following API call as shown in Figure7.1.

Here, the user calls `TameParallelTCP()` with the destination of her transfer and the maximum percentage impact she is willing to have on cross traffic. The call evaluates the path and returns the number of parallel flows she should use to achieve the maximum possible throughput, while causing no more impact than the specified.

We refer to this as the *maximum nondisruptive throughput (MNT).*

The following sections address the implementation of such a function. With this in mind, we look for answers to the following questions:

- How does parallel TCP affect the throughput of the user's transfer, the throughput of cross traffic, and the aggregate throughput, in different scenarios?
- How can these throughputs be predicted, online and with a small set of measurements, as functions of the number of parallel TCP flows?
- How can these predictions be used to implement the `TameParallelTCP()` function?

To the best of our knowledge, we are the first to propose a practical mechanism to predict the throughput of parallel TCP flows and to answer `TameParallelTCP()`-like questions by estimating the impact on the cross traffic.

Throughout this chapter, we use "parallelism level" interchangeably with "the number of parallel TCP flows". A version of our `TameParallelTCP()` implementation is available from
http://plab.cs.northwestern.edu/Clairvoyance/Tame.html

7.2 Related work

The available bandwidth of a path is defined as "the maximum rate that the path can provide to a flow, without reducing the rate of the rest of the traffic." [117, 122]. Available bandwidth has been a central topic of research in packet networks over the years. To measure it accurately, quickly, and non-intrusively, researchers have developed a variety of algorithms and systems. Tools that measure either the

bottleneck link capacity or the available bandwidth include IGI [117], Remos [141], Nettimer [134] and pathload [122], among others. Most of these tools use packet pair or packet train techniques to conduct the measurements and typically take a long time to converge.

Previous research [134] has shown that, in most cases, the throughput that TCP achieves is considerably lower than the available bandwidth. Parallel TCP is one response to this observation. Sivakumar et al. [202] present PSockets, a library that stripes data over several sockets and evaluate its performance through wide-area experimentation. The authors concluded that this approach can enhance TCP throughput and, in certain situations, be more effective than tuning the TCP window size. Allcock et al. [22] evaluate the performance of parallel GridFTP data transfers on the wide-area, and applied GridFTP to the data management and transfer service in Grid environments.

Considerable effort has been spent on understanding the aggregate behavior of parallel TCP flows on wide area networks. Shenker et al [201] were first to point out that a small number of TCP connections with the same RTT and bottleneck can get their congestion window synchronized. Qiu et al. [183] studied the aggregate TCP throughput, goodput and loss probability on a bottleneck link via extensive ns2-based simulations. They found that a large number of TCP flows with the same round trip time (RTT) can also become synchronized on the bottleneck link when the average size of each TCP congestion window is larger than three packets. A detailed explanation for this synchronization was given in [183]. Due to global synchronization, all the flows share the resource fairly: in the steady state they experience the same loss rate, RTT and thus the same bandwidth.

Considerable effort has been spent on understanding the aggregate behavior of parallel TCP flows on wide area networks. Shenker et al [201] were first to point

out that a small number of TCP connections with the same RTT and bottleneck can get their congestion window synchronized. Qiu et al. [183] studied the aggregate TCP throughput, goodput and loss probability on a bottleneck link via extensive ns2-based simulations. The authors found that a large number of TCP flows with the same round trip time (RTT) can also become synchronized on the bottleneck link when the average size of each TCP congestion window is larger than three packets. The reason for the synchronization is that, at the end of each epoch, the bottleneck buffer becomes full and each flow incurs a loss in the same RTT when it increments its congestion window. Since most flows have more than three outstanding packets before the loss, they can recover from the loss by fast retransmission and reduce the window by half, leading to global synchronization. Due to the global synchronization, all the flows share the resource fairly: in the steady state they experience the same loss rate, RTT and thus same bandwidth. Their findings are highly significant for our work, as they support our assumption that parallel TCP flows on the bottleneck link share the same loss rate.

The work most relevant to ours is that of Hacker et al [108]. The authors observe that parallel TCP increases aggregate throughput by recovering faster from a loss event when the network is not congested. The authors go on to propose a theoretical model for the upper bound of parallel TCP throughput for an uncongested path. The model produces a tight upper bound only if the network is not congested before and after adding the parallel TCP flows; the aggregated throughput then increases linearly with the number of parallel TCP flows. Clearly this reduces the utility of the model as networks are often congested.

Hacker et al also concluded that, in the absence of congestion, the use of parallel TCP flows is equivalent to using a large MSS on a single flow, with the added benefit of reducing the negative effects of random packet loss. They advise application

developers not to use an arbitrary large number of parallel TCP flows, but conclude that it is difficult, if not impossible, to determine the point of congestion in the end-to-end path a priori, and therefore to decide on the proper number of parallel TCP flows.

Most TCP throughput models have limited practical utility due to the difficulty of obtaining accurate model parameters such as TCP loss rate and RTT. For example, Goyal et al [105] concluded that it is hard to obtain accurate estimates of network loss rates as observed by TCP flows using probing methods, and that polling SNMP MIBs on the routers can do much better. However, the MIB statistics are for the aggregate traffic crossing a interface on the router while it is well-known that TCP has a bias against long round trip time connections [183]; the approach is thus limited to those paths where the bottleneck router is using RED. It is also necessary in this and similar approaches to determine the bottleneck router on the end-to-end path (a difficult problem) and have SNMP access to it (rarely available today). Even if this is possible, with current models for parallel TCP we would have to know the loss rate *after* adding in n parallel TCP flows. However, even with the tools like web100 [159], we cannot obtain this rate by simply measuring the network.

Our work makes the following new contributions to the state of the art:

- We predict throughput for *both* congested and uncongested paths as a function of the level of parallelism.
- We estimate the impact of parallel TCP on cross traffic as a function of the level of parallelism.
- We do so using only a small number of probes and no additional tools.

It is widely believed that, under congested situations, parallel TCP flows achieve better performance by effectively behaving unfairly, stealing bandwidth from cross

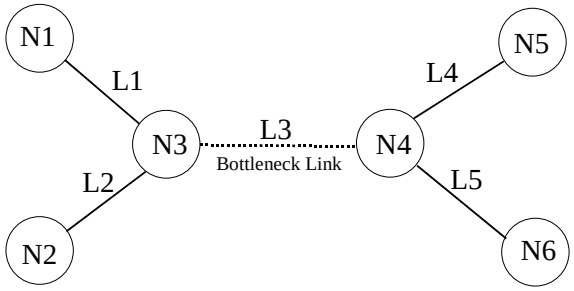


Figure 7.2: Simulation Topology. Cross traffic goes from node N1 to N5, while parallel TCP flows go from node N2 to N6. Cross traffic and parallel TCPs share the same bottleneck link L_3. Each simulation lasts 100 seconds with individual TCP cross traffic flows starting randomly during the first 8 seconds, and all parallel TCPs starting simultaneously at time 10 sec.

traffic. This has prompted some researchers to propose modifying TCP in order to make it better suited for parallel transfers by considering both efficiency and fairness [109, 110]. We believe it will be difficult to persuade people to modify their TCP implementations just to use parallel TCP more fairly. By relying on our prediction tools, a user or administrator should be able to trade off a transfer's throughput and its degree of impact on cross traffic, achieving what we refer to as the *maximum nondisruptive throughput (MNT)*. All these are at application level without requiring modifications to pre-existing TCP implementations.

7.3 Analyzing parallel TCP throughput

In this section, we use simulation to understand the behavior of parallel TCP under different scenarios. For all our simulation-based studies we make use of the ns2 network simulator [5].

7.3.1 Simulation Setup

In a simulation study on aggregate TCP throughput on a bottleneck link, Qiu et al. [183] developed a simple yet realistic topology model for wide-area Internet connections based on the Internet hierarchical routing structure (Figure 7.2). We adopt this same topology for our simulations. Each simulation is 100 seconds long, with cross traffic randomly starting during the first 8 seconds and parallel TCP flows all starting at 10 seconds into the simulation. Cross traffic goes from N1 to N5, while parallel TCP flows go from N2 to N6. The bottleneck link is L3. We employ TCP Reno [87] for both cross traffic and parallel TCP flows, as this is the most widely deployed TCP congestion control algorithm. In addition, comparable results were obtained using TCP Tahoe. Both DropTail and Random Early Detection (RED) [92] queue management policies are studied as they are the most commonly used queue management policies on the Internet. DropTail and RED have similar performance in most our simulations. The exception is in Scenario 1. Here, when there are more than 10 cross traffic flows, the cross traffic dominates the queue and starves the parallel TCP flows under the DropTail policy. Unless otherwise noted, we show results for the DropTail policy.

We use TCP flows as cross traffic because of TCP's dominance in the current Internet, as reported in the the work by Smith et al. [203], in which TCP accounted for 90-91% of the packets and about 90-96% of the bytes transferred in traces collected in 1999-2000 from a educational institution (UNC) and a research lab (NLANR).

We analyze Parallel TCP throughput under a variety of representative scenarios including a typical slow connection such as cable or DSL (Scenario 1), a coast-to-coast high-speed Internet connection (Scenario 2) and a current (Scenario 3) and next generation global-scale Internet connections (Scenario 4). Two additional scenarios

Scenario	L_3 latency	L_3 BW	L_1,L_2 BW	L_4,L_5 BW	TCP buffer
1	20 ms	1.5 Mbps	10 Mbps	10 Mbps	$\geq$ Bandwidth*RTT
2	20 ms	100 Mbps	1000 Mbps	1000 Mbps	$\geq$ Bandwidth*RTT
3	50 ms	100 Mbps	1000 Mbps	1000 Mbps	$\geq$ Bandwidth*RTT
4	50 ms	1000 Mbps	10000 Mbps	10000 Mbps	$\geq$ Bandwidth*RTT
5	50 ms	1000 Mbps	10000 Mbps	10000 Mbps	60 KB
6	20 ms	100 Mbps	1000 Mbps	1000 Mbps	60 KB

Figure 7.3: Bandwidth and latency configuration for different scenarios. The latency for L_1 and L_2 is fixed at 4 milliseconds, while the latency for L_4 and L_5 is fixed at 5 milliseconds. The buffer size on each node is fixed at 25 packets. Both DropTail and RED queue management policies are simulated.

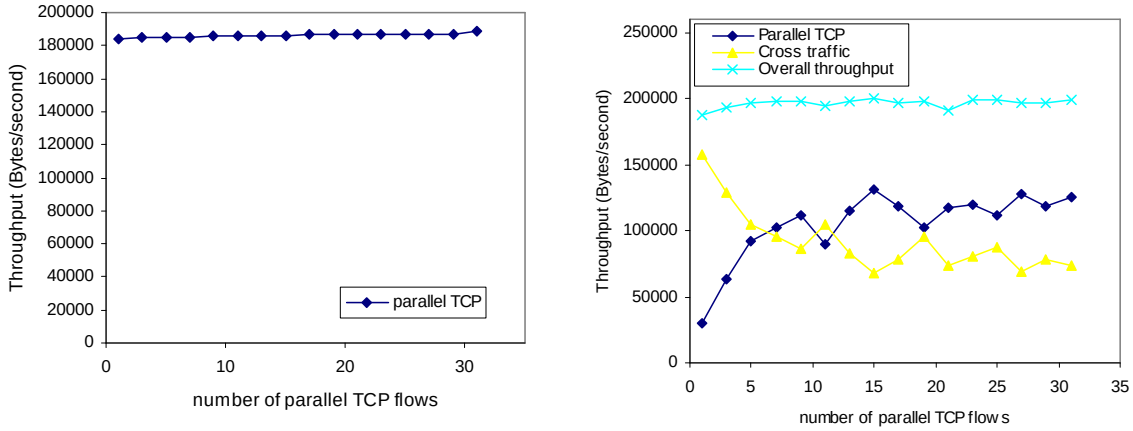
(Scenarios 5 and 6) are used to represent cases where the TCP buffer has not been appropriately tuned [217]. Figure 7.3 summarizes the different simulation scenarios. For each scenario, we simulate from 1 to 31 parallel TCP flows with 5, 10, 15, 20, 25 and 30 random TCP cross traffic flows.

7.3.2 Simulation results

Figures 7.4 to 7.9 plot the aggregated throughput of parallel TCP as a function of the number of flows used for the different scenarios. Plots are shown both without (left graph) and with (right graph) cross traffic. In the latter case, we also plot the cross traffic's and total throughput, i.e. the sum of both the parallel TCP and cross traffic throughputs. We summarize our results in this section. Much more detail is available in our technical report [148].

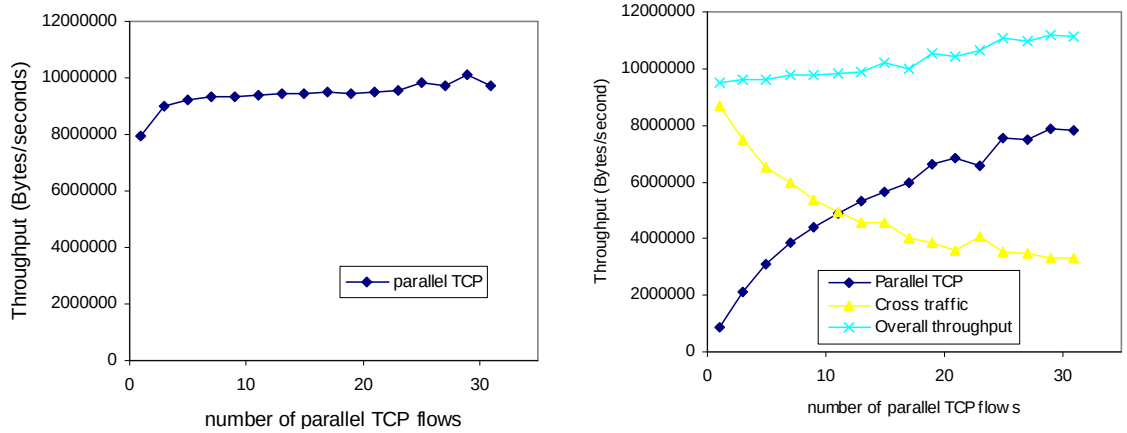
Scenario 1 is used to represent a typical slow connection. Our simulations show that that the primary benefit from parallel TCP comes from being able to steal bandwidth from the existing cross traffic.

Scenario 2 represents a current coast-to-coast connection with low latency and medium bandwidth. Our simulations show that there are some limited benefits from



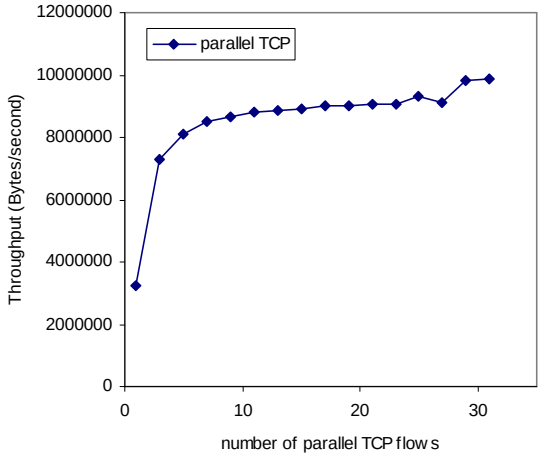
(a) No cross traffic (b) 5 cross traffic

Figure 7.4: Simulation results for scenario 1: latency of L_3 is 20 ms; bandwidth of L_3 is 1.5 Mbps; TCP buffer is properly tuned. Refer to Figure 7.3 for details.

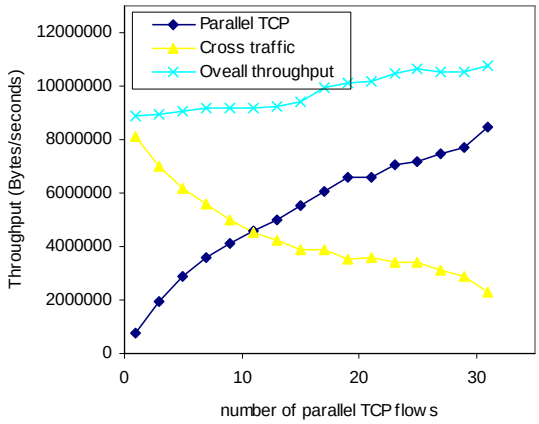


(a) No cross traffic (b) 10 cross traffic

Figure 7.5: Simulation results for scenario 2: latency of L_3 is 20 ms; bandwidth of L_3 is 100 Mbps; TCP buffer is properly tuned. Refer to Figure 7.3 for details.

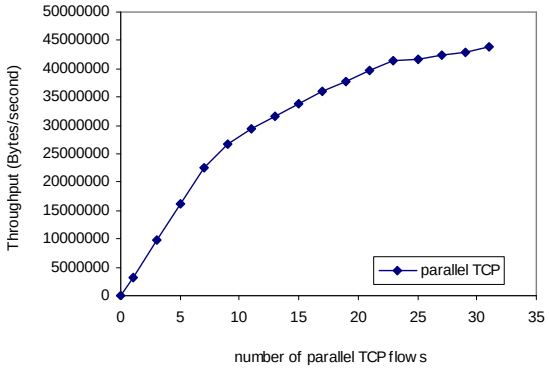


(a) No cross traffic

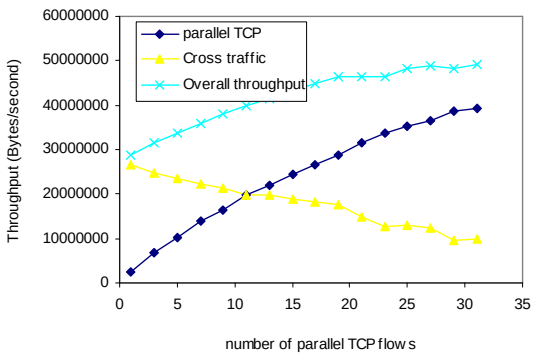


(b) 10 cross traffic

Figure 7.6: Simulation results for scenario 3: latency of L_3 is 50 ms; bandwidth of L_3 is 100 Mbps; TCP buffer is properly tuned. Refer to Figure 7.3 for details.

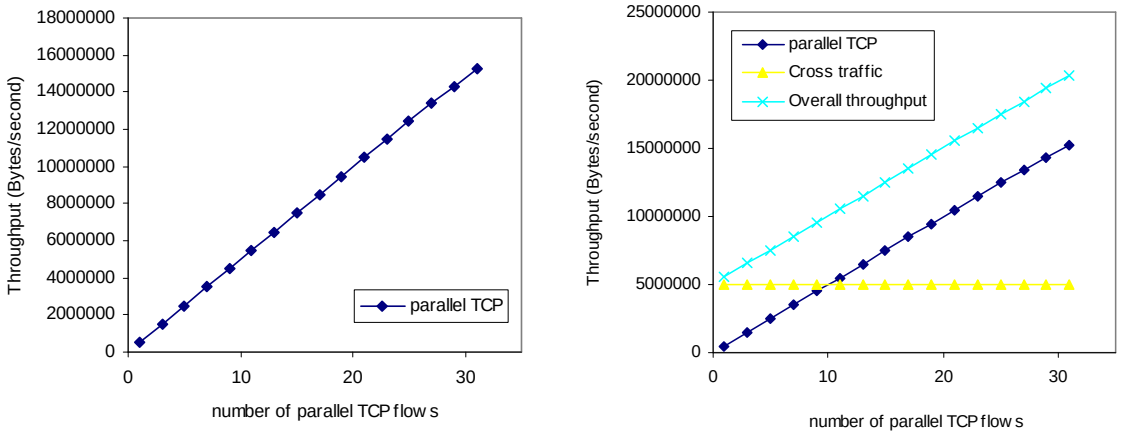


(a) No cross traffic



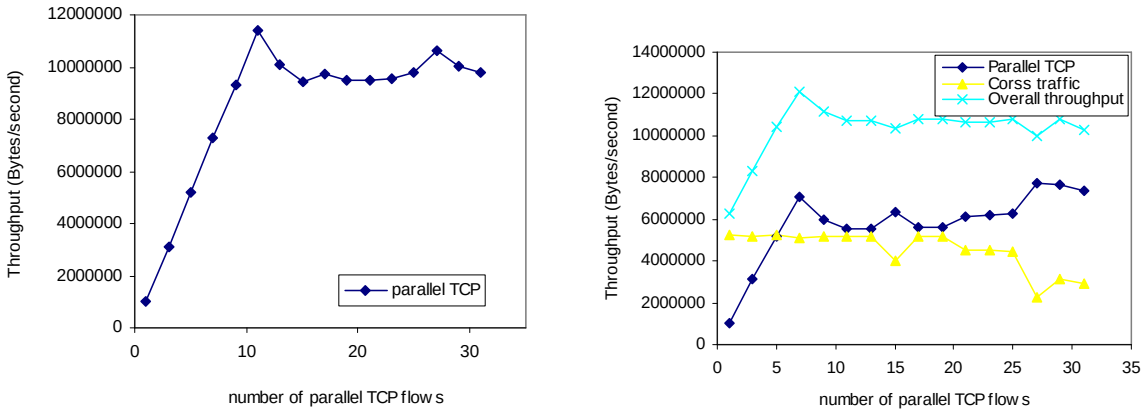
(b) 10 cross traffic

Figure 7.7: Simulation results for scenario 4: latency of L_3 is 50 ms; bandwidth of L_3 is 1000 Mbps; TCP buffer is properly tuned. Refer to Figure 7.3 for details.



(a) No cross traffic (b) 10 cross traffic

Figure 7.8: Simulation results for scenario 5: latency of L_3 is 50 ms; bandwidth of L_3 is 1000 Mbps; TCP buffer is not properly tuned. Refer to Figure 7.3 for details.



(a) No cross traffic (b) 5 cross traffic

Figure 7.9: Simulation results for scenario 6: latency of L_3 is 20 ms; bandwidth of L_3 is 100 Mbps; TCP buffer is not properly tuned. Refer to Figure 7.3 for details.

using parallel TCP without competition in this scenario. In the presence of cross traffic, however, parallel TCP is an even stronger competitor. Parallel TCP allows us to increase overall throughput, albeit marginally.

Scenario 3 is a high latency, medium bandwidth link representing a current global-scale fast Internet connection. In this case there are significant benefits to using parallel TCP even in the absence of cross traffic. The performance of parallel TCP under scenarios 2 and 3, without cross traffic, can be explained using Hacker's theory [108] that parallel TCP recovers faster than single TCP when there is a time out. This effect is more important as the RTT increases, because the time out will be longer and a single TCP cannot recover fast enough.

The benefit of using parallel TCP, with and without cross traffic, are quite high in Scenario 4. Additional throughput in the presence of cross traffic, is mainly due to an increase in overall throughput.

The advantage of parallel TCP is even more significant with mistuned TCP buffers. Scenario 5 represents a high bandwidth and high latency link with a small socket buffer size. The benefits of parallel TCP are quite high, regardless of the amount of cross traffic. These gains come at no cost to the existing cross traffic. Parallel TCP gains performance not only by recovering faster after a time out, but also by providing an effectively larger buffer size. There are diminishing returns as the number of flows is increased. Scenario 6 is similar.

7.3.3 Observations

The dramatically different behaviors of the previous section illustrate the challenges in providing a sound `TameParallelTCP()`-like call. Parallel TCP and cross traffic as functions of the number of flows adopt a wide range of forms, depending on the

topology of the network and the configuration of endpoints. In addition, even if one were to disregard the almost prohibitively high costs of directly measuring these curves, the cross traffic impact would be very difficult to determine. Without a priori knowledge of the parallel TCP loss rate, the model proposed by Hacker, et al [108] only works in uncongested networks like our Scenario 5.

7.4 Modeling and predicting throughput

In this section we combine our simulation work with our analytic treatment of TCP performance to develop a model that can be used to predict the throughput of parallel TCP flows in practice. Our approach only needs to send two probes at different parallelism levels and record their throughput. We don't need any additional tools to measure the RTT and loss rate, which can be hard to obtain in practice as we discussed in Section 7.1.

7.4.1 Algorithm

Mathis et al. [160] developed a simple model for single flow TCP Reno throughput on the assumption that TCP's performance is determined by the congestion avoidance algorithm and that retransmission timeouts are avoided:

$$BW = \frac{MSS}{RTT\sqrt{\frac{2bp}{3}}} \tag{7.1}$$

Here, p is the loss rate or loss probability, and b is the number of packets that are acknowledged by a received message. MSS and RTT are the maximum segment size and round trip time respectively.

Padhye et al. [169] developed an improved single flow TCP Reno throughput model that considers timeout effects.

Bollinger et al [43] show that these two models are essentially equivalent with packet loss rates less than 1/100, which was validated on the current Internet by Zhang et al [239]. Hacker et al. [108], based on Bollinger's findings, present a model for the upper bound of the throughput of n parallel TCP flows. The authors assume that both MSS and RTT are stable. Hacker's upper bound model can be stated as:

$$BW_n \leq \frac{MSS}{RTT}\left(\frac{1}{\sqrt{p_1}} + \frac{1}{\sqrt{p_2}} + \cdots + \frac{1}{\sqrt{p_n}}\right) \tag{7.2}$$

where p_i is the packet loss rate for flow i. However, the authors don't provide any mechanism to estimate the loss rate at other parallel levels for prediction purposes. Therefore, the authors acknowledge that the upper bound is tight only when the network is not congested and the loss rate doesn't increase with more parallel TCP flows. The model only has limited utility otherwise.

In our model, we introduce the notion of the number of cross traffic flows, m, and assume that m does not change dramatically over significantly large time periods. Note that our model doesn't require knowledge of m. Both previous work [240] and our own work on characterizing, modeling, and predicting single flow TCP throughput [151] have shown this assumption to be a valid one.

It is widely believed that the TCP throughput shows statistical stability over considerable periods of time. Balakrishnan et al found that end-to-end TCP throughput to hosts often varied by less than a factor of two over timescales on the order of many tens of minutes, and that the throughput was piecewise stationary over timescales of similar magnitude [28]. Myers et al examined performance from a wide range of clients to a wide range of servers and found that bandwidth to the servers and server

rankings from the point of view of a client were remarkably stable over time [167]. Zhang et al [240] used the notion of the Operational Constancy Region (OCR) to evaluate the temporal locality of end-to-end TCP throughput. The OCR is the length of the period where the ratio between the maximum and minimum observed TCP throughput is less than a constant factor ρ. They found that $\approx 60\%$ of OCRs are longer than 1 hour when $\rho = 2$ and $> 80\%$ of all OCRs exceed 3 hours when $\rho = 10$.

The Internet does, however, dynamically change thus new measurements are necessary when the TCP throughput has significantly changed. The Network Weather Service [230] periodically probes the network to resample the TCP throughput. Instead, our system dynamically resamples the path at each OCR [151]. Dynamic monitoring is beyond the scope of this chapter and is addressed in the previous chapter and our paper [151].

We also assume that all of the parallel TCP flows see the same loss rate and have the same RTT, although both are functions of n and m. These two assumptions have been independently verified [183], as discussed in Section 7.2. We denote with p_n the loss rate after adding n parallel TCP connections, and with RTT_n the round trip time at this point.

Along different paths, the value of MSS can vary ranging from the default 536 bytes to much larger values (for example to support 9000 byte Ethernet jumbo frames on LANs). Our prediction model does not depend on the a priori knowledge of MSS. We do assume, however, that this value does not change after connection establishment. This is a valid assumption as both sides with either use path MTU discovery at connection establishment time [164] or use the default 576 byte path MTU.

Based on Equation 7.1 and the assumptions discussed above, we developed the

following parallel TCP throughput model that essentially sums n TCP flows:

$$BW_n = \frac{MSS}{RTT_n} \frac{n}{\sqrt{p_n}} \frac{c_1}{\sqrt{\frac{2b}{3}}} \tag{7.3}$$

The TCP flows share the same RTT and loss rate and thus the same throughput. Both p_n and RTT_n are actually functions of n and m. Given that we assume m is stable during a period of time, we treat them as functions of n alone. c_1 is a constant in the range $(0, 1]$ that we use to represent the effects of TCP timeouts. In the following, we assume that c_1 is stable for a path over at least short periods, so that our model is equivalent to Padhye's model with timeout considerations [169]. This assumption is firmly supported by the plethora of research on the statistical stability of TCP throughput as discussed above. Note that c_1 will be canceled in the following derivations, therefore our model doesn't require the knowledge of c_1.

If we had a model that could compute the relationship between p_n, RTT_n and the parallelism level n based on a small set of measurements, we could then use Equation 7.3 to predict the throughput for any parallelism level. This is in essence what we do. We developed several parametric models for this relationship based on measurements.

Morris [165] and Qiu, et al [183, 184] independently found that the loss rate is proportional to the square of the total number of TCP connections on the bottleneck link, namely $(m+n)^2$. Through wide area experiments, Hacker, et al [108, 109] showed that RTT on a given path is stable and can be treated as constant. Similarly, we also assume that RTT is a constant during a short period of time. Therefore we have

$$p_n \times RTT_n^2 = a \times (m+n)^2 + b_1 \tag{7.4}$$

where $b1$ is a constant. Given that m is also a constant, Equation 7.4 is equivalent to a full order 2 polynomial:

$$p_n \times RTT_n^2 = a \times n^2 + b_2 \times n + c_2 \tag{7.5}$$

where $b_2 = 2am$ and $c_2 = am^2 + b_1$. To use Equation 7.5, we need to send three probes at different parallelism levels to determine the value of a, b_2 and c_2. Clearly, there is a trade-off between the sophistication of the model and the number of measurements needed to fit it. Recognizing this trade-off, we simplified the full order 2 polynomial to a partial order 2 polynomial as shown in Equation 7.6. This model requires only two probes to determine the parameters a and b.

$$p_n \times RTT_n^2 = a \times n^2 + b \tag{7.6}$$

Here a and b are parameters to be fit based on measurements. We could further simplify the partial order 2 model to a linear model that also requires two probes.

$$p_n \times RTT_n^2 = a \times n + b \tag{7.7}$$

We measured the performance of these three alternatives in a wide-area testbed [6], and found that

1. Equations 7.5 and 7.6 are better models than Equation 7.7.

2. The full order two polynomial model (Equation 7.5) is not significantly better than the partial order 2 polynomial (Equation 7.6) and can occasionally be worse due to its sensitivity to sampling errors caused by small network fluctuations. Another problem with the full order two polynomial model is that it is sensitive to the choice of probe parallelism.

3. The full order 2 model requires three probes instead of the two needed for the linear and

partial polynomial models.

As a result, we use Equation 7.6 for our system and the discussion in the rest of this chapter, unless otherwise noted.

In order to use the model in practice, we have to actively probe a path at two different parallelism levels. The procedure is derived as follows.

We denote $\frac{c_1}{\sqrt{\frac{2b}{3}}}$ in Equation 7.3 as C. Note that C and MSS are all constants under our assumptions. We define a new variable p'_n:

$$p'_n = p_n \frac{RTT_n^2}{C^2 MSS^2} = a'n^2 + b' \tag{7.8}$$

Combining Equations 7.3 and 7.8, we obtain:

$$BW_n = \frac{n}{\sqrt{p'_n}} \tag{7.9}$$

Based on Equation 7.9, we could use the TCP throughput at two different parallelism levels to predict the TCP throughput at other levels. Let n_1 and n_2 be the two parallelism levels that are probed:

$$BW_{n_1} = \frac{n_1}{\sqrt{p'_{n_1}}} = \frac{n_1}{\sqrt{a'{n_1}^2 + b'}} \tag{7.10}$$

and

$$BW_{n_2} = \frac{n_2}{\sqrt{p'_{n_2}}} = \frac{n_2}{\sqrt{a'{n_2}^2 + b'}} \tag{7.11}$$

From which we can determine:

$$a' = \frac{\frac{{n_2}^2}{{BW_{n_2}}^2} - \frac{{n_1}^2}{{BW_{n_1}}^2}}{{n_2}^2 - {n_1}^2} \tag{7.12}$$

and

$$b' = \frac{{n_1}^2}{{BW_{n_1}}^2} - a'{n_1}^2 \tag{7.13}$$

By substituting a' and b' in Equation 7.8 with the expressions in Equations 7.12 and 7.13, we can now predict the TCP throughput for other levels of parallelism using Equation 7.9.

Notice how our prediction requires only *two* TCP throughput probes, one for each of the two different parallelism levels (n_1 and n_2). Both the probing and the calculation process are simple and incur little overhead, the majority being the communication cost of the two probes.

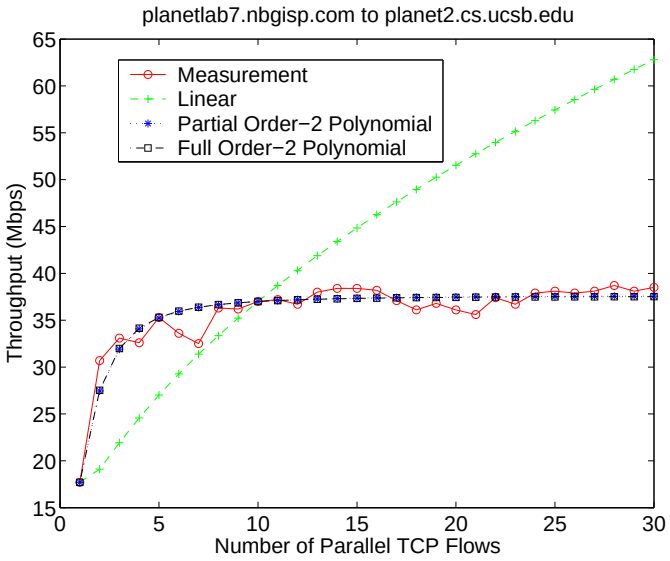
7.4.2 Evaluation

We evaluated our model through online experimentation on PlanetLab [6], a planetary-scale testbed. We randomly choose 41 distinct end-to-end paths with end nodes located in North America, Asia, Europe and Australia. For each path, we conduct 10 rounds of experiments using Iperf [4] to obtain our measurements. A round of experiment starts with two probes for prediction purposes, immediately followed by parallel TCP transfers with up to 30 parallel TCP flows.

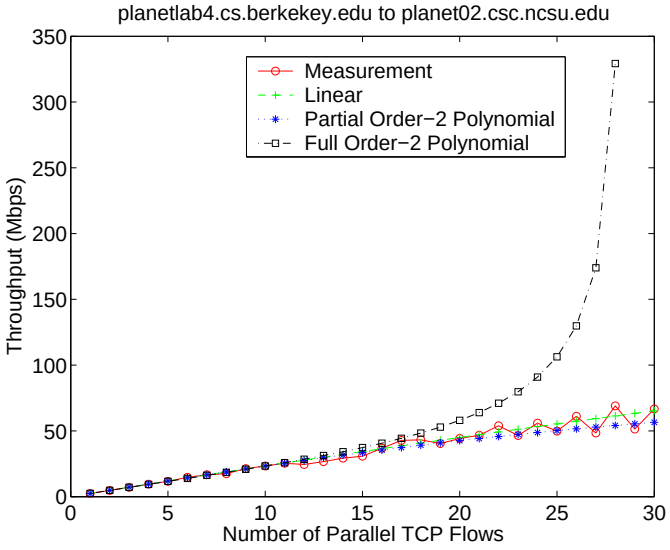
We adopt the *mean relative error* as our performance metric. Relative error is defined as:

$$relativeerror = \frac{prediction - measurement}{measurement} \tag{7.14}$$

Mean relative error on a path is the average of all the relative prediction errors on the path. Mean relative error for a given number of parallel TCP flows is the average of the relative prediction errors of all the experiments for that number of parallel TCP flows.



(a) Example 1: from nbgisp.com to ucsb.edu



(b) Example 2: from berkeley.edu to ncsu.edu

Figure 7.10: Throughput prediction examples.

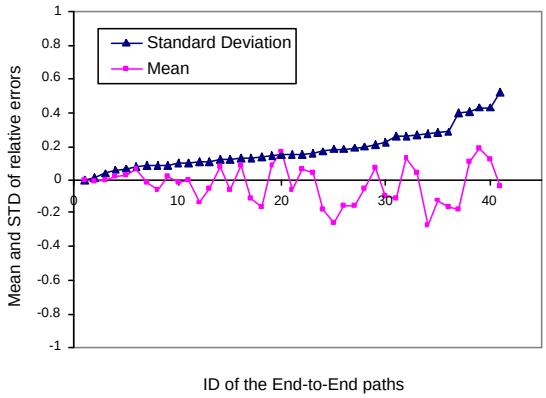


Figure 7.11: Prediction error statistics. Paths ID are ordered by the standard deviation.

Figure 7.10 shows two examples of prediction using our model. The graphs show the actual and predicted throughput (based on measurements at $n_1 = 1$ and $n_2 = 10$). It can be seen that, for Example 1, predictions made based on the partial order 2 and full order 2 polynomials are virtually identical and have similar accuracy, while the prediction curve derived using the linear model deviates significantly from the measurement curve. In our second example, the prediction made using the partial order 2 polynomial and the linear model are virtually identical and equally accurate. The prediction curve generated by the full order 2 polynomial, however, deviates significantly from the measurement curve.

Figure 7.12 shows the performance of our parallel TCP throughput predictor using two probes at parallelism levels $n_1 = 1$ and $n_2 = 10$ for all of the PlanetLab pairs. Only the partial order 2 polynomial model is used here. Both the mean and standard deviation of the relative errors (across different parallelism levels) is shown, with the graph ordered by the standard deviation. The results are quite encouraging: in most cases, our predictions have a small mean and standard deviation of relative prediction errors.

Source Host	Destination Host	Mean	Standard Deviation
planetlab2.postel.org	planetlab-1.cmcl.cs.cmu.edu	0.0822	0.1264
planetlab2.it.uts.edu.au	planetlab1.diku.dk	0.1110	0.4061
planetlab1.millennium.berkeley.edu	planetlab3.cs.uoregon.edu	0.0910	0.1325
planetlab-1.it.uu.se	planet2.cs.ucsb.edu	0.1302	0.2599
ds-pl1.technion.ac.il	planetslug3.cse.ucsc.edu	-0.0453	0.1084
planetlab-2.cs.princeton.edu	planetlab9.cs.berkeley.edu	0.07049	0.15274
planetlab2.flux.utah.edu	planet1.cs.ucsb.edu	-0.1081	0.2583
planetlab2.chin.internet2.planet-lab.org	planetlab5.millennium.berkeley.edu	0.0211	0.0594
planetlab2.frankfurt.interxion.planet-lab.org	planet1.cc.gt.atl.ga.us	-0.1676	0.2887
planetlab2.bgu.ac.il	planetlab1.it.uts.edu.au	-0.0533	0.1264
planetlab2.cs.berkeley.edu	planetslug2.cse.ucsc.edu	-0.0138	0.1033
planet1.calgary.canet4.nodes.planet-lab.org	planetlab2.sanjose.equinix.planet-lab.org	-0.1088	0.1331
planetlab2.cs-ipv6.lancs.ac.uk	planetlab4.cs.berkeley.edu	-0.0518	0.1514
planetlab2.cs.uoregon.edu	planet02.csc.ncsu.edu	-0.0522	0.0895
planetlab2.postel.org	planetlab2.it.uts.edu.au	-0.1208	0.2825
planetlab2.flux.utah.edu	planetlab2.chin.internet2.planet-lab.org	0.1235	0.4334
planetlab2.frankfurt.interxion.planet-lab.org	planetlab2.bgu.ac.il	-0.0956	0.2232
planetlab9.millennium.berkeley.edu	planetlab2.cs.berkeley.edu	-0.0070	0.0111
planetlab2.cs-ipv6.lancs.ac.uk	planetlab2.cs.uoregon.edu	-0.2581	0.1828
planetlab-2.it.uu.se	planetlab3.sanjose.equinix.planet-lab.org	0.0664	0.0848
pli1-pa-3.hpl.hp.com	planetlab7.millennium.berkeley.edu	0.0400	0.1542
planetlab1.cs.berkeley.edu	planetlab01.ethz.ch	-0.1546	0.1937
planetlab3.cs.uoregon.edu	planetlab02.cs.washington.edu	-0.1346	0.1068
planetlab7.nbgisp.com	planet2.cs.ucsb.edu	0.0033	0.0394
planetslug3.cse.ucsc.edu	planetlab9.cs.berkeley.edu	-0.2750	0.2743
planet1.cs.ucsb.edu	planetlab5.millennium.berkeley.edu	0.0743	0.2118
planet1.cc.gt.atl.ga.us	planetlab1.it.uts.edu.au	-0.1753	0.3964
planetlab4.millennium.berkeley.edu	planetslug2.cse.ucsc.edu	5.1483e-04	0.0028
planetlab4.cs.berkeley.edu	planet02.csc.ncsu.edu	-0.0136	0.0882
planetlab2.postel.org	planetlab2.cs.berkeley.edu	0.0029	0.1051
planetlab-2.cmcl.cs.cmu.edu	planetlab2.cs.uoregon.edu	0.0258	0.0664
planetlab2.flux.utah.edu	planetlab3.sanjose.equinix.planet-lab.org	0.1742	0.1491
planetlab2.frankfurt.interxion.planet-lab.org	planetlab2.tau.ac.il	0.03886	0.2650
planetlab9.millennium.berkeley.edu	planetlab1.flux.utah.edu	0.0922	0.1430
planetlab2.cs-ipv6.lancs.ac.uk	planetlab7.millennium.berkeley.edu	-0.1643	0.1345
planetlab-2.it.uu.se	planetlab1.enel.ucalgary.ca	-0.1604	0.1833
s2_803.ie.cuhk.edu.hk	planetlab01.ethz.ch	-0.0375	0.5193
planet2.pittsburgh.intel-research.net	planetlab02.cs.washington.edu	0.190	0.4300
planetlab2.millennium.berkeley.edu	planetlab1.it.uts.edu.au	-0.1769	0.1695
planetlab1.cs.berkeley.edu	planetslug2.cse.ucsc.edu	0.0200	0.0912
planetlab7.nbgisp.com	planetlab5.millennium.berkeley.edu	0.0093	0.0747

Figure 7.12: Relative Prediction Error Statistics for Parallel TCP Throughput.

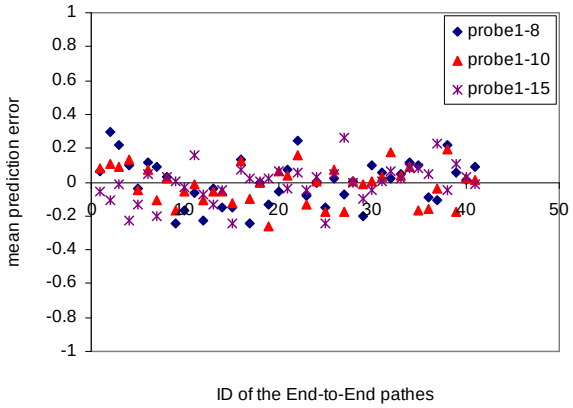


Figure 7.13: Prediction sensitivity to the selection of probes.

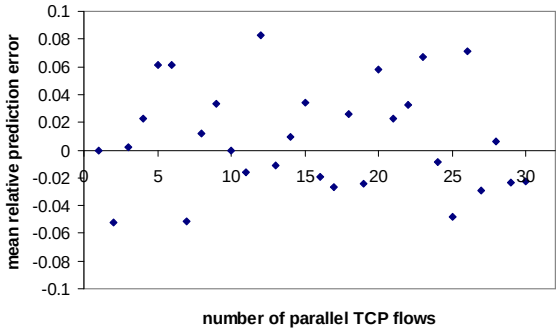


Figure 7.14: Relative prediction error for parallel TCP throughput as a function of number of parallel TCP flows.

Our predictor is relatively insensitive to the particular level of parallelism for the probes. Figure 7.13 shows the mean relative error for our predictor using $(1, 8)$, $(1, 10)$ and $(1, 15)$ parallel probes. We can see that we obtain similar performance in all cases. Of course, it is important not to use parallelism levels that are too close together (such as $(1, 2)$), as such probes are very sensitive to small fluctuations in the network or the existing cross traffic.

As it can be seen from Figure 7.14, the mean relative error for a given number of parallel TCP flows is not related to the number of parallel TCP flows. The figure, a scatter plot of the mean relative error (across all 41 paths) versus the number of parallel TCP flows, shows no clear trend. The correlation coefficient R between the mean relative prediction error and the number of parallel TCP flows is < 0.1.

Our experimental results have shown how, using the model derived in this section, one can effectively predict the throughput of parallel TCP for a wide range of parallelism relying only on two active probes at different levels of parallelism. In the following we try to estimate the effect of parallel TCP in the existing cross traffic for a given level of parallelism, the last "piece" necessary to make the `TameParallelTCP()` call possible.

7.5 Taming parallel TCP

There are a number of considerable challenges when trying to estimate the effect on cross traffic with an online system running on the end points:

1. The available bandwidth on the bottleneck link(s) is unknown.

2. The number of cross traffic flows and their loss rates and bandwidths on the bottleneck link(s) (the offered load) are unknown.

3. Making use of an additional network measurement tool (such as Pathload [122]) to determine the current load on the path is problematic since it can take a long time to converge. In addition, the measurement accuracy cannot be guaranteed. One would like to avoid any additional overhead beyond the required two active probes necessary to predict the throughput of parallel TCP flows.

In what follows, we make simplifying assumptions about the cross traffic's view of the shared links on the path in order to provide an estimate of impact on the cross traffic from the same two probes from which we derived the throughput curve in the previous section.

7.5.1 Algorithm

We assume that all TCP connections, including our parallel TCP flows and the cross traffic, share the same loss rate on a bottleneck link. This assumption is valid as long as one of the two following conditions can be satisfied:

1. The cross traffic has an RTT similar to our parallel TCP flows. In that case, all connections are very likely to have their congestion window synchronized,

and thus share the same loss rate. This fact has been independently verified by other research groups [201, 183, 184].

2. The router on the bottleneck link is using Random Early Detection (RED) [92] as its queue management policy, something that is becoming increasingly common. Research has demonstrated that with RED, different flows roughly experience the same loss rate (the RED rate, which depends on the queue occupancy) under steady state [92, 184].

Our approach to determining the effect of parallel TCP on cross traffic is based on our algorithm to estimate the parallel TCP throughput (Section 7.4). The key idea is to estimate $p_n \times RTT_n^2$ as a function of the number of parallel TCP flows. Based on the assumption that cross traffic shares the same loss rate as parallel TCP flows, we can then use the simple TCP throughput model (Equation 7.1) to estimate the relative change to the cross traffic throughput.

Recall in Section 7.4 that we model $p_n \times RTT_n^2$ with a partial order 2 polynomial function $a \times n^2 + b$ (Equation 7.6). After obtaining the two necessary measurements, we can calculate the value of a and b and are now able to estimate the loss rate as a function of the number of parallel TCP flows.

Relying on our assumptions, we have also obtained the loss rate of the cross traffic as a function of the number of parallel TCP flows n given there are m cross traffic flows (recall that m is relatively stable, see Section 7.4).

Thus, based on Equation 7.1, we can now estimate the relative change on each of the individual TCP throughputs without knowing m using the following equation:

$$relc = \frac{\frac{MSS \times C}{RTT_{n1} \times \sqrt{p_{n1}}} - \frac{MSS \times C}{RTT_{n2} \times \sqrt{p_{n2}}}}{\frac{MSS \times C}{RTT_{n1} \times \sqrt{p_{n1}}}} \quad (7.15)$$

$$= 1 - \sqrt{\frac{p_{n1}}{p_{n2}}} \quad (7.16)$$

$$= 1 - \sqrt{\frac{a \times {n_1}^2 + b}{a \times {n_2}^2 + b}} \quad (7.17)$$

Here, $relc$ is the relative throughput change for each flow. Equation 7.16 shows that all the flows share the same relative throughput change. MSS and C are constants as described in Section 7.4, and RTT_n is stable as was shown by Hacker, et al [108]. $\sqrt{\frac{p_{n1}}{p_{n2}}}$ can be estimated using Equation 7.6. Both a and b can be obtained with two probes as we discussed in Section 7.4. Note that n_1 and n_2 can be any parallelism levels. In practice, however, we are most interested in estimating the relative throughput change after adding in n_2 parallel TCP flows in comparison with adding in only one TCP flow, therefore n_1 equals 1 in this case.

In practice, we add another constraint to the `TameParallelTCP()` function to avoid the potential "diminishing returns" problem where more parallel TCP flows bring only marginal benefits. With the `TameParallelTCP()` function, we can estimate the aggregate throughput at any parallelism level. We then check to ensure that the performance gain is over an administrator-determined threshold after adding in an additional TCP flow. If the performance gain is below the threshold, we do not add more flows even when the impact on cross traffic is within the user's limit. This is important because we can avoid the system overhead and network overhead by avoiding unnecessary TCP flows.

Parallelism	Aggregate Throughput (Mbps)	Estimated Impact
1	89.7	0
2	89.85	0.499
3	89.88	0.666

Figure 7.15: Performance of parallel TCP on a 100 Mb LAN where the RTT is very small (about 0.2 0.4 ms). One TCP flow is the optimal in this case. Our tool accurately estimated it.

7.5.2 Evaluation

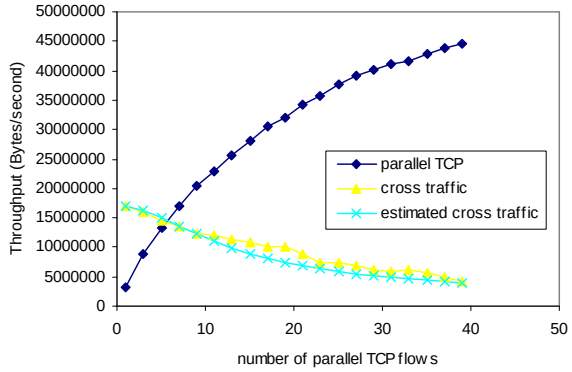
We have done a thorough ns2-based evaluation of our cross traffic estimator. The simulator allows us to analyze our estimator by controlling settings including bottleneck bandwidth and cross traffic characteristics.

Our simulation configuration was introduced in Section 7.3. We consider the same set of the scenarios presented there. As in Section 7.3, we employ Qiu et al's [183] simulation topology (Figure 7.2).

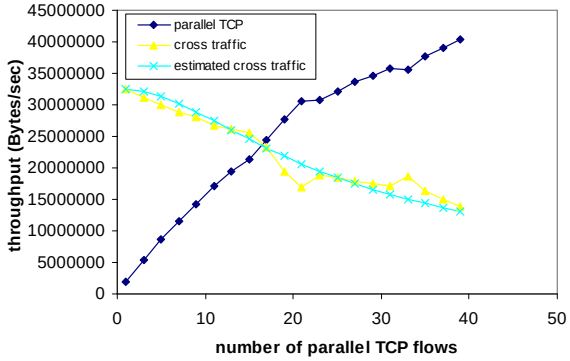
Figure 7.16 shows two examples, for Scenarios 4 and 6, of the performance of our estimator. In these cases we can accurately predict the impact on cross traffic as a function of the parallelism level using only two probes, the same probes we use to predict the throughput of the parallel flows as a function of parallelism level.

We summarize our prediction results as a CDF of the relative error in predicting the impact on cross traffic across all of our scenarios in Figure 7.17. We can see that 90% of predictions have relative prediction error less than 0.25. The cross traffic estimator is slightly biased. It conservatively predicts a greater impact on the cross traffic on average.

To further evaluate our cross traffic estimation algorithm, we designed a more complex topology with two groups of cross traffic. The topology and the simulation configuration is shown in Figure 7.18. Each simulation is 100 seconds long with cross



(a) Scenario 4 with 5 cross traffic



(b) Scenario 6 with 15 cross traffic

Figure 7.16: Cross traffic estimation examples.

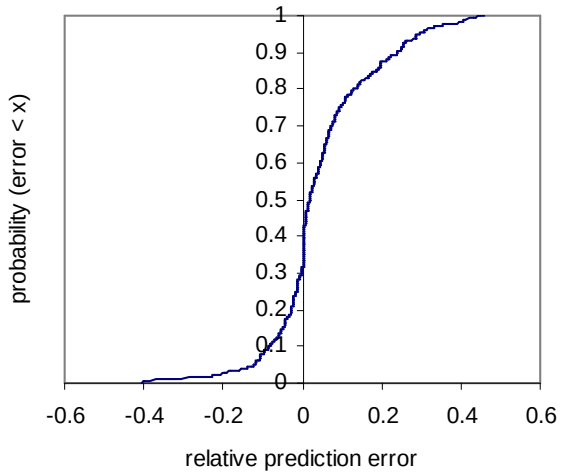


Figure 7.17: Cumulative distribution function of relative prediction error for cross traffic estimation for all the simulations with 6 scenarios as described in Figure 7.3.

traffic starting randomly between 0 and 8 seconds and all the parallel TCP flows starting at 10 seconds. L1 and L4 have latency 3 ms, L2 and L5 have latency 6 ms, L6 and L7 have latency 10 ms. L3 has latency 50m s and bottleneck bandwidth 1000 Mbit/s. N3 is using the RED queue management policy. Parallel TCP flows go from N2 to N6. Cross traffic group 1 goes from N7 to N8. Cross traffic group 2 goes from N1 to N5. We applied our estimation algorithm to these scenarios, resulting in Figure 7.19.

We also tested the cross traffic estimator for scenarios in which different TCP flows have different RTTs, and where RED is not used on the routers. Our estimator shows the right trend of the cross traffic throughput change, although accurate prediction cannot be guaranteed as flows with longer RTT tend to have higher loss rate than parallel TCP flows and vice versa. In essence, in situations in which cross traffic RTT and loss rate is unknown, our estimator is less accurate.

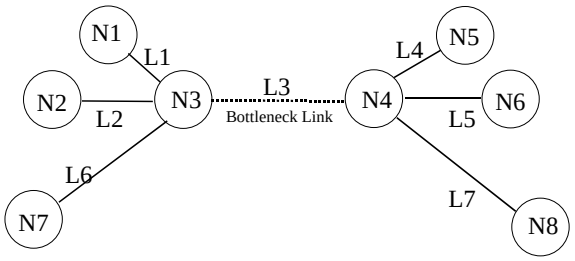


Figure 7.18: More complex topology for further evaluation of cross traffic estimation.

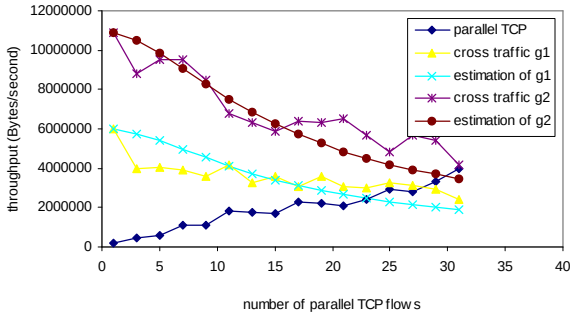


Figure 7.19: Estimation results with 14 TCP flows in cross traffic group 1 (g1) and 14 TCP flows in cross traffic group 2 (g2).

7.5.3 Outcome

We have demonstrated the feasibility of predicting the impact on cross traffic of a parallel TCP transfer as a function of the degree of parallelism. Under the assumption that all flows share the same loss rate, we can accurately predict the relative impact using the same two measurement probes used to predict the throughput of the parallel TCP transfer as a function of the degree of parallelism.

Combining these two predictions, we can implement the `TameParallelTCP()` API call:

1. Execute two probes at different parallelism levels.
2. Using the probe results, estimate the parallel TCP throughput as a function of the number of parallel TCP flows n using the techniques of the previous section.
3. Using the probe results, estimate the relative impact on cross traffic as a function of n using the techniques of this section.
4. Conduct a binary search on the cross traffic impact function, looking for the degree of parallelism, l, that has the largest impact less than that permitted in the API call.
5. Return l, and the impact and throughput predictions at parallelism l.

The cost of this implementation is dominated by executing the two probes.

7.6 Conclusions and future work

We have shown how to predict both parallel TCP throughput and its impact on cross traffic as a function of the degree of parallelism using only two probes at different

parallelism levels. Both predictions are monotonically changing with parallelism levels. Hence, the `TameParallelTCP()` function can be implemented using a simple binary search. To the best of our knowledge, our work is the first to provide a practical parallel TCP throughput prediction tool and to estimate the impact on the cross traffic.

We have made a few simplifying assumptions about the cross traffic in order to predict impact on it while having no knowledge of the actual cross traffic. While these assumptions are reasonable in many cases, we are now working on how to relax them. An implementation of a version of our `TameParallelTCP()` function is available from http://plab.cs.northwestern.edu/Clairvoyance/Tame.html.

Although the Internet paths show statistical stability, the transient stability won't hold over the long term. Either periodic resampling as in NWS [230] or the dynamic sampling rate adjustment algorithm from our other work [151] can be applied for the long term monitoring.

We have shown that parallel TCP flows can be used in a controlled manner, enhancing end-to-end throughput without significant influence on the background traffics. Therefore, our modeling technique can be used in the rate adaption for the data propagation among the RGIS servers.

Chapter 8

FatTree Based End-System Multicast

The Content Distribution Network (CDN) of RGIS is based on the publish/subscribe model as described in Chapter 2. For better scalability, we need to enhance the current prototype that relies on unicast to deliver updates to subscribers. Carzaniga, et al [54] showed that under many circumstances publish/subscribe systems can be built with end-system multicast techniques for better scalability and efficiency.

This chapter describes how to enhance the CDN of RGIS using FatNemo [41], a novel fat-tree based end-system multicast protocol designed by Birrer et al. at Northwestern University. I contributed to the design of the FatNemo protocol.

8.1 Introduction

As was described in Chapter 2, RGIS servers do not talk directly with each other, but indirectly via a content delivery network (CDN), which is based on the publish/subscribe model and is used solely to propagate updates to friendly RGIS servers.

In the current implementation, the CDN uses unicast to send updates to subscribers. This works fine with a small or medium size RGIS system, but may have scalability issues.

Carzaniga, et al [54] showed that publish/subscribe systems can be built with end-system multicast techniques for better scalability and efficiency. Multicast decouples the size of the receiver set from the amount of state kept at any single node and potentially avoids redundant communication in the network.

The FatNemo protocol can be adopted for the publish/subscribe model when a group of friendly RGIS servers subscribe to each other's updates. In such scenarios, each RGIS server participates into the distributed RGIS system and propagate its updates using the end-system multicast mechanism. The advantage of this approach is better scalability. This chapter first describes the FatNemo protocol and then discuss how it can be used to enhance the scalability of the CDN in RGIS.

The end-system multicast [64, 124, 97, 176, 58, 30, 57, 187, 241, 170, 218] was developed as an effective alternative to IP multicast, which is not widely deployed on the current Internet [70, 79]. In an end-system multicast approach participating peers organize themselves into an overlay topology for data delivery. Each edge in this topology corresponds to a unicast path between two end-systems or peers in the underlying Internet. All multicast-related functionality is implemented at the peers instead of at routers, and the goal of the multicast protocol is to construct and maintain an efficient overlay for data transmission.

Among the end-system multicast protocols proposed, tree-based systems have proven to be highly scalable and efficient in terms of physical link stress, state and control overhead, and end-to-end latency. However, normal tree structures have two inherent problems.

Resilience: They are highly dependent on the reliability of non-leaf nodes. Re-

silience is particularly relevant to the application-layer approach, as trees here are composed of autonomous, unpredictable end systems. The high degree of transiency of the hosts [1] has been pointed out as one of the main challenges for these architectures [37].

RGIS is designed for grids, which consist of large scale, dynamic computing resources. We therefore speculate that RGIS servers will show some degree of transiency, and it is important to design a highly resilient protocol.

Bandwidth limitations: They are likely to be bandwidth constrained [2] as bandwidth availability monotonically decreases as one descends into the tree. The bandwidth limitations of normal tree structures is particularly problematic for multi-source, bandwidth intensive applications. For a set of randomly placed sources in a tree, higher level paths (those closer to the root) will become the bottleneck and tend to dominate response times. Once these links become heavily loaded or overloaded, packets will start to be buffered or dropped.

RGIS is designed for large scale grid systems that can potentially scale to the size of the Internet. In such a huge distributed systems there could be thousands of RGIS servers, and we speculate that the updates sent among them would require significant bandwidth. Our assumption is supported by Smith [204], who discovered that the average workload on their centralized GIS is 8.8 operations per second, and the majority of the operations are updates to the server. This fact together with the potential large scale of the RGIS system indicate that the CDN has to be bandwidth efficient to gracefully handle the

[1] Measurement studies of widely used application-layer/peer-to-peer systems have reported median session times ranging from an hour to a minute [46, 106, 188, 62].

[2] The access link of an end system becomes its bandwidth bottleneck, thus we can model the bandwidth capacity as a property of the end-system.

updates propagation.

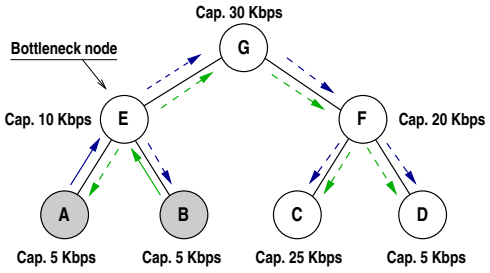
Birrer and Bustamante have addressed the resilience issue of tree-based systems in previous work [40] through the introduction of *co-leaders* and the reliance on *triggered negative acknowledgements* (NACKs). The FatNemo protocol addresses the bandwidth limitations of normal tree overlays.

FatNemo's approach capitalizes on Leiserson's seminal work on fat-trees [137]. Paraphrasing Leiserson, a fat-tree is like a real tree in that its branches become thicker the closer to the root, thus overcoming the "root bottleneck" of a regular tree. Figure 8.1 shows a schematic example of a binary fat-tree. FatNemo organizes participant end-systems in a tree that closely resembles a Leiserson fat-tree by dynamically placing higher degree nodes (nodes with higher bandwidth capacity) close to the root and increasing the cluster sizes as one ascends the tree.

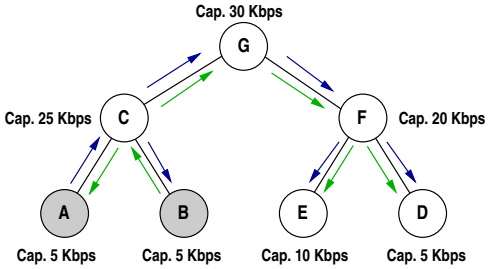
8.2 Related Work

Publish/subscribe communication is becoming the medium of choice for the design and development of loosely-coupled, component-based, distributed systems. Applications range from e-commerce frameworks (e.g., BEA's WebLogic) to peer-to-peer information sharing systems (e.g., Gnutella) to military command and control (e.g., US Air Force's Joint Battlespace Infosphere).

Publish/subscribe communication is characterized by the notion that senders and receivers are not required to have a priori knowledge of each other. The flow of information from senders to receivers is determined by the specific interests of the receiver rather than by an explicit destination address assigned by the sender. With this communication pattern, receivers subscribe to information that is of interest to them without regard to any specific source, while senders simply publish information



(a) Normal Tree



(b) Fat Tree

Figure 8.1: Two binary trees with nodes A and B as sources, publishing at 5 Kbps each. (a) shows a normal binary tree where node E becomes the bottleneck, resulting on a reduced (dash line) outgoing stream quality. Node E has to forward the stream A to node B and node G, as well as stream B to node A and node G, thus it needs an outgoing bandwidth capacity of 20 Kbps. However, it has only 10 Kbps available, making it a bottleneck in the tree. (b) shows a fat-tree with higher capacity nodes placed higher in the tree.

without addressing it to any specific destination. A publish/subscribe communication service is responsible for interpreting subscriptions and delivering the relevant publications to subscribers.

There are numerous work and commercial products on the publish/subscribe model [54, 86, 47, 216, 194]. The most related work is by Carzaniga [54], who concluded that under many circumstances the multicast approach can be adopted for higher scalability and efficiency.

The limited deployment of IP Multicast [70, 79] has led to considerable interest in alternate approaches that are implemented at the application layer, using only end-systems [64, 124, 97, 176, 58, 30, 57, 187, 241, 170, 218]. One of the first end-system multicast protocol was Narada [63], a multisource multicast system designed for small to medium sized multicast groups. Peers in Narada are organized into a mesh with fixed out-degree, with every peer monitoring all others to detect end-system failures and network partitions. The per-source multicast tree is built on top of this mesh from the reverse shortest path between each recipient and the source. Since the tree construction algorithm does not account for cross traffic, a powerful link is likely to be used by many multicast links, limiting the efficiency of the multicast system. FatNemo uses crew members to share the forwarding load, thus relaxing the burden on a single high bandwidth path. Overcast [124] organizes dedicated servers in a single-source, bandwidth optimized, multicast tree. In contrast, FatNemo is an end-system overlay that constructs a global optimized fat-tree for multisource multicast. Banerjee et al. [30] introduce Nice and demonstrate the effectiveness of overlay multicast across large scale networks. The authors also present the first look at the robustness of alternative overlay multicast protocols under group membership changes. FatNemo adopts the same implicit approach, and its design draws on a number of ideas from Nice such as its hierarchical control topology. FatNemo introduces co-leaders

to improve the resilience of the overlay and adopts a periodic probabilistic approach to reduce/avoid the cost of membership operations.

A new set of projects have started to address the resilience of overlay multicast protocols [32, 56, 218, 40, 170, 233]. ZigZag [218], a single-source protocol, explores the idea of splitting the control and data delivery task between two peers in each level, making both responsible for repairs under failures. With PRM [32] Banerjee et al. propose the use of probabilistic forwarding and NACK-based retransmission to improve resilience. In order to reduce the time-to-repair, Yang and Fei [233] argue for proactively, ahead of failures, selecting parent replacements. CoopNet [170] improves resilience by building several disjoint trees on a centralized organization protocol and employing Multiple Description Coding (MDC) for data redundancy. Nemo [40] and FatNemo build redundancy into the overlay through co-leaders; different from the previously described protocols, they make all crew members share forwarding responsibilities while all cluster members are in charge of repair operations. These simple measures enable an uninterrupted service to downstream peers especially during recovery intervals.

Aiming at bulk data distribution, protocols such as Splitstream [56] , Bittorrent [65] and Bullet [132] have proposed simultaneous data streaming over different paths to better share the forwarding load and increased downloading capacity. The methods differ in how they locate alternate streaming peers. In comparison, FatNemo exploits alternate paths for resilience and load balancing.

8.3 Fat-Trees and the Overlay

The communication network of a parallel machine must support global collective communication operations in which all processors participate [136]. These operations

have a wide range, including reduction and broadcast trees, and neighbor exchange. All-to-all personalized communication [115], in which each processor sends a unique message to every other processor, is key to many algorithms. To support such operations well, the network should have (1) minimal and scalable diameter, and (2) maximal and scalable bisection bandwidth. These goals are very similar to those of a multisource multicast overlay, which can be thought of as providing many-to-many personalized communication, a subset of all-to-all personalized communication.

In his seminal work on fat-trees [137], Leiserson introduced a universal routing network for interconnecting the processors of a parallel supercomputer, where communication can be scaled independently of the number of processors. Based on a complete binary tree, a fat-tree consists of a set of processors, located at the leaves, and interconnected by a number of switching nodes (internal to the tree) and edges. Each edge in the tree corresponds to two unidirectional *channels* connecting a parent with each of its children. Channel consist of a bundle of *wires*, and the number of wires in a channel is called its *capacity*. The capacity of the channels of a fat-tree grows as one goes up the tree from the leaves, thus allowing it to overcome the "root bottleneck" of a regular tree. Since their introduction, fat-trees have been successfully applied in massively parallel systems [138, 116] as well as in high performance cluster computing [120].

FatNemo organizes the end-systems participant in a multicast group, in an overlay tree that closely resembles a Leiserson fat-tree. In common with Leiserson, the goal is to minimize the mean and standard deviation of inter-node communication performance with multiple potential sources.

Emulating fat-trees in an overlay entails a number of challenges such as handling the high level of transiency of end-system populations and addressing their degree of heterogeneity. A straightforward way of approximating a fat-tree is placing those

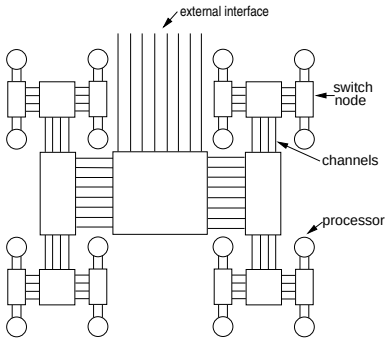


Figure 8.2: Leiserson's organization of a fat-tree in a supercomputer [137].

nodes with higher bandwidth capacity[3] closer to the root. Since interior nodes are involved in most inter-node communication paths they strongly influence the overall end-to-end delay and can soon become bottlenecks as one increases the number of sources. Figure 8.1 shows a schematic example of both a regular binary tree with two 5 Kbps sources (A and B) and a bottleneck node (E) unable to keep up with the publishing rate of the nodes downstream.

Available bandwidth can differ significantly from bandwidth capacity over time, due typically to competing traffic, and any algorithm that attempts to emulate fat-trees in an overlay needs to take account of such dynamism. Also, per-path characteristics must be taken into consideration. Since end-to-end latency is an important factor in the performance of interactive applications, the latency of each link in the overlay, the processing time at each node, and the number of intermediate nodes should be considered carefully. When selecting among possible parents, a closer node may be a better candidate, if it is able to support the load, than an alternative node offering higher available bandwidth. Finally, the mean time to failure of end-systems

[3]The maximum outgoing bandwidth that the node is capable of, the capacity of the IP link attaching the node to the network.

is significantly lower than for routers, possibly resulting in long interruptions to the data stream as the failure is discovered and the distribution tree repaired.

Although overlay fat-trees can be built above most tree-based multicast systems, FatNemo was implemented using Nemo, a high-resilience, latency-optimized overlay multicast protocol that was developed by Birrer and Bustamante [40]. The following section provides some background material on overlay multicast in general and on the operational details of Nemo, before describing the FatNemo design in Section 8.5.

8.4 Background

All peer-to-peer or application-layer multicast protocols organize the participating peers into (1) a control topology for group membership related tasks, and (2) a delivery tree for data forwarding. Available protocols can be classified according to the sequence adopted for their construction [29, 63]. In a tree-first approach [97, 124, 176], peers directly construct the data delivery tree by selecting their parents from among known peers. Additional links are later added to define, in combination with the data delivery tree, the control topology. With a mesh-first approach [63, 58], peers build a more densely connected graph (mesh) over which (reverse) shortest path spanning trees, rooted at any peer, can be constructed. Protocols adopting an implicit approach [30, 57, 187, 241, 218] create only a control topology among the participant peers. Their data delivery topology is implicitly determined by the defined set of packet-forwarding rules.

FatNemo builds on Nemo to emulate a fat-tree; thus, it inherits the latter's high scalability and resilience. In the following paragraphs, we provide a summarized description of Nemo; for more complete details we direct the reader to the work by Birrer et al. [40].

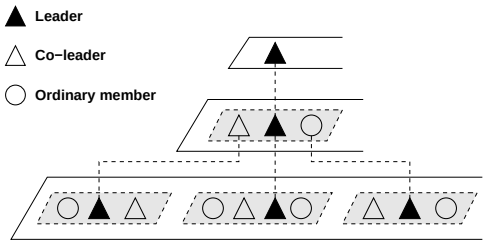


Figure 8.3: Nemo's logical organization. The shape illustrates only the role of a peer within a cluster: a leader of a cluster at a given layer can act as leader, co-leader, or an ordinary member at the next higher layer.

Nemo

Nemo [40] follows the implicit approach to building an overlay for multicasting. The set of communication peers are organized into clusters based on network proximity, where every peer is a member of a cluster at the lowest layer. Clusters vary in size between d and $3d-1$, where d is a constant known as the *degree*. Each of these clusters selects a *leader* that becomes a member of the immediately superior layer. In part to avoid the dependency on a single node, every cluster leader recruits a number of co-leaders to form a supporting crew. The process is repeated at each new layer, with all peers in a layer being grouped into clusters, crew members selected, and leaders promoted to participate in the next higher layer. Hence peers can lead more than one cluster in successive layers of this logical hierarchy. Figure 8.3 illustrates the logical organization of Nemo.

A new peer joins the multicast group by querying a well-known special end-system, the rendezvous point, for the identifier of the root node. Starting there and in an iterative manner, the incoming peer continues: (*i*) requesting the list of members at the current layer from the cluster's leader, (*ii*) selecting from among them who to contact next based on the result from a given cost function, and (*iii*)

moving into the next layer. When the new peer finds the leader with minimal cost at the bottom layer, it joins the associated cluster.

Member peers can leave Nemo in a graceful manner (e.g. user disconnects) or in an ungraceful manner (unannounced, e.g. when the end-system crashes). For graceful departures, since a common member has no responsibilities towards other peers, it can simply leave the group after informing its cluster's leader. On the other hand, a leader must first elect replacement leaders for all clusters it owns before it leaves the session.

To detect unannounced departures, Nemo relies on heartbeats exchanged among the cluster's peers. Unreported members are given a fixed time interval before being considered dead, at which point a repair algorithm is initiated. If the failed peer happens to be a leader, the tree itself must be fixed, the members of the victim's cluster must elect the replacement leader from among themselves.

To deal with dynamic changes in the underlying network, every peer periodically checks the leaders of the next higher layers and switches clusters if another leader has a lower cost (i.e. lower latency) than the current one. Additionally, in a continuous process of refinement, every leader checks its highest owned cluster for better suited leaders and transfers leadership if such a peer exists.

Nemo addresses the resilience issue of tree-based systems through the introduction of co-leaders. Co-leaders improve the resilience of the multicast group by avoiding dependencies on single nodes and providing alternative paths for data forwarding. In addition, crew members share the load from message forwarding, thus improving scalability.

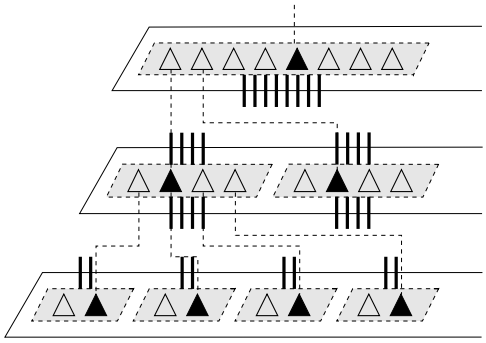


Figure 8.4: FatNemo's Topology. The figure illustrates how the tree gets fatter when moving toward the root. This tree has a cluster degree of 2.

8.5 FatNemo Design

To build an overlay fat-tree, FatNemo relies on three heuristics: (1) higher bandwidth degree nodes should be placed higher up in the tree, (2) all peers must serve as crew members in order to maximize load balancing, and (3) the size of clusters should increase exponentially as one ascends the tree. The following paragraphs provide the motivations behind each of these heuristics.

The per-node bandwidth constraint is critical for bandwidth-demanding applications and can be stated as the number of full-rate streams a peer is able to support, i.e. its *out-degree*. By organizing peers based on their out-degrees [225, 63], FatNemo intends to reduce the bandwidth constraints of links higher up the tree. Since the process of estimating available bandwidth is time consuming, peers initially join the tree based on proximity. Once in the tree, every leader checks its highest owned cluster for better suited leaders in terms of bandwidth availability, and transfers leadership if such a peer exists. This process assures that high out-degree peers will gradually ascend to higher layers, thus transforming the tree into a bandwidth optimized fat-tree.

In traditional fat-trees the number of wires increases as one ascends the tree. Considering an overlay unicast connection as a "wire", the number of wires in FatNemo increases together with the crew size as one moves toward the root – the maximum possible number of wires is thus achieved by setting the crew size equal to the cluster size. The size of a cluster at layer i in FatNemo varies between d_i and $2d_i + 2$, and grows exponentially ($d_i = d_0^{i+1}$) as we move up the layers. The 0-th layer contains the leaf nodes and has a degree d_0 of 3 (same as Nice and Nemo). The increased number of wires helps avoid higher level links from becoming the bottleneck of the system, as alternate paths share the load and reduce the forward responsibility of each peer.

Beyond increasing the number of wires, large crew sizes also help reduce the depth of a tree (when compared with its constant cluster-sizes equivalent). A smaller depths means a lower total number of end-system hops, and should translate in a reduction on the end-to-end delay.

Figure 8.4 illustrates how FatNemo constructs a fat-tree. In this simple example $d_0 = 2$, so clusters scale up by a factor of 2 ascending the tree. Notice that these links/wires are indeed $(d_{i+1} \ldots 2d_{i+1} + 2)$ to $(d_i \ldots 2d_i + 2)$ relations, as every crew member of the next higher layer will talk to the crew members of the immediately lower layer. For clarity in the graph, this set of links is represented in the graph by d_i lines.

To better understand the positive effect of FatNemo's heuristics, we show an instantiation of them with a population of 20,000 peers from a popular on-line game. [4]

To begin, let's generalize the concept of *out-degree*. The out-degree of a peer, d_{out}, is equivalent to the total forwarding responsibility of a node, and it can be stated

[4]The number corresponds to the active populations of players in *hattrick.org*, an online soccer game.

Table 8.1: Cluster and Crew Size as a function of the cluster degree d, for a 20,000 peer population. The variable x is a place holder for the cluster index starting at 0 for the lowest layer.

Protocol	Cluster Size $k(x)$		Crew Size $c(x)$
Nice	$d \ldots 3d-1$	$d = 3:3 \ldots 8$	1
Nemo	$d \ldots 3d-1$	$d = 3:3 \ldots 8$	3
FatNemo	$d^{x+1} \ldots 2d^{x+1}+2$	$d = 3, x = 1:9 \ldots 20$	$k(x)$

as a function of the number of layers L in which the node participates, the cluster size $k(x)$ and the crew size $c(x)$ at layer x (Equation (8.1)). Table 8.1 illustrates the parameter values for FatNemo and two alternative protocols.

$$d_{out} = \sum_{x=0}^{L-1} \frac{k(x)-1}{c(x)} \tag{8.1}$$

Nice and Nemo have, in expectation, 5.5 nodes per cluster at every layer. Using this value as an approximation for the cluster size, a traditional tree for this population size will be about 7 layers in depth. FatNemo, on the other hand, has a variable expected number of nodes per cluster, and its expected tree depth is 4 layers.

Based on the expected depth of the different trees for this example population, the out-degree requirements on their root nodes can be calculated. According to the generalized out-degree equation introduced in the previous paragraph (Equation (8.1)), Nice requires a root out-degree of 31.5, or almost three times more than what is needed from a Nemo's root (10.5) with a crew size of 3. In other words, the root of a traditional tree for a 20,000 peer population must support 31.5 times the source rate to fulfill its forwarding responsibility! By emulating a fat-tree in the overlay, FatNemo avoids this "root bottleneck" requiring root out-degree only of 3.7.

8.6 Evaluation

The performance of FatNemo is evaluated through simulation, and the performance metrics are:

Response Time: End-to-end delay (including retransmission time) from the source to the receivers, as seen by the application. This includes path latencies along the overlay hops, as well as queuing delay and processing overhead at peers along the path. A lower mean response time indicates a higher system responsiveness, and a smaller standard deviation implies better synchronization among the receivers.

Delivered Packets: Number of packets successfully delivered to all subscribers within a fixed time window. It indirectly measures the protocol's ability to avoid bottlenecks in the delivery tree.

Delivery Ratio: Ratio of subscribers that have received a packet within a fixed time window. Disabled receivers are not accounted for.

Duplicate Packets: Number of duplicate packets per sequence number, for all enabled receivers, reflecting an unnecessary burden on the network. Packets arrived outside of the delivery window are accounted for as duplicates, since the receiver already assumed them as lost.

Control-Related Traffic: Total control traffic in the system, in mega bits per second (Mbps); part of the system's overhead. A network packet traversing four routers (including the source and destination node) will account as three sent packets, one for every router which has to forward it.

Figure 8.5: Three simulation scenarios: Low-, Medium- and High-Bandwidth. Bandwidth is expressed in Kbps.

Scenario	Routers	End systems	Links	Client-Stub	Stub-Stub	Transit-Stub	Transit-Transit
Low-B/W	510	5000	11240	400-6000	3000-8000	4000-10000	10000-20000
Medium-B/W	312	6000	12730	800-8000	4000-10000	6000-15000	15000-30000
High-B/W	615	7500	16450	1000-15000	10000-30000	10000-50000	50000-100000

8.6.1 Experimental Setup

The simulations were done using the packet-level, event-based simulator SPANS [40]. Simulations were run using GridG [144, 146] topologies with 5510, 6312 and 8115 nodes, and a multicast group of 256 members. GridG leverages Tiers [81, 51] to generate a three-tier hierarchical network structure, before it applies a power law enforcing algorithm that retains the hierarchical structure.

Members were randomly associated with end systems, and a random delay of between 0.1 and 80 ms was assigned to every link. The links use drop-tail queues with a buffer capacity of 0.5 sec. GridG was configured to use different bandwidth distributions for different link types [132]. The assumption is that the core of the Internet has higher bandwidth capacities than the edge, as shown in Fig. 8.5. In all three scenarios, the bandwidth has a uniform distribution with ranges shown in Fig. 8.5.

Each simulation experiment lasted for 500 sec. (simulation time). All peers join the multicast group by contacting the rendezvous point at uniformly distributed, random times within the first 100 sec. of the simulation. A warm-up time of 200 sec. is omitted from the figures. Publishers join the network and start publishing at the beginning of the simulation. Starting at 200 sec. and lasting for about 300 sec., each simulation has a phase with membership changes. During this phase, each protocol was exercised with and without host failures. Failure rates are set based on those obtained from a published report of field failures for networked systems [232]. Nodes fail independently at a time sampled from an exponential distribution (with *mean*

time to failure (MTTF) equal to 60 min.) and rejoin shortly after (time sampled from an exponential distribution with *mean time to repair* (MTTR) equal to 10 min.). The two means were chosen asymmetrically to allow, on average, 6/7 of all members to be up during this phase. The failure event sequence was generated a priori based on the above distribution and used for all protocols and all runs.

In all experiments, multi-source multicast streams were modeled to a group. Each source sends constant bit rate (CBR) traffic of 1000 Byte payload at a rate of 10 packets per second. The buffer size was set to 16 packets, which corresponds to the usage of a 1.6-second buffer, a realistic scenario for applications such as video conferencing.

8.7 Experimental Results

This section presents early evaluation results of FatNemo and compares them with those of three alternative protocols. The reported results are from five runs per protocol obtained with the different GridG topologies and the Low-Bandwidth scenario. Similar results were obtained with the Mid- and High-Bandwidth scenarios.

Figure 8.6 shows the average number of delivered packets of all runs with no host failures. The protocol's data delivery topology collapses as too many publishers are added. This happens first for Narada, which is unable to handle the full publishing rate from one publisher. Nice and Nice PRM handle an increasing number of publishers better; however, they deliver substantially fewer packets when compared with FatNemo. FatNemo is best at avoiding bottlenecks in the delivery tree, delivering the most packets when the network is overloaded (as seen with 8 publishers).

The performance of a multi-source multicast system can be measured in terms of mean and standard deviation of the response time. Table 8.2 shows these two

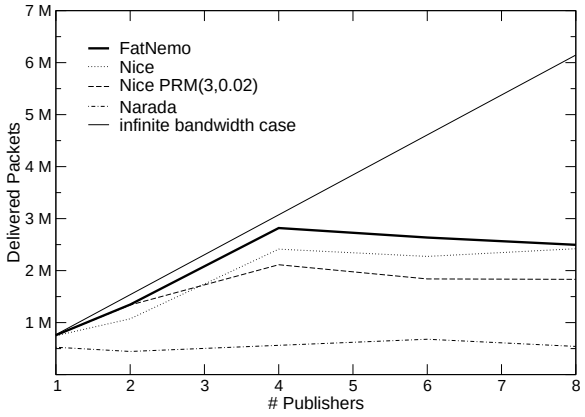


Figure 8.6: Delivered packets (256 end hosts, Low-Bandwidth scenario).

Table 8.2: Response Time (1 Publisher, 256 end hosts, Low-Bandwidth scenario).

Protocol	Mean	Std
FatNemo	0.158	0.073
Nice	0.183	0.082
Nice-PRM(3,0.02)	0.195	0.086
Narada	0.770	0.464

metrics for the evaluated protocols with one publisher. FatNemo outperforms Nice, Nice PRM and Narada in terms of mean and standard deviation of response time. With an increased number of publishers the relative number of delivered packets for Nice, Nice PRM and Narada decreases compared to FatNemo, which makes it impossible to fairly compare the response time for more than one publisher based only on one number. The problem stems from that fact that, when lowering its delivery ratio a protocol will drop those packets with high response time more likely than others. Thus, comparing response times across protocols with significantly different delivery ratios under stress will give less resilient protocols an unfair advantage.

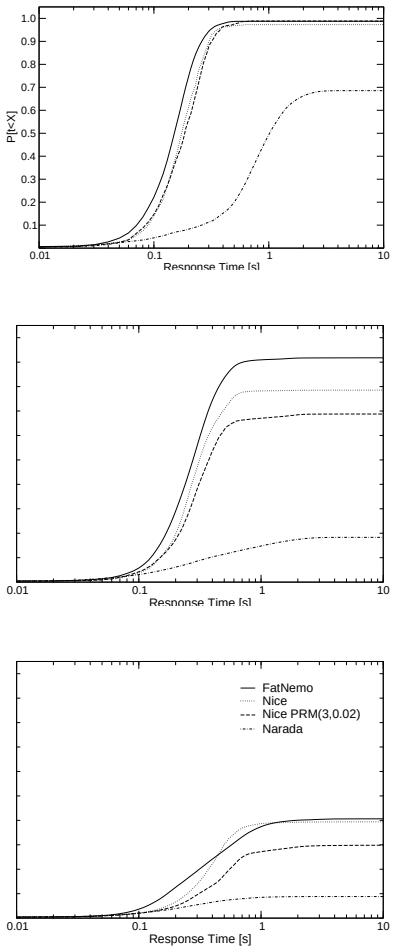


Figure 8.7: Response Time CDF with 1, 4 and 8 publishers (figures ordered top-down; 256 end hosts, Low-Bandwidth scenario).

Table 8.3: Delivery Ratio (1 Publisher, 256 end hosts, Low-Bandwidth scenario).

Protocol	No Failures	With Failures
FatNemo	0.987	0.966
Nice	0.973	0.956
Nice-PRM(3,0.02)	0.989	0.970
Narada	0.685	0.648

Figure 8.7.(a) shows the Cumulative Distribution Function (CDF) of the response time per packet for one publisher. The y-axis is normalized to the infinite bandwidth case, i.e. when all receivers receive all possible packets. FatNemo, Nice and Nice PRM perform well, but FatNemo's flatter tree results in an improved response time. Narada is only able to deliver a fraction of all possible packets, and only then with a substantially high delay. With increasing number of publishers, the protocols start running into bottlenecks. Despite the harder conditions, FatNemo is able to outperforms the alternative protocols in terms of packet delivery times as illustrated in Fig. 8.7.(b) and Fig. 8.7.(c).

Table 8.3 shows the delivery ratio using one publisher with and without end-systems failures. It can be seen that FatNemo performs as well as Nice PRM under a low-failure rate scenario with only a drop of 2.1% in delivery ratio. Nice has a slightly lower delivery ratio, while Narada suffers already from a collapsed delivery tree with only about 70% delivery ratio. In general, the delivery ratio will decrease as the number of publishers increases, as the protocol's data delivery topology slowly collapses.

The overhead of a protocol can be measured in terms of duplicate packets. This metric is shown in the second column of Table 8.4. Despite its high delivery ratio, FatNemo incurs, in average, only 0.367 duplicate packets per sequence number, while

Table 8.4: Overhead (1 Publisher, 256 end hosts, Low-Bandwidth scenario).

Protocol	Duplicate packets	Control Traffic [Mbps]
FatNemo	0.367	17.99
Nice	0.000	9.211
Nice-PRM(3,0.02)	5.168	11.48
Narada	0.006	157.3

Nice-PRM suffers from 5.168 duplicate packets per sequence number generated by its probabilistic forwarding algorithm. Nice and Narada feature almost no duplicates, but at a high cost in term of delivery ratio as shown in Table 8.3. FatNemo's control related traffic is higher than for Nice and Nice-PRM, a result of its larger cluster cardinality higher in the tree. The control traffic is accounted for at router level, thus the choice of a peer's neighbors in FatNemo also adds additional overhead, as it opts not for the closest, but for the peer with highest bandwidth.

8.8 Protocol modifications required

FatNemo and Nemo [40] were originally designed for real time multimedia, and its triggered negative acknowledgement (NACK) assumes continuous data flow. Although in a potentially huge grid system, the CDN is likely to have continuous data traffic, we need to handle situations where continuous data flow doesn't exist. We need to either disable the NACK feature or use alternative mechanisms.

Simply disabling the NACK could solve this problem at the expense of potentially decreasing the resilience of the FatNemo protocol. Fortunately, RGIS only requires loose consistency among the servers, which indicates that disabling NACK may be a feasible solution.

Alternatively, a hop-to-hop ACK mechanism can solve this problem without affecting the resilience, but the overhead of this approach requires more investigation.

8.9 Conclusions and Further Work

This chapter first described FatNemo, a novel scalable end-system multicast protocol that emulates a fat-tree on the wide area network to build data delivery topologies with minimized mean and standard deviation of the response time, then discussed how to enhance the RGIS CDN using FatNemo. FatNemo is based on Nemo, a resilient end-system multicast protocol [40]. Simulation results show that FatNemo can achieve significantly higher delivery ratios than alternative protocols (an increase of up to 360% under high load), while reducing the mean (by up to 80%) and standard deviation (by up to 84%) of the response time in the non-overloaded case. Under a heavy load and a realistic host failure rate, the resulting protocol is able to attain high delivery ratios with negligible cost in terms of control-related traffic. To adopt the FatNemo protocol for the CDN in RGIS, the NACK feature should be disabled or we can use alternative mechanisms such as hop-to-hop ACK.

Chapter 9

Conclusions and Future Work

9.1 Summary

A GIS built on the relational data model and modern relational database systems allows the application to ask complex and sophisticated questions about compositions of resources. We have described the architecture of our RGIS relational grid information service system, followed by the description of the building components of the RGIS system including query rewriting, topology generation, scheduling with inaccurate job size information, serial and parallel TCP throughput modeling and monitoring, and end-system multicast.

The dissertation started with GridG, our synthetic grids topology generator and annotator. GridG can generate network topologies that follow the power-laws of the Internet topology yet also keeping the clear hierarchy of the Internet. GridG can also sensibly annotate the topology from the network to the hosts level, considering both intro- and inter- hosts correlations.

The dissertation then described our query rewriting techniques for fast user response time, which trade off between the running time of a query and the number

of results it returns. We show that tradeoffs over many orders of magnitude are possible, and the techniques can also be used to keep query processing time largely independent of query complexity, albeit returning fewer results with more complex queries. RGIS uses the techniques and a watchdog to bound the running time of queries.

Our work on the size-based scheduling policies first shows the performance of SRPT and FSP with inaccurate job size information and then applied them to web server scheduling and P2P server side scheduling. Our work has shown that it is possible to apply size-based schedulers on the RGIS servers.

Our work on the modeling and monitoring of sequential/parallel TCP throughput is important for distributed systems developer and researchers. Also, these techniques can be applied in the building of the CDN on which the RGIS servers rely to propagate its updates.

Our work on multicast proposed and evaluated a novel end-system multicast protocol based on the idea of fat-trees. The current CDN within RGIS is based on subscribe/publish model, which can use either unicast or multicast as its data transferring mechanism. Our fat-tree based multicast protocol can be adopted by the CDN of RGIS as it has significantly higher performance and better scalability than unicast in most cases.

As a conclusion, our work demonstrates for the first time that it is feasible to provide the flexibility and power of a relational data model in a distributed GIS system while controlling the costs associated with it. Several major components for the system are also described in detail.

9.2 Integration of the RGIS system

Our next step in the development of the RGIS system is to integrate all the available research components and software modules into the current prototyped system.

9.3 Future work

Dynamic data is an important component of computational grids. Although our current implementation of the RGIS system has considered dynamic contents in the schema design, we are not entirely clear how to handle them efficiently in a scalable manner. Some dynamic contents (such as CPU load) are transient by nature, which makes the overhead of keeping all the relevant RGIS server consistent too high. Our strategy is to utilize other monitoring tools such as ReMoS [141], RPS [78] and DualPats [152] to help clients get dynamic data on the computing resources in a real time manner. More work is needed to explore this interesting topic.

Scheduling the queries and updates on the RGIS servers for Quality of Service is another interesting research topic. Although this thesis has addressed the scheduling problem for better performance, the scheduling component is not targeted at QoS.

Bibliography

[1] The apache software foundation. http://www.apache.org/.

[2] Bonnie, a unix file system benchmark. http://www.textuality.com/bonnie/.

[3] http://codeen.cs.princeton.edu.

[4] http://dast.nlanr.net/projects/iperf/.

[5] http://www.isi.edu/nsnam/ns/.

[6] http://www.planet-lab.org.

[7] The internet traffic archive. http://ita.ee.lbl.gov/.

[8] The ircache project. http://www.ircache.net/.

[9] Kazaa homepage. http://www.kazaa.com.

[10] The squid web proxy cache project. http://www.squid-cache.org/.

[11] Bittorrent homepage, 2004. http://bitconjurer.org/BitTorrent.

[12] eDonkey homepage, 2004. http://www.edonkey2000.com.

[13] Gnutella homepage, 2004. http://www.gnutella.com.

[14] Kazaa homepage, 2004. http://www.kazaa.com.

[15] Mutella homepage, 2004. http://mutella.sourceforge.net.

[16] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley. The design and implementation of an intentional naming system. In *Proceedings of the 17th ACM Symposium on Operating System Principles*, December 199.

[17] K. Aida, A. Takefusa, H. Nakada, S. Matsuoka, S. Sekiguchi, and U. Nagashima. Performance evaluation model for scheduling in a global computing system. *International Journal of High Performance Computing Applications*, 14(3), 2000.

[18] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *ACM Symposium on Theory of Computing*, 2000.

[19] Reka Albert and Albert laszlo Barabasi. Statistical mechanics of complex networks. *Reviews of modern physics*, 74, 2002.

[20] Paul Albitz and Cricket Liu. *DNS and BIND.* O'Reilly and Associates, Inc., Sebastopol, California, 1992.

[21] W. Allcock, J. Bester, J. Bresnahan, A. Cervenak, L. Liming, and S. Tuecke. GridFTP: Protocol extensions to ftp for the grid. Technical report, Argonne National Laboratory, August 2001.

[22] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in highperformance computational grid environments. *Parallel Computing*, 28, 2002.

[23] Arnold O. Allen. *Probability, statistics, and queueing theory with computer science applications.* Academic press, Inc., 1990.

[24] M. Allen and R. Wolski. The livny and plank-beck problems: Studies in data movement on the computational grid. In *Supercomputing 2003*, November 2003.

[25] J. Almeida, M. Dabu, A. Manikutty, and P. Cao. Providing differentiated quality-of-service in web hosting services. In *Proc. 1st WISP*, 1998.

[26] Eitan Altman, Konstantin Avrachenkov, and Chadi Barakat. A stochastic model of TCP/IP with stationary random. In *ACM SIGCOMM*, pages 231–242, 2000.

[27] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, Mark Stemm, and Randy H. Katz. TCP behavior of a busy internet server: Analysis and improvements. In *INFOCOM (1)*, pages 252–262, 1998.

[28] Hari Balakrishnan, Srinivasan Seshan, Mark Stemm, and Randy H. Katz. Analyzing Stability in Wide-Area Network Performance. In *ACM SIGMETRICS*, June 1997.

[29] Suman Banerjee and Bobby Bhattacharjee. A comparative study of application layer multicast protocols, 2002. Submitted for review.

[30] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast. In *Proc. of ACM SIGCOMM*, August 2002.

[31] Suman Banerjee, Seungjoon Lee, Bobby Bhattacharjee, and Aravind Srinivasan. Resilient multicast overlays. In *Sigmetrics 2003*, 2003.

[32] Suman Banerjee, Seungjoon Lee, Bobby Bhattacharjee, and Aravind Srinivasan. Resilient multicast using overlays. In *Proc. of ACM SIGMETRICS*, June 2003.

[33] Nikhil Bansal and Mor Harchol-Balter. Analysis of SRPT scheduling: investigating unfairness. In *Proceedings of SIGMETRICS/Performance*, pages 279–290, 2001.

[34] Nikhil Bansal and Mor Harchol-Balter. Analysis of SRPT scheduling: Investigating unfairness. In *Proc. ACM SIGMETRICS*, 2001.

[35] A.L. Barabasi and R.Albert. Emergence of scaling in random networks. *Science*, pages 509–512, 1999.

[36] Paul Barford and Mark Crovella. Measuring web performance in the wide area. *Performance Evaluation Review*, 27(2):37–48, 1999.

[37] Mayank Bawa, Hrishikesh Deshpande, and Hector Garcia-Molina. Transience of peers & streaming media. In *Proc. of HotNets-I*, October 2002.

[38] Daniel S. Bernstein, Zhengzhu Feng, Brian Neil Levine, and Shlomo Zilberstein. Adaptive peer selection. In *Proc. 2nd IPTPS*, 2003.

[39] P. Bernstein and N. Goodman. The failure and recovery problem for replicated distributed databases. *ACM Transactions on Database Systems*, December 1984.

[40] Stefan Birrer and Fabián E. Bustamante. Nemo - resilient peer-to-peer multicast without the cost. In *Proceedings of the 12th Annual Multimedia Computing and Networking Conference (MMCN'05)*.

[41] Stefan Birrer, Dong Lu, Fabian Bustamante, Yi Qiao, and Peter Dinda. Fatnemo: Building a resilient multisource multicast fat-tree. In *Proceedings of the 9th Web Content Caching and Distribution Workshop (WCW)*, October 2004.

[42] Jurg Bolliger, Thomas Gross, and Urs Hengartner. Bandwidth modeling for network-aware applications. In *INFOCOM (3)*, pages 1300–1309, 1999.

[43] Jurg Bolliger, Thomas Gross, and Urs Hengartner. Bandwidth modeling for network-aware applications. In *INFOCOM (3)*, pages 1300–1309, 1999.

[44] O.J. Boxma and J.W. Cohen. Heavy-traffic analysis for the G/G/1 queue with heavy-tailed distributions. *Queueing Systems*, 33:177–204, 1999.

[45] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[46] Fabián E. Bustamante and Yi Qiao. Friendships that last: Peer lifespan and its role in P2P protocols. In *Proc. of IWCW*, October 2003.

[47] Fabian E. Bustamante, Patrick Widener, and Karsten Schwan. Scalable directory services using proactivity. In *Proceedings. of Supercomputing 2002 (SC 2002)*, 2002.

[48] Werner Bux. Analysis of a local-area bus system with controlled access. *IEEE Transactions on Computers*, 32(8):760–763, 1983.

[49] R. Buyya and M. Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, May 2002 (to appear).

[50] Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–168, June 1997.

[51] Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997.

[52] Marne C. Cario and Barry L. Nelson. Numerical Methods for Fitting and Simulating Autoregressive-to-Anything Processes. *INFORMS Journal on Computing*, 10(1):72–81, 1998.

[53] Robert Carter and Mark Crovella. Measuring bottleneck link speed in packet-switched networks. *Performance Evaluation*, (28):297–318, 1996.

[54] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3), 2001.

[55] H. Casanova. Simgrid: A toolkit for the simulation of application scheduling. In *the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, 2001.

[56] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *Proc. of the 19th ACM SOSP*, October 2003.

[57] Miguel Castro, Antony Rowstron, Anne-Marie Kermarrec, and Peter Druschel. SCRIBE: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communication*, 20(8), 2002.

[58] Yatin Chawathe. *Scattercast: an architecture for Internet broadcast distribution as an infrastructure service*. Ph.D. Thesis, U. of California, Berkeley, CA, Fall 2000.

[59] A. Chervenak, V. Vellanki, and Z. Kurmas. Protecting file systems: A survey of backup techniques. In *Proceedings of the Joint NASA and IEEE Mass Storage Conference*, 1998.

[60] Bill Cheswick, Hal Burch, and Steve Branigan. Mapping and visualizing the internet. In *Proceedings of the USENIX Annual Technical Conference*, June 2000.

[61] Bilal Chinoy. Dynamics of internet routing information. In *SIGCOMM*, pages 45–52, 1993.

[62] Yang-Hua Chu, Aditya Ganjam, T. S. Eugene Ng, Sanjay G. Rao, Kunwadee Sripanidkulchai, Jibin Zhan, and Hui Zhang. Early experience with an Internet broadcast system based on overlay multicast. In *Proc. of USENIX ATC*, June 2004.

[63] Yang-Hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communication*, 20(8), October 2002.

[64] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *Proc. of ACM SIGMETRICS*, June 2000.

[65] Bram Cohen. BitTorrent. bitconjurer.org/BitTorrent/, 2001. File distribution.

[66] Mark Crovella, Robert Frangioso, and Mor Harchol-Balter. Connection scheduling in web servers. In *Proc. USENIX USITS*, 1999.

[67] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of HPDC 2001*, August 2001.

[68] J. da Silva and O. Gudmundsson. The amanda network backup system manager. In *Proceedings of the USENIX Systems Administration conference*, 1993.

[69] J. da Silva, O. Gudmundsson, and D. Mosse. Performance of a parallel network backup manager. In *Proceedings of USENIX*, pages 17–26, 1992.

[70] Stephen E. Deering. Multicast routing in internetworks and extended LANs. In *Proc. of ACM SIGCOMM*, August 1988.

[71] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Sixth Symposium on Principles Of Distributed Computing*, 1987.

[72] Shuang Deng. Empirical model of WWW document arivals at access links. In *IEEE International Conference on Communication*, June 1996.

[73] P. Dinda and D. O'Hallaron. An evaluation of linear models for host load prediction. In *8th IEEE International Symposium on High Performance Distributed Computing (HPDC-8)*, 1999.

[74] P. Dinda and B. Plale. A unified relational approach to grid information services. *Grid Forum Informational Draft GWD-GIS-012-1*, February 2001.

[75] Peter A. Dinda. The statistical properties of host load. *Scientific Programming*, 7(3,4), 1999. A version of this paper is also available as CMU Technical Report CMU-CS-TR-98-175. A much earlier version appears in LCR '98 and as CMU-CS-TR-98-143.

[76] Peter A. Dinda. Online prediction of the running time of tasks. *Cluster Computing*, 5(3), 2002.

[77] Peter A. Dinda and Dong Lu. Nondeterministic queries in a relational grid information service. In *Proceedings of ACM/IEEE SC 2003 (Supercomputing 2003)*, 2003. To Appear. (In this volume.).

[78] Peter A. Dinda and David R. O'Hallaron. An extensible toolkit for resource prediction in distributed systems. Technical Report CMU-CS-99-138, School of Computer Science, Carnegie Mellon University, July 1999.

[79] Christopher Diot, Brian N. Levine, Bryan Lyles, Hassan Kassem, and Doug Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14(1), January/February 2000.

[80] Matthew B. Doar. A better model for generating test networks. *IEEE GLOBECOM*, 1996.

[81] Matthew B. Doar. A better model for generating test networks. In *Proc. of Globecom*, November 1996.

[82] Michael Doar. A better model for generating test networks. In *Proceedings of GLOBECOM '96*, November 1996.

[83] Benoit Donnet, Philippe Raoult, Timur Friedman, and Mark Crovella. Efficient algorithms for large-scale topology discovery. In *Proceedings of the 2005 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, June 2005. To Appear.

[84] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. What do packet dispersion techniques measure? In *INFOCOM*, pages 905–914, 2001.

[85] Allen B. Downey. Using pathchar to estimate internet link characteristics. In *Measurement and Modeling of Computer Systems*, pages 222–223, 1999.

[86] John Dunagan, Nicholas J.A. Harvey, Michael B. Jones, Dejan Kostic, and Marvin Theimerand Alec Wolman. Fuse: Lightweight guaranteed distributed failure notification. In *Proceedings of the Sixth Symposium on Operating Systems Design and Implementation (OSDI)*, 2004.

[87] Kevin Fall and Sally Floyd. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *Computer Communication Review*, 26(3):5–21, July 1996.

[88] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *Proceedings of SIGCOMM '99*, 1999.

[89] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[90] Renato Figueiredo, Peter A. Dinda, and Jose Fortes. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003)*, May 2003. To Appear.

[91] Steve Fisher. Relational model for information and monitoring. Technical Report Informational Draft GWD-GP-7-1, Grid Forum, 2001.

[92] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[93] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Grid services for distributed system integration. *Computer*, 35(6), 2002.

[94] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15, 2001.

[95] Ian Foster. Globus web page. Technical Report http://www.mcs.anl.gov/globus, Argone National Laboratory.

[96] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[97] Paul Francis. Yoid: Extending the Internet multicast architecture. http://www.aciri.org/yoid, April 2000.

[98] Eric J. Friedman and Shane G. Henderson. Fairness and efficiency in web server protocols. In *Proceedings of SIGMETRICS/Performance*, 2003.

[99] V. Fuller, T. Li, J. Yu, and K. Varadhan. (rfc1519) Classless Inter-Domain Routing (CIDR): an address assignment and aggregation strategy, September 1993. http://www.faqs.org/rfcs/rfc1519.txt.

[100] Fyodor. Remote os detection via tcp/ip stack fingerprinting. (web page). http://www.insecure.org/nmap/nmap-fingerprinting-article.html.

[101] Simson Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly and Assocates, 1994.

[102] Global Grid Forum. Global grid forum web site. http://www.gridforum.org.

[103] Mingwei Gong and Carey Williamson. Quantifying the properties of srpt scheduling. In *Proceedings of IEEE MASCOTS*, 2003.

[104] Mingwei Gong and Carey Williamson. Simulation evaluation of hybrid srpt scheduling policies. In *Proceedings of IEEE MASCOTS*, 2004.

[105] M. Goyal, R. Guerin, and R. Rajan. Predicting tcp throughput from non-invasive network sampling. In *IEEE INFOCOM*, 2002.

[106] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling and analysis of a peer-to-peer file-sharing workload. In *Proc. of ACM SOSP*, December 2003.

[107] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proc. 19th ACM SOSP*, 2003.

[108] T. Hacker, B. Athey, and B. Noble. The end-to-end performance effects of parallel tcp sockets on a lossy wide-area network. In *16th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS)*, 2002.

[109] Thomas J. Hacker, Brian D. Noble, and Brian D.Athey. The effects of systemic packet loss on aggregate tcp flows. In *IEEE/ACM Supercomputing*, 2002.

[110] Thomas J. Hacker, Brian D. Noble, and Brian D.Athey. Improving throughput and maintaining fairness using parallel TCP. In *IEEE Infocom*, 2004.

[111] Mor Harchol-Balter, Mark E. Crovella, and SungSim Park. The case for srpt scheduling in web servers. Technical Report MIT-LCR-TR-767, MIT lab for computer science, October 1998.

[112] Mor Harchol-Balter, Bianca Schroeder, Nikhil Bansal, and Mukesh Agrawal. Size-based scheduling to improve web performance. *ACM Transactions on Computer Systems (TOCS)*, 21(2), May 2003.

[113] Mor Harchol-Balter, Karl Sigman, and Adam Wierman. Asymptotic convergence of scheduling policies with respect to slowdown. *Performance Evaluation*, 49(1/4), 2002.

[114] R. Hinden. (rfc1517) Applicability statement for the implementation of Classes Inter-Domain Routing (CIDR), September 1993. http://www.faqs.org/rfcs/rfc1517.txt.

[115] Susan Hinrichs, Corey Kosak, David O'Hallaron, Thomas Stricker, and Riichiro Take. An architecture for optimimal all-to-all personalized communication. In *Proceedings of the 6th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 310–319, 1994.

[116] Mark Homewood and Moray McLaren. Meiko CS-2 interconnect elan – elite design. In *IEEE Hot Interconnects Symposium*, August 1993.

[117] Ningning Hu and Peter Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling*, 21(6), August 2003.

[118] B. Huffaker, Daniel Plummer, David Moore, and K. Claffy. Topology discovery by active probing. In *Proceedings of Symposium on Applications and the Internet*, January 2002.

[119] IBM International Technical Support Organization. *Understanding LDAP*. IBM Corporation, 1998.

[120] InfiniBand Trade Association. Infiniband architecture specification (1.0.a). www.infinibandta.com, June 2001.

[121] International Telecommunication Union. Information technology – open systems interconnection – the directory: Overview of concepts, models, and services, August 1997.

[122] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodolody, dynamics, and relation with tcp throughput. In *ACM SIGCOMM*, 2002.

[123] M. Jain and C. Dovrolis. Pathload: A measurement tool for end-to-end available bandwidth. In *Passive and Active Measurement Workshop*, 2002.

[124] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole Jr. Overcast: Reliable multicasting with and overlay network. In *Proc. of the 4th USENIX OSDI*, October 2000.

[125] Cheng Jin, Qian Chen, and Sugih Jamin. Inet: Internet topology generator. Technical Report CSE-TR443-00, Department of EECS, University of Michigan Ann Arbor, 2000.

[126] G. Jin and B. Tierney. Netest: A tool to measure maximum burst size, available bandwidth and achievable throughput. In *International Conference on Information Technology*, 2003.

[127] G. Jin, G. Yang, B. Crowley, and D. Agarwal. Network characterization service (ncs). In *10th IEEE Symposium on High Performance Distributed Computing, Aug. 2001.*, 2001.

[128] Guojun Jin and Brian L. Tierney. System capability effects on algorithms for network bandwidth measurement. In *ACM SIGCOMM conference on Internet measurement*, 2003.

[129] S. Keshav. A control-theoretic approach to flow control. *Proceedings of the conference on Communications architecture and protocols*, pages 3–15, 1993.

[130] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. In *Thirteenth ACM Symposium on Operating Systems Principles*, volume 25, pages 213–225, Asilomar Conference Center, Pacific Grove, U.S., 1991. ACM Press.

[131] Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The web as a graph: Measurements, models and methods. *Lecture Notes in Computer Science*, 1627:1–18, 1999.

[132] Dejan Kostić, Adolfo Rodriguez adn Jeannie Albrecht, and Amin Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proc. of the 19th ACM SOSP*, October 2003.

[133] B. Krishnamurthy and J. Rexford. *Web Protocols and Practice: HTTP1.1, Networking Protocols, Caching, and Traffic Measurements.* Addison-Wesley, 2001.

[134] Kevin Lai and Mary Baker. Nettimer: A tool for measuring bottleneck link bandwidth. In *USENIX Symposium on Internet Technologies and Systems*, pages 123–134, 2001.

[135] Butler W. Lampson. Designing a global name service. In *4th ACM Symposium on Principles of Distributed Computing*, August 1986.

[136] Thomas Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann, 1992.

[137] Charles E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, 34(10):892–901, October 1985.

[138] Charles E. Leiserson, Zahi S. Abuhamdeh, David C. Douglas, Carl R. Feynman, Mahesh N. Ganmukhi, Jeffrey V. Hill, W. Daniel Hillis, Bradley C. Kuszmaul, Margaret A. St Pierre, David S. Wells, Monica C. Wong-Chan, Shaw-Wen Yang, and Robert Zak. The network architecture of the Connection Machine CM-5. *Journal of Parallel and Distributed Computing*, 33(2):145–158, 1996.

[139] Chuang Liu and Ian Foster. A constraint language approach to grid resource selection. Technical Report TR-2003-07, Department of Computer Science, University of Chicago, March 2003.

[140] X. Liu and A. Chien. Realistic large-scale online network simulation. 2004.

[141] Bruce Lowekamp, Nancy Miller, Dean Sutherland, Thomas Gross, Peter Steenkiste, and Jaspal Subhlok. A resource monitoring system for network-aware applications. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 189–196. IEEE, July 1998.

[142] Bruce Lowekamp, David R. O'Hallaron, and Thomas R. Gross. Topology discovery for large ethernet networks. In *Proceedings of SIGCOMM 2001*, August 2001.

[143] Dong Lu and Peter Dinda. Gridg: Generating realistic computational grids. *Performance Evaluation Review*, 30(4):33–40, 2003.

[144] Dong Lu and Peter A. Dinda. GridG: Generating realistic computational grids. *ACM Sigmetrics Performance Evaluation Review*, 30(4):33–41, March 2003.

[145] Dong Lu and Peter A. Dinda. Synthesizing realistic computational grids. In *Proceedings of ACM/IEEE SC 2003 (Supercomputing)*, November 2003.

[146] Dong Lu and Peter A. Dinda. Synthesizing realistic computational grids. In *Proc. of SC2003*, November 2003.

[147] Dong Lu, Peter A. Dinda, and Jason A. Skicewicz. Scoped and approximate queries in a relational grid information service. In *Proceedings of the 4th International Workshop on Grid Computing (Grid 2003)*, November 2003.

[148] Dong Lu, Yi Qiao, P. Dinda, and F. Bustamente. Modeling and taming parallel tcp on the wide area network. Technical Report NWU-CS-04-35, Northwestern University, Computer Science Department, April 2004.

[149] Dong Lu, Yi Qiao, Peter Dinda, and Fabian Bustamante. Characterizing and predicting tcp throughput on the wide area network. Technical Report NWU-CS-04-34, Northwestern University, Computer Science Department, April 2004.

[150] Dong Lu, Yi Qiao, Peter Dinda, and Fabian Bustamante. Modeling and taming parallel tcp on the wide area network. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 05)*, April 2005.

[151] Dong Lu, Yi Qiao, Peter A. Dinda, and Fabian E. Bustamante. Characterizing and predicting tcp throughput on the wide area network. Technical Report NWU-CS-04-34, Northwestern University, Department of Computer Science, April 2004.

[152] Dong Lu, Yi Qiao, Peter A. Dinda, and Fabian E. Bustamante. Characterizing and predicting tcp throughput on the wide area network. In *ICDCS*, 2005.

[153] Dong Lu, H. Sheng, and P. Dinda. Effects and implications of file size/service time correlation on web server scheduling policies. Technical Report NWU-CS-04-33, Northwestern University, Computer Science Department, April 2004.

[154] Dong Lu, Huanyuan Sheng, and Peter Dinda. Size-based scheduling policies with inaccurate scheduling information. In *Proceedings of IEEE MASCOTS*, 2004.

[155] Dong Lu, Huanyuan Sheng, and Peter A. Dinda. Effects and implications of file size/service time correlation on web server scheduling policies. Technical Report NWU-CS-04-33, Northwestern University, Department of Computer Science, 2004.

[156] Dong Lu, Huanyuan Sheng, and Peter A. Dinda. Size-based scheduling policies with inaccutate scheduling information. In *Proc. 12th Annual Meeting of the IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2004.

[157] Stephen Manley and Margo Seltzer. Web Facts and Fantasy. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS97)*, Monterey, CA, 1997.

[158] M. Mathis and M. Allman. A framework for defining empirical bulk transfer capacity metrics, rfc3148, July 2001.

[159] M. Mathis, J Heffner, and R Reddy. Web100: Extended tcp instrumentation for research, education and diagnosis. *ACM Computer Communications Review*, 33(3), July 2003.

[160] Matthew Mathis, Jeffrey Semke, and Jamshid Mahdavi. The macroscopic behavior of the tcp congestionavoidance algorithm. *Computer Communication Review*, 27(3), 1997.

[161] Alberto Medina, Anukool Lakhina, Ibrahim Matta, John Byers famedina, and anukool. Brite: An approach to universal topology generation. In *IEEE MASCOTS '01 (Tools track)*, 2001.

[162] Alberto Medina, Ibrahim Matta, and John Byers. On the origin of power laws in internet topologies. *ACM SIGCOMM Computer Communication Review (CCR)*, 30:18–28, 2000.

[163] Milena Mihail and Christos Papadimitriou. On the eigenvalue power law. *Springer-Verlag Lecture Notes in Computer Science*, 2002.

[164] J. Mogul and S. Deering. A framework for defining empirical bulk transfer capacity metrics, rfc3148, November 1990.

[165] R. Morris. TCP behavior with many flows. In *ICNP*, pages 205–211, 1997.

[166] Matt W. Mutka and Miron Livny. The available capacity of a privately owned workstation environment. *Performance Evaluation*, 12(4):269–284, July 1991.

[167] Andy Myers, Peter A. Dinda, and Hui Zhang. Performance characteristics of mirror servers on the internet. In *INFOCOM (1)*, pages 304–312, 1999.

[168] Object Management Group. The common object request broker: Architecture and specification (version 2.3.1). Technical report, Object Management Group, 1999.

[169] Jiter Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling tcp throughput: A simple model and its empirical validation. In *ACM SIGCOMM*, 1998.

[170] Venkata N. Padmanabhan, Helen J. Wang, and Philip A. Chou. Resilient peer-to-peer streaming. In *Proc. of IEEE ICNP*, 2003.

[171] Ventata N. Padmanabhan and Kunwadee Sripanidkulchai. The case for cooperative networking. In *IPTPS*, 2002.

[172] Christopher R. Palmer and J. Gregory Steffan. Generating network topologies that obey power laws. In *GLOBECOM '2000*, 2000.

[173] Vern Paxson. End-to-end routing behavior in the Internet. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, volume 26,4 of *ACM SIGCOMM Computer Communication Review*, pages 25–38, New York, August 1996. ACM Press.

[174] Vern Paxson. End-to-end routing behavior in the Internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, 1997.

[175] Vern Paxson and Sally Floyd. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[176] Dimitrios Pendarakis, Sherlia Shi, Dinesh Verma, and Marcel Waldvogel. ALMI: An application level multicast infrastructure. In *Proc. of USENIX USITS*, March 2001.

[177] Ranjit Perera. The variance of delay time in queueing system M/G/1 with optimal strategy SRPT. *Archiv fur Elektronik und Uebertragungstechnik*, 47(2):110–114, 1993.

[178] Karin Petersen, Mike J. Spreitzer, Douglas B. Terry, Marvin M. Theimer, and Alan J. Demers. Flexible update propagation for weakly consistent replication. In *Proceedings of the 16th ACM Symposium on Operating SystemsPrinciples (SOSP-16), Saint Malo, France*, 1997.

[179] Beth Plale, Peter Dinda, and Gregor von Laszewski. Key concepts and services of a grid information service. In *Proceedings of the 15th International Conference on Parallel and Distributed Computing Systmes (PDCS 2002)*, 2002.

[180] Beth Plale, Craig Jacobs, Charlie Moad, Rupali Parab, and Prajakta Vaidya. Synthetic database benchmark/workload for grid information servers. In *Proceedings of the 4th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2004)*, 2004.

[181] Yi Qiao, Dong Lu, F. Bustamante, and P. Dinda. Looking at the server side of peer-to-peer systems. Technical Report NWU-CS-04-37, Department of Computer Science, Northwestern University, March 2004.

[182] Yi Qiao, Dong Lu, Fabian Bustamante, and Peter Dinda. Looking at the server side of peer-to-peer systems. In *7th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR 2004)*, 2004.

[183] Lili Qiu, Yin Zhang, and Srinivasan Keshav. On individual and aggregate TCP performance. In *ICNP*, pages 203–212, 1999.

[184] Lili Qiu, Yin Zhang, and Srinivasan Keshav. Understanding the performance of many TCP flows. *Computer Networks*, 37(3–4):277–306, 2001.

[185] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC '98),*, pages 140–146, July 1998.

[186] Rajesh Raman, Miron Livny, and Marvin Solomon. Resource management through multilateral matchmaking. In *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC 2000),*, pages 290–291, July 2000.

[187] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Application-level multicast using content-addressable networks. In *Proc. of NGC*, November 2001.

[188] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a DHT. In *Proc. of USENIX ATC*, December 2004.

[189] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *Passive and Active Measurement Workshop*, 2003.

[190] Vinay Ribeiro, Mark Coates, Rudolf Riedi, Shriram Sarvotham, Brent Hendricks, and Richard Baraniuk. Multifractal cross-traffic estimation, 2000.

[191] Matei Ripeanu and Ian Foster. Mapping gnutella network: Macroscopic propersies of large-scale peer-to-peer systems. In *1st International Workshop on Peer-to-Peer Systems*, 2002.

[192] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[193] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Middleware*, pages 329–350, 2001.

[194] RTI Inc. Rti inc. web site. http://www.rti.com.

[195] Stefan Saroiu, Krishna P. Gummadi, Richard J. Dunn, Steven D. Gribble, and Henry M. Levy. An analysis of internet content delivery systems. In *Proc. 5th USENIX OSDI*, 2002.

[196] L. E. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16:678–690, 1968.

[197] L. E. Schrage and L. W. Miller. The queue M/G/1 with the shortest remaining processing time discipline. *Operations Research*, 14:670–684, 1966.

[198] F. Schreiber. Properties and applications of the optimal queueing strategy srpt - a survey. *Archiv fur Elektronik und Uebertragungstechnik*, 47:372–378, 1993.

[199] B. Schroeder and M. Harchol-Balter. Web servers under overload: How scheduling can help. Technical Report CMU-CS-02-143, Carnegie Mellon School of Computer Science, June 2002.

[200] Srinivasan Seshan, Mark Stemm, and Randy H. Katz. SPAND: Shared passive network performance discovery. In *USENIX Symposium on Internet Technologies and Systems*, 1997.

[201] S. Shenker, L. Zhang, and D. Clark. Some observations on the dynamics of a congestion control algorithm. *ACM Computer Communication Review*, 1990.

[202] Harimath Sivakumar, Stuart Bailey, and Robert L. Grossman. PSockets: The case for application-level network striping for data intensive applications using high speed wide area networks. In *Supercomputing*, 2000.

[203] F. Donelson Smith, Felix Hernandez-Campos, Kevin Jeffay, and David Ott. What TCP/IP protocol headers can tell us about the web. In *SIGMETRICS/Performance*, pages 245–256, 2001.

[204] Warren Smith, Abdul Waheed, David Meyers, and Jerry C. Yan. An evaluation of alternative designs for a grid information service. *Cluster Computing*, 4:29–37, 2001.

[205] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, Kenjiro Taura, and Andrew A. Chien. The microgrid: a scientific tool for modeling computational grids. In *Supercomputing*, 2000.

[206] N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rocketfuel. In *Proceedings of ACM/SIGCOMM*, August 2002.

[207] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM 2001*, pages 149–160, 2001.

[208] Jacob Strauss, Dina Katabi, and Frans Kaashoek. A measurement study of available bandwidth estimation tools. In *Internet Measurement Conference*, 2003.

[209] Martin Swany and Rich Wolski. Multivariate resource performance forecasting in the network weather service. In *ACM/IEEE conference on Supercomputing*, 2002.

[210] Martin Swany and Rich Wolski. Representing dynamic performance information in grid environments with the network weather service. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, 2002.

[211] A. N. Tabet and D. D. Kouvatsos. On the approximation of the mean response times of priority classes in a stable G/G/C/PR queue. *Journal of the Operational Research Society*, 43:227–239, 1992.

[212] A.S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 1995.

[213] Hongsuda Tangmunarunkit, Ramesh Govindan, and Sugih Jamin. Network topology generators: degree-based vs. structural. In *Proceedings of SIGCOMM '02*, 2002.

[214] The Open Group. *DCE 1.2.2: Introduction to OSF DCE*. The Open Group, September 1997. http://www.opengroup.org/pubs/catalog/f201.htm.

[215] Marvin Theimer and Michael B. Jones. Overlook: Scalable name service on an overlay network. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS 2002)*, July 2002.

[216] Tibco Inc. Tibco inc. web site. http://www.tibco.com.

[217] B. Tierney. Tcp tuning guide for distributed application on wide area networks. *USENIX and SAGE Login*, 26(1), 2001.

[218] Duc A. Tran, Kien A. Hua, and Tai Do. ZIGZAG: An efficient peer-to-peer scheme for media streaming. In *Proc. of IEEE INFOCOM*, April 2003.

[219] Transaction Processing Council. Tpc benchmarks. http://www.tpc.org.

[220] Amin Vahdat, Michael Dahlin, Thomas Anderson, and Amit Aggarwal. Active names: flexible location and transport of wide-area resources. In *USENIX Symposium on Internet Technology and Systems*, October 1999.

[221] Sudharshan Vazhkudai and Jennifer Schopf. Predicting sporadic grid data transfers. In *12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*, 2002.

[222] Sudharshan Vazhkudai, Jennifer Schopf, and Ian Foster. Predicting the performance of wide area data transfers. In *The 16th Int'l Parallel and Distributed Processing Symposium (IPDPS 2002).*, 2002.

[223] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service location protocol. Internet RFC 2165, June 1997.

[224] Jim Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, 1999.

[225] Zheng Wang and Jon Crowcroft. Bandwidth-delay based routing algorithms. In *Proc. of IEEE GlobeCom*, November 1995.

[226] B.M. Waxman. Routing of multipoint connections. *IEEE J. of Selected Areas in Communications*, 6(9):1622–1671, 1988.

[227] D. Wessels and K. Claffy. (rfc2186) Internet cache protocol (ICP), version 2, September 1997. http://www.faqs.org/rfcs/rfc2186.html.

[228] Duane Wessels and K Claffy. ICP and the Squid Web cache. *IEEE Journal on Selected Areas in Communication*, 16(3):345–357, 1998.

[229] Jared Winick and Sugih Jamin. Inet-3.0: Internet topology generator. Technical Report CSE-TR-456-02, Department of EECS, University of Michigan Ann Arbor, 2002.

[230] Richard Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1(1):119–132, 1998.

[231] Richard Wolski, Neil Spring, and Jim Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6):757–768, 1999.

[232] Jun Xu, Zbigniew Kalbarczyk, and Ravishankar K. Iyer. Networked Windows NT system field failure data analysis. In *Proc. of PRDC*, December 1999.

[233] Mengkun Yang and Zongming Fei. A proactive approach to reconstructing overlay multicast trees. In *Proc. of IEEE INFOCOM*, March 2004.

[234] T. Ylonen. SSH — secure login connections over the internet. In *Proceedings of the 6th USENIX Security Symposium*, pages 37–42, 1996.

[235] Haifeng Yu and Amin Vahdat. Design and evaluation of a continuous consistency model for replicated services. In *Fourth Symposium on Operating Systems Design and Implementation (OSDI)*, pages 305–318, 2001.

[236] Marcia Zangrilli and Bruce B. Lowekamp. Comparing passive network monitoring of grid application traffic with active probes. In *Fourth International Workshop on Grid Computing*, 2003.

[237] Xuehai Zhang, Jeffrey L. Freschl, and Jennifer M. Schopf. A performance study of monitoring and information services for distributed systems. In *Proceedings of the 12th International Symposium on High Performance Distributed Computing (HPDC)*, 2003.

[238] Y. Zhang, N. Du, e Paxson, and S. Shenker. On the Constancy of Internet path properties. In *ACM SIGCOMM Internet Measurement Workshop*, 2001.

[239] Yin Zhang, Lee Breslau, Vern Paxson, and Scott Shenker. On the Characteristics and Origins of Internet flow rates. In *ACM SIGCOMM*, 2002.

[240] Yin Zhang, Nick Duffield, Vern Paxson, and Scott Shenker. On the constancy of internet path properties. In *ACM SIGCOMM Internet Measurement Workshop*, November 2001.

[241] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John D. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proc. of NOSSDAV*, June 2001.

Appendix A

Domain-Based Scheduling on Web Servers

A.1 Introduction

In a web server, requests continuously arrive to be serviced. A request requires a certain service time to be completed, time whose components include the CPU, the disk, and the network path. A request is queued when it arrives and remains in the system until it is complete, the total time from arrival to completion being the sojourn time or response time. Scheduling policies determine which requests in the queue are serviced at any point in time, how much time is spent on each, and what happens when a new request arrives. Common goals of the scheduling policy are to minimize the mean sojourn time (response time of the request), the average slowdown (the ratio of its response time to its size), and to behave fairly to all requests.

Many policies are possible. First Come First Served (FCFS) is a non-preemptive policy in which the requests are run to completion in the order in which they were received. A more common policy is Processor Sharing (PS), which is preemptive. In

PS all requests in the queue are given an equal share of the web server's attention. Generalized Processor Sharing (GPS) generalizes PS with priorities. Often, FCFS can be combined with PS or GPS, with FCFS dispatching of requests from the queue to a pool of processes or threads that are collectively scheduled using PS or GPS. These polices ignore the service time of the request.

Recently, size-based scheduling policies, those that incorporate the service time of the request into their decisions, have been proposed for use in web servers. Harchol-Balter, et al, have proposed the use of the Shortest Remaining Processing Time (SRPT) scheduling policy in web servers [33, 112], showed how to incorporate it into actual implementations [112], and proved that the performance gains of SRPT usually do not come at the expense of large jobs [33]. In other words, SRPT is fair with heavy-tailed job size distributions. Gong, et al further investigated the fairness issues of SRPT through simulation [103] and proposed two hybrid SRPT scheduling policies [104] to trade off the fairness with performance. The Fair Sojourn Protocol (FSP) is a modified version of SRPT that has been proven to be more efficient and fair than PS given any arrival sequence and service time distribution [98].

In the *implementation* of size-based polices such as SRPT and FSP on a web server, the service time of the request is needed. The common assumption is that the service time is the size of the file being served, as this is very easy to discover when the request enters the system. More broadly, the assumption is that the service time is strongly correlated to the file size. In this appendix, we examine the validity of this assumption, and the impact that the degree of correlation between file size and service time has on the performance of SRPT and FSP.

To evaluate this impact, we developed a simulator that can support PS, SRPT, and FSP in both M/G/1/m and G/G/n/m. The simulator operates on a trace of request arrivals, which can come either from an augmented Apache [1] web server

log, or from a trace generator. The trace contains the request arrivals, the file sizes, and the actual service times in microseconds. We use traces that we have captured on our department-level web server, and traces captured by others on web caches.

In our earlier work [154] on size-based scheduling policies with inaccurate job size information, we showed that for mean response time and slowdown, the performance of SRPT-FS and FSP-FS is dramatically affected by R, falling below that of PS for low R values; and an effective job size estimator is needed to successfully apply size-based scheduling plicies. This appendix focues on applying size-basd scheduling in web servers when the correlation R is low. We first study how the performance of the file-size based policies (SRPT-FS, FSP-FS) diverges from their ideal versions (SRPT, FSP) as we increase the load on the web server. We then propose domain-based scheduling and evaluate it via trace-driven simulations.

We study G/G/n/m in addition to M/G/1/m because previous research [72, 175] has shown that Poisson processes are valid only for modeling the arrival of user-initiated TCP sessions such as the arrival of TELNET connections and FTP connections. HTTP arrivals are not Poisson. That is because HTTP document transmissions are not entirely initiated by the user: the HTTP client will automatically generate a series of additional requests to download embedded files, thus resulting in a more bursty process. Previous work [72] pointed out that the aggregated interarrival times of HTTP requests can be modeled with a heavy-tailed Weibull distribution.

There has been significant work on the G/G/n queuing model. However, we are aware of no analytical results on G/G/n/m for SRPT or FSP scheduling in regimes where interarrival times and service times are heavy-tailed. Therefore, the work we describe in this appendix is based on measurement and simulation.

Using our infrastructure, and measured and synthesized trace data, we address the following questions:

1. What is the actual degree of correlation between file size and service time in practice? (Section A.2)

2. What is the performance of SRPT, FSP and PS under typical real workloads? (Section A.3)

3. Is there a simple and low-overhead estimator for service time that would make SRPT and FSP on M/G/1/m and G/G/n/m perform better? (Section A.4)

It is important to point out that our results in addressing questions 2 and 3 are largely independent of our results for question 1, and the algorithm we develop in response to question 4 provides benefits to SRPT and FSP over a wide range of possible answers to question 1.

Our definition of service time is the time needed to send all of requested data in the absence of other requests in the system. Our measurements show that the assumption that file size and service time are strongly correlated is unwarranted—the correlation is, in fact, often rather weak. We believe that the reason for this phenomenon is *path diversity* to different clients. Even if for every specific request, the service time $t_s = L + N/B$, where N is the number of bytes in the transfer and L and B are the latency and bandwidth of the path, every path will likely have a different L and B. In aggregate, this will mean that t_s will be weakly correlated with N. Notice that this explanation does not require that the bottleneck for file transfer be the network. Path diversity is simply a fact of life of a large network.

Our trace-driven simulations show that the performance of file size-based SRPT and FSP is strongly affected by the weak correlation between file size and service time reflected in our web server traces. In fact, R is indeed low enough that both file-size based SRPT and FSP perform worse than PS. We believe that the job size distribution, arrival process and load decide the threshhold value of R that SRTP

Scheduling Policy	Description
PS	Processor Sharing scheduling policy.
FSP	Ideal Fair Sojourn Protocol, service times are known exactly a priori.
SRPT	Ideal Shortest Remaining Processing Time, service times are known exactly a priori.
FSP-FS	File size-based Fair Sojourn Protocol, file sizes are used as service times.
SRPT-FS	File size-based Shortest Remaining Processing Time, file sizes are used as service times.
FSP-D	Domain-based Fair Sojourn Protocol, estimated service times are used as service times.
SRPT-D	Domain-based Shortest Remaining Processing Time, estimated service times are used as service times.

Figure A.1: Scheduling policies used in the appendix.

and FSP need to outperform PS.

These results led us to believe that a better estimator for service time was needed. We refer to our estimator as a domain estimator, and the use of our domain-based estimator with a size-based scheduling policy such as SRPT or FSP as *domain-based scheduling*. The basic idea is to use the high order k bits of the source IP address to assign the request to one of 2^k domains. For each domain, we estimate the service rate (file size divided by service time) based on all previous completed transfers to the domain. The service rate is then used to estimate the service time of a new request based on its file's size. Based on our traces, there is a strong relationship between the correlation of these estimates and the actual service time, which grows with the number of bits k used. In short, by choosing k appropriately, we can create enough correlation to make SRPT and FSP perform well. Surprisingly, k can be kept relatively small, making the implementation of domain-based scheduling feasible and fast. Throughout the appendix, we refer to the scheduling policies as listed in Figure A.1, and refer to the queuing models used as listed in Figure A.2.

Queuing Model	Description
$M/G/1/m$	Poisson arrival process; General service time distribution; Single server ; Limited queue capacity m.
$G/G/n/m$	General arrival process (Pareto and Weibull); General service time distribution; n servers ; Limited queue capacity m.

Figure A.2: Queuing models used in the appendix. Both Pareto and Weibull service time distributions are considered.

A.2 Is file size a good indicator of service time?

Size-based SRPT scheduling appeared in digital communication networks research in 1983 [48]. In this context, the service time was taken to be equal to the transmission time of a message, which is proportional to the length of the message stored in the node buffers. A web server serving static requests appears superficially similar in that it transmits files to the client. However, there are differences. First, in the digital communication network context, the work represented by the service time is pushing the bits of the message onto the wire, while for the web server context, the work involves end-to-end cooperation along an entire shared heterogeneous path. Although most transfers are likely to be dominated by the bottleneck bandwidth in the path and the latency of the path, there are multiple possible bottlenecks along the path and they can vary with time due to packet losses and congestion. Second, the disk(s), memory system(s), and CPU(s) of the web server and the client are also potential bottlenecks. These complexities suggest that the service time of a request may not be proportional or even well correlated with the size of the file it serves.

There are several possible definitions for service time in the web server context. For example, we could focus on a bottleneck resource on the server, such as the CPU, and define the service time as the total CPU time needed to execute the request.

Alternatively, we could treat the CPU, disk, and network link of the server as a single resource and consider the total non-blocked time of a request on it. We could also take a holistic view and consider it the time spent on the bottleneck resource on the path from server to client. We take the position that the service time of a request is the time that the combination of server, client, and network would take to finish the request given no other requests in the whole end-to-end system (no load on any server resource). In the following sections, we use this definition and argue that our measurement methodology measures it by verifying that the loads on the resources of the end-to-end system that we measure are miniscule.

To measure correlation between file size and service time we use the correlation coefficient (Pierson's R) [23]. To answer the question posed by this section, we examine R values for a large trace acquired by us from a typical web server, as well as 70 traces collected from web cache servers. The main conclusion is that R can vary considerably from server to server, and can be quite small. $R = 0.14$ for our web server trace, while the web caches have R evenly distributed in the range $[0.12, 0.61]$. In subsequent sections, we use our web server trace to drive our simulation. However, we also use synthetically generated traces in which we can control R directly. While many web server traces are available, none that we could find record the actual service time of the request and thus are not useful for the purposes of our study.

A.2.1 Measurement on a typical web server

We modified the code of the Apache log module so that it records the *response time* of each request with microsecond granularity (using the IA32 cycle counter to measure time). Under extremely low load conditions, as we document below, this time is *equivalent* to the service time according to our definition above.

We deployed the module on our department-level web site. We collected data

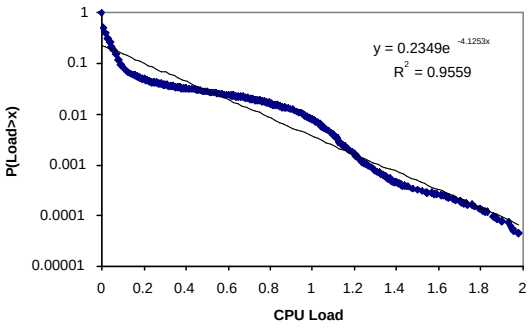


Figure A.3: Complementary distribution of CPU load on the web server.

from September 15, 2003 to October 19, 2003. This trace includes approximately 1.5 million HTTP requests, among which less than 2% are dynamic PHP requests that collectively took less than 1% of the total service time recorded. $>$ 98% of our requests and $>$ 99% of the service time in the trace are for static pages. Hence, our web server is dominated by static web content. our results are comparable to previous work [157, 133, 112] that claims static content dominates web traffic. The requests originated from 110 "/8" IP networks, 7220 "/16" IP networks and 31250 "/24" IP networks spread over the world. We claim that this server is typical. However, the conclusions of this appendix are also supported by other measured traces and generated traces.

The bottleneck resource of a request in this trace is hardly ever the CPU of the server. The web server is a dual processor Pentium IV Xeon machine running Red Hat Linux 7.3. CPU load is defined as the exponentially averaged number of jobs in the run queue of the OS kernel scheduler (the Unix load average), The machine can serve two CPU intensive applications with full CPU cycles. Figure A.3 plots the complementary distribution of the CPU load during the period of the traces with the vertical axis in log scale to better show details. This distribution can be modeled with

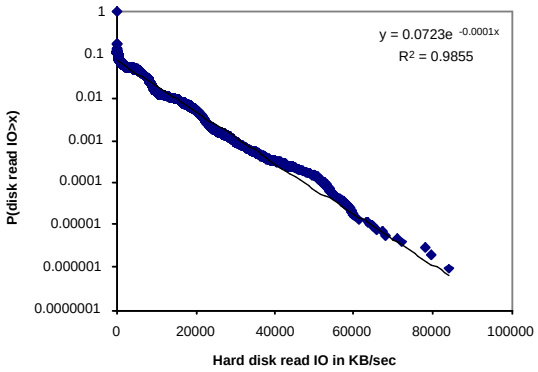


Figure A.4: Complementary distribution of hard disk read I/O on the web server.

Char read	Block read	WebRead	Char write	Block write
23604.2	1399254.2	29879.3	16777.9	50355.8

Figure A.5: Hard disk to memory bandwidth, KB/sec.

a exponential distribution with $R^2 \approx 0.96$. Figure A.3 shows that the probability $P[CPUload > 2]$ is minuscule. The memory system is also clearly not a bottleneck based on these results as significant cache stalls would show up as increased load.

The bottleneck resource of a request in this trace is hardly ever the disk system of the server. The machine's file systems reside on a NFS-mounted (over private gigabit Ethernet SAN) RAID 5 storage server. Figure A.4 shows the complementary distribution of the storage system reads during the period of the trace. The vertical axis is log scale to show details. The distribution can be modeled with a exponential distribution with R^2 close to 0.99.

We benchmarked the storage system using Bonnie, which is a widely used utility that measures the performance of Unix file system operations that an application sees [2]. Bonnie reads and writes a 100 MB file (marked uncacheable) by character or by block. Both sequential and random access are tested. Random block and character throughput give us upper and lower bounds on the performance of file

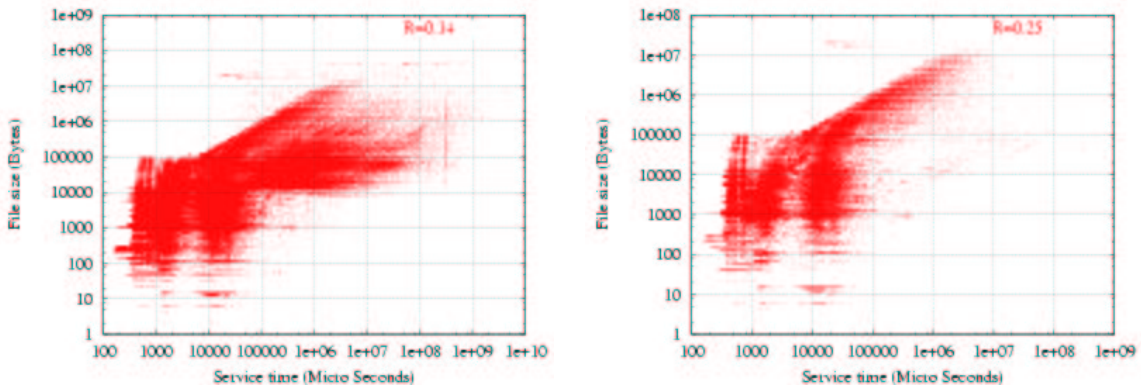
system I/O that Apache sees. We also wrote our own benchmark (WebRead) to get a sense of the typical read performance that Apache sees. WebRead simply reads the files in our access log, in order, as fast as possible. Not surprisingly, the WebRead performance is in between the character read and block read benchmark given by Bonnie. WebRead's performance is shown Figure A.4 as a vertical line, while all the results are shown Figure A.5. We can see that probability of read throughput being larger than the throughput measured in the WebRead benchmark is < 0.001, while no recorded read throughput was larger than Bonnie's block read benchmark. Notice also that the highest throughputs seen are lower than the 125 MB/s throughput limit of the Ethernet SAN, hence the SAN is also not a bottleneck.

As it is clear that the CPU, memory, and disk systems are not bottlenecks, if there is any bottleneck it is in the network or the client. Based on many earlier measurements of load behavior on clients that indicate their resources spend much of their time idle [166, 73], it is extremely unlikely for a client to be the bottleneck. If there is any bottleneck, it is in the network path to the client, which agrees with earlier work [171, 112], which showed that the network is the bottleneck for the web servers serving mainly static contents. Given the low rate of requests, it is highly likely that a single request would perform similarly to the requests in our trace. Hence, the high-resolution response time that we record in the Apache log is a close approximation of the service time as defined above. Obviously, there are situations where CPU or disk can become bottlenecks, such as in virtual server configuration in which one physical server hosts several web sites, or on a web server that hosts database-based dynamic web content.

Given the provenance of the trace, we can now use it to answer our question. Figure A.7 (a) is a log-log scatter plot of file size versus service time. Visually, we can see hardly any correlation between file size and service time. File transfer times

File Size	R
$X < 30$ KB	0.0616
$30 \leq X < 500$ KB	0.1121
$X > 500$ KB	0.1033

Figure A.6: R depends on file size.



(a) R=0.14 (b) R=0.25

Figure A.7: Scatter plot of file size VS. service time. (a) shows the plot of the whole web trace, where the R is about 0.14. (b) shows the trace of a particular /16 network, where the R is about 0.25.

vary over several orders of magnitude with same file size. *Over the entire 1.5 million requests in the trace, we find that R is a very weak 0.14.* R varies slightly with file size, as can be seen in Figure A.6.

Within a domain, R is larger. We define precisely what we mean by a domain and connect it with CIDR in Section A.4. Here, simply consider it as a single network that may be recursively decomposed into subnetworks. For example, Figure A.7 (b) is a log-log scatter plot of file size versus service time for requests originating with a single "/16" IP network, where the network address is 16 bits. $R = 0.25$ for this situation. As the domain grows smaller (has fewer IP addresses, or more bits representing its network address), R grows larger. For example, if we focus on a particular "/24" LAN subnet (24 bit network address) that is contained within the previous network,

$R = 0.39$. We speculate that the reason for this behavior is that network bandwidth heterogeneity from the server to the clients of a domain decreases as the size of the domain decreases. This provides a different, but compatible, explanation for earlier findings [33] that file size-based SRPT scheduling can decrease mean sojourn time by a factor of 3-8 over PS in a LAN for load higher than 0.5, but can only decrease the mean sojourn time by 25% on the WAN. In Section A.3, our simulations show that when $R \approx 0.4$, as on the example LAN, file size-based SRPT outperforms PS by a factor of about 3, but when the $R < 0.2$ (recall that our web trace showed $R = 0.14$) file size-based SRPT performs similar to PS and can perform worse than PS if R goes down further, when file sizes are hardly any indicator of service times at all.

We are actively acquiring additional traces, but this is difficult because web server modifications are necessary to acquire fine grain service times. Many available traces, such as those from the Internet Traffic Archive [7], our institution's other web servers, and others provide only file size, not service time and thus are unsuitable for our work. We have, however, acquired many traces from web caches, described next, and built a trace generator that allows us to control R as well as the distributions of service time and interarrival time, described later.

A.2.2 Measurement on web caches

We examined 70 sanitized access logs from Squid web caches, made available through the ircache site [8]. These traces contain actual fine grain service times (not response times) in addition to file sizes. Internet object caching stores frequently requested Internet objects (i.e., pages, images, and other data) on caches closer to the requester than to the source. Clients can then use a local cache as an HTTP proxy, reducing access time as well as bandwidth consumption.

Squid is a high-performance proxy caching server for web clients. Unlike tradi-

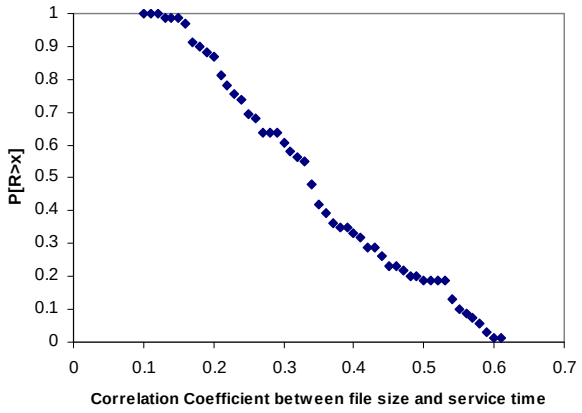


Figure A.8: Complementary distribution of R in web cache traces.

tional caching software, Squid handles all requests in a single, non-blocking, I/O-driven process [10], making it very easy to determine the service time of a request. Squid is similar to a web server in that it also accepts HTTP requests and sends back requested files, but it is different in that the Squid servers form a overlay network that uses the Internet Cache Protocol (ICP) to perform server selection for web clients and load balancing among the cache servers [228, 227]. A client sees that it typically receives a reply from the nearest cache server, while from the Squid cache servers' points of view, the Internet is divided into several regions with each cache server typically serving requests for a specific region.

Because a single Squid cache serves clients largely from one region of the Internet, the bandwidth heterogeneity to the clients is likely to be less than that seen by a web server, which services clients regardless of region. This, we believe, should lead to larger R being measured on Squid caches than on web servers. The partitioning of the network as seen from the web server into domains that we describe in Section A.4 builds on this observation.

While we cannot (and do not) use web cache traces as proxies for web server traces, it is instructive that R on the caches is also rather weak. Figure A.8 shows

a complementary distribution plot of the R values in the 70 traces. The traces were collected from 10 squid web cache servers over 7 days Each trace contains from 0.1 to 1.1 million requests. The smallest $R = 0.12$, while the largest $R = 0.61$. The mean is 0.34 with standard deviation 0.13. Given that we expect that R for web servers will be lower than R for web caches by the reasoning in the previous paragraph, that measured Rs on web caches are low suggests that R on web servers is likely to be low as well.

In combination with the low R seen on our web server trace, we believe that we can now answer the question posed by this section in the negative: *The correlation between request file size and service time on web servers is weak.*

A.3 How is the performance affected by the weak correlation?

We have seen that request service time on web servers and caches is not strongly correlated with request file size. Here, we investigate, via simulation, how this weak correlation (R) affects the performance of size-based scheduling policies (SRPT and FSP, where actual service time is known a priori, and SRPT-FS and FSP-FS, where the file size is used as the service time) and compare with a size-oblivious policy (PS). Our metrics are the mean sojourn time and mean queue length. In our earlier work [154], we find that for these metrics, the performance of SRPT-FS and FSP-FS is dramatically affected by R, falling below that of PS for low R values. Here for the web server case, where we have a fixed low R, we study how the performance of the file-size based policies (SRPT-FS, FSP-FS) could divergence from their ideal versions (SRPT, FSP) as we increase the load on the web server.

A.3.1 Simulator

Our simulator supports both M/G/1/m and G/G/n/m queuing systems. It is driven by a trace in which each request contains the arrival time, file size, and service time. In addition to the web server trace described in the previous section, our simulator can also support synthetic traces generated with interarrival times from exponential, bounded Pareto, and Weibull distributions, and file sizes from bounded Pareto and Weibull distributions, and service times from bounded Pareto. The correlation R between file size and service time in a synthetic trace can also be directly controlled. We refer readers to our earlier work [154] for more details about this general-purpose simulator.

A.3.2 Simulation with web server trace

Here we consider the performance of SRPT, SRPT-FS, FSP, FSP-FS, and PS on the measured web server trace ($R = 0.14$) described in Section A.2.1. The mean service time is 1250 microseconds. The scheduling policies are described in Figure A.1. Note that although our web server trace represents very low load, here we vary the load in the system by controlling the arrival process of the requests represented in the trace. We make use of Poisson arrivals, Pareto arrivals, and Weibull arrivals and control their mean rate in order to control the load. Load control is important, because, as we discussed in Section A.2.1, the load captured in the trace is rather low. The time units are microseconds throughout the rest of the appendix. Each simulation throughout the rest of the appendix is repeated 20 times.

First, we consider G/G/1/m (Job interarrival has a heavy-tailed Pareto distribution, file sizes and service times as in the trace). Figure A.9 shows the mean sojourn times of different scheduling policies with increasing load, while Figure A.10

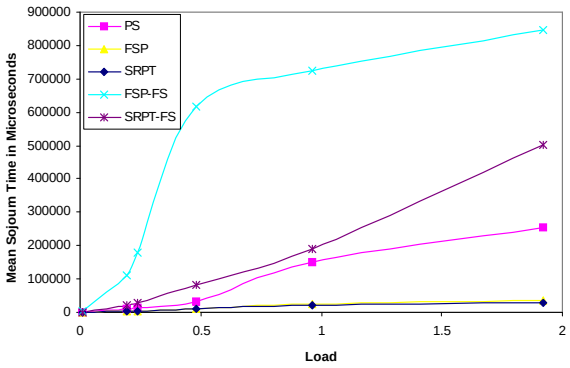


Figure A.9: Mean sojourn time versus load, G/G/n/m, Pareto arrivals. Web server trace driven simulation.

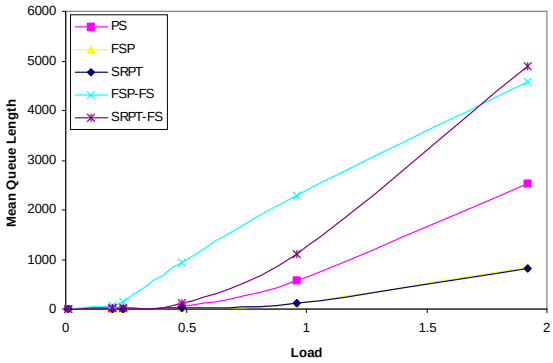


Figure A.10: Mean queue length versus load, G/G/n/m, Pareto arrivals. Web server trace driven simulation.

shows the mean queue length of different scheduling policies with increasing load on the queue. In both figures, ideal SRPT and FSP perform very well and almost identically. However, SRPT-FS and FSP-FS both perform quite poorly, and their performance diverges dramatically from their ideal performance with increasing load. SRPT-FS and FSP-FS perform worse than SRPT and FSP in all our simulations.

For a queue with unlimited queue capacity, the mean sojourn time and queue length tend to be infinity if the load is over unity, and therefore it is meaningless to present mean sojourn time and queue length. Our simulator uses a finite queue

capacity to better match implementations. The server will begin to reject jobs when it is overloaded for some period of time (when queue is full). Hence, both mean sojourn time and mean queue length are meaningful; they represent the behavior of the server under transient overload.

We have also investigated a Weibull arrival process and Poisson arrival process, where the interarrival times of requests in the trace are drawn from a Weibull distribution and exponential distribution respectively. The results are similar to those for the Pareto arrival process shown earlier.

Our simulations show that the performance of SRPT-FS and FSP-FS, SRPT and FSP where request file size is used as request service time, is largely affected by the weak correlation R between file size and service time. With such a low R in our trace, performance can degrade so far that PS is preferable to either of these policies. Our trace shows such a low R. On the other hand, our earlier work [154] shows that over wide range of R, including the range seen in the web cache traces examined, increasing R can dramatically improve the performance for SRPT-FS. In the next Section, we describe and evaluate a better estimator for service time that uses file size, the network "domain" of the client, and past performance to the domain to produce more accurate service time estimates in those regimes where the correlation between file size and service time is low. We believe these regimes are commonplace.

A.4 Domain-based scheduling

We have found that request file size and service time are weakly correlated and that the performance of size-based scheduling policies are strongly dependent on the degree of this correlation. Given these results, a natural question is whether there is a better service time estimator than file size, one whose estimates are more strongly correlated

with actual service time. Such an estimator must also be lightweight, requiring little work per request. For this reason, we cannot use active probing techniques. Instead, we explore the web logs and use past web requests as our probes.

A.4.1 Statistical stability of the Internet

Domain-based scheduling relies on the Internet being statistically stable over periods of time, particularly from the point of view of the web server. Fortunately, there is significant evidence that this is the case. This evidence falls into two classes, routing stability and spatial and temporal locality of end-to-end TCP throughput.

Routing stability: Paxson [173] proposed two metrics for route stability, prevalence and persistency. Prevalence, which is of particular interest to us here, is the probability of observing a given route over time. If a route is prevalent, then the observation of it allows us to predict that it will be used again. Persistency is the frequency of route changes. The two metrics are not closely correlated. Paxson's conclusions are that Internet paths are heavily dominated by a single route, but that the time periods over which routes persist show wide variation, ranging from seconds to days. However, 2/3 of the Internet paths Paxson studied had routes that persisted for days to weeks. Chinoy found that route changes tend to concentrate at the edges of the network, not in its "backbone" [61]. Barford, et al measured the web performance in the wide area network and found that the routes from/to the client to/from a web servers was asymmetric, but very stable [36].

Spatial locality and temporal locality of end-to-end TCP throughput: Balakrishnan, et al analyzed statistical models for the observed end-to-end network performance based on extensive packet-level traces collected from the primary web site for the Atlanta Summer Olympic Games in 1996. They concluded that nearby Internet hosts often have almost identical distributions of observed throughput. Al-

though the size of the clusters for which the performance is identical varies as a function of their location on the Internet, cluster sizes in the range of 2 to 4 hops work well for many regions. They also found that end-to-end throughput to hosts often varied by less than a factor of two over timescales on the order of many tens of minutes, and that the throughput was piecewise stationary over timescales of similar magnitude [28]. Seshan, et al applied these findings in the development of the Shared Passive Network Performance Discovery (SPAND) system [200]. Myers, et al examined performance from a wide range of clients to a wide range of servers and found that bandwidth to the servers and server rankings from the point of view of a client were remarkably stable over time [167]. Yin Zhang, et al [238] found that three Internet path properties, loss rate, delay and TCP throughput show various degrees of constancy and concluded that one can generally count on constancy on at least the time scale of minutes.

A.4.2 Algorithm

Although the Internet, web servers, and clients form a highly dynamic system, the stability we pointed out in the previous section suggests that previous web requests (the web server's access log) are a rich history which can be used to better estimate the service time of a new request. We assume that after processing a request we know (1) its file size, (2) the actual service time, and (3) the IP address of the client. Collecting this information is simple and efficient. Our goal is to develop an efficient estimator that uses a history of such requests, combined with the file size and IP address of the current request to determine the likely service time of the current request. The correlation R between the estimated service time and the actual service time should be higher than the correlation between file size and actual service time. Recall that R must exceed a threshold in order for SRPT to perform better than PS,

and as R increases, the performance of SRPT increases.

Classless Inter Domain Routing (CIDR) [99] was proposed in 1993 as "a strategy for address assignment of the existing IP address space with a view to conserve the address space and stem the explosive growth of routing tables in default-route-free routers". The CIDR strategy has been widely deployed since 1993. "One major goal of the CIDR addressing plan is to allocate Internet address space in such a manner as to allow aggregation of routing information along topological lines". Consider a *domain*, a neighborhood in the network topology. The broad use of CIDR implies that routes from machines in the domain to a server outside the domain will share many hops. Similarly, the routes from the server to different machines in the domain will also have considerable overlap. This also means that the routes will be likely to share the same bottleneck network link and therefore have similar throughput to/from the server. The smaller the domain, the more the sharing.

The aggregation of CIDR is along a hierarchy of increasingly larger networks and is reflected in IP addresses. The first k bits of an IP address gives the network of which the address is a part, the first $k-1$ bits give the broader network that contains the first network, and so on. We exploit this hierarchy in domain-based scheduling, the algorithm of which is given below.

1. Use the high order k bits of the client IP address to classify the clients into 2^k domains, where the k bits are treated as the domain address.

2. Aggregate past requests to estimate the service rate (or representative bandwidth) for each domain. This can be done with several estimators, but our experiments show that the estimator $S_R = \frac{F_s}{S_t}$ performs the best. Here S_R is the representative service rate, F_s is the sum of the requested file sizes from the domain, and S_t is the sum of the service times for these requests. Notice that

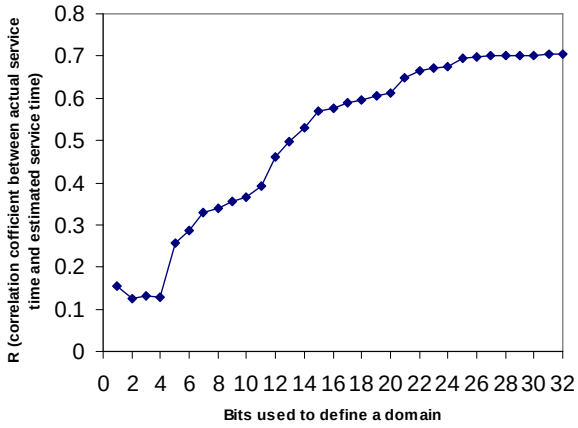


Figure A.11: Correlation R versus number of bits used to define a domain k.

updating this estimate after a request has been processed is trivial: simply add the request's file size and service time to F_s and S_t, respectively (two reads, two adds, two writes). For each domain, we store F_s and S_t. An array of these pairs is kept, indexed by the domain address. The total state size is 2^{k+1} floating point numbers.

3. For each incoming client request, the web server first extracts the domain address, indexes the array and computes S_R for the domain. It then estimates the request's service time as $T_{estimate} = \frac{f_s}{S_R}$, where f_s is the request file size. The estimator requires a logical shift, two reads, a division, and a multiply. For a request from a heretofore unobserved domain, which occurs exactly once per domain, we simply use file size as the estimate.

4. Apply a size-based scheduling policy such as SRPT using the estimated service times. We suffix the scheduling policy with "-D": SRPT-D, FSP-D.

As we might expect, as domains become smaller (k gets larger), predictive performance increases, at the cost of memory to store the state. Figure A.11 shows the

relationship between k, the number of bits used to define a domain and the correlation R between the actual service time and estimated service time. The figure is derived from our web server trace. R jumps to 0.26 with $k = 5$ bits, beyond the threshold at which SRPT begins to perform better than PS. Notice that this is a mere 32 domains (state size of 256 bytes with 4 byte floats). After $k = 24$ bits, there are only very small increases of R, probably because at this point we have divided the Internet into LANs, where each machine on a LAN shares a common route to every other machine in the Internet, and thus shares the same bottlenecks. The maximum R we were able to achieve was 0.704.

A.4.3 Performance evaluation

To evaluate domain-based scheduling (SRPT-D and FSP-D, also see Figure A.1), we use the methodology of Section A.3.2. We replay our web trace with Poisson, Pareto, and Weibull arrivals to control load. We vary k, the number of high-order bits we use to define a domain.

Figures A.12 and A.13 show the mean sojourn time and mean queue length of all the scheduling policies with heavy-tailed Pareto arrivals as a function of k. Notice that PS, FSP, SRPT, FSP-FS, and SRPT-FS are flat lines. PS ignores service time. FSP and SRPT have exact knowledge of the service times (they represent the ideal performance of these policies). FSP-FS and SRPT-FS use file size as a proxy for service time (representing current practice). Notice that as we increase the number of bits k used to define a domain, the performance of SRPT-D and FSP-D first exceeds that of PS and finally converges to near the ideal performance.

While SRPT-D's performance increases continuously, with diminishing returns, with increasing k, FSP-D is rather insensitive until $k = 16$ to 24 bits, at which point its performance jumps dramatically and comes very close to SRPT-D's. Since R

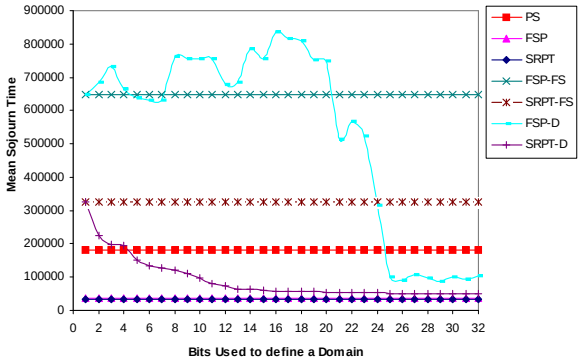


Figure A.12: Mean sojourn time versus k for web trace, domain-based scheduling, G/G/1/m, Pareto arrivals with $\alpha = 1.32$, Lower bound 84, Upper bound 5×10^5, load 0.88.

doesn't increase much beyond $k = 24$ bits, as we might expect, the performance of SRPT-D and FSP-D plateaus. Similar conclusions can be drown for Poisson arrivals and Heavy-tailed Weibull arrivals.

Our performance evaluation of SPRT-D and FSP-D demonstrates that better, practical estimators of service time are possible and that they can dramatically improve the performance of size-based scheduling policies on web servers.

A.5 Conclusions and future work

This appendix has made the following contributions:

- We have demonstrated that the assumption that file size is a good indicator of service time for web servers is unwarranted. File size and service time are only weakly correlated. The implication is that size-based scheduling policies such as SRPT and FSP are likely to perform worse than expected.

- We have evaluated the performance of SRPT-FS and FSP-FS, SRPT and FSP by running simulations driven by our web server traces. We found that their

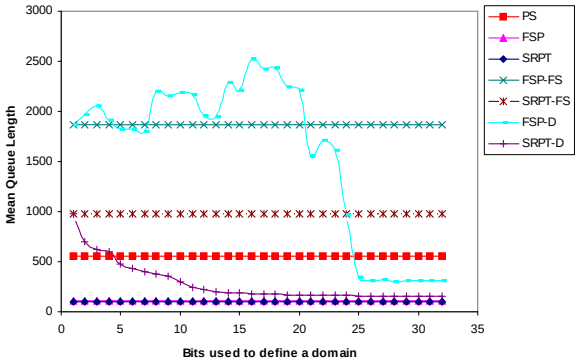


Figure A.13: Mean queue length versus k for web trace, domain-based scheduling, G/G/1/m; Pareto arrivals with $\alpha = 1.32$, Lower bound 84, Upper bound 5×10^5, load 0.88.

performance does indeed degrade dramatically with the weak correlation reflected in the trace. In some cases SRPT-FS and FSP-FS can actually perform worse than PS.

- We have proposed, implemented, and evaluated a better service time estimator that makes use of the hierarchical nature of routing on the Internet and the history of past requests available on the web server. We refer to SRPT and FSP augmented with our domain-based estimator as SRPT-D and FSP-D. The state size of our estimator is a parameter.

- We have found that, with a small state size, SRPT-D can outperform PS. With a practical state size, SRPT-D can exhibit close to ideal performance. FSP-D requires a significantly larger state size to perform close to its ideal. SRPT reacts very quickly to increasingly accurate service time estimates.

Fairness is an important concern in the deployment of domain-based scheduling. Slowdown has been used in previous research work [33, 112] and our earlier work [154] as the fairness metric. We have studied fairness using our simulator and our initial

results show that SRPT-D outperforms PS in fairness under most conditions. We are working to extend these results.

Because the TCP connection (and disk) that a request uses can block, implementations of size-based scheduling in web servers often use what we call *back-filling*. An executing request that becomes blocked is preempted in favor of a request with a larger number of bytes remaining to be handled—it is the non-blocked request of smallest remaining size that is run, not the smallest request. Our simulations do not model such a system. However, as far as we are aware, there are no analytical results for size-based scheduling with blocking behavior either, making it quite difficult to validate a simulator. We are working to extend our simulations and analysis to cover this gap.

Appendix B

P2P server side scheduling

I also contributed to the P2P server side scheduling project conducted at the Computer Science department of Northwestern University. The initial paper was published in LCR'04 [182].

B.1 Introduction

The popularity and tremendous success of peer-to-peer (P2P) systems have motivated considerable research on many of the paradigm's technical aspects. In the context of data-sharing services, a number of projects have explored a wide variety of issues including more scalable object location, query and routing protocols, fair resource sharing, and high churn-resilient systems, just to name a few. The majority of these projects have, so far, concentrated on either the whole P2P infrastructure or the *client* side of a peer. Little attention has been given to the peer's *server*-side, although that side determines much of the everyday user's experience.

After determining alternative sources for a desired object, the requesting peer initiates the object downloads from a subset of possible providers; each party effectively

adopting *client* and *server* roles. Recent studies suggest that the server-side in this interaction often turn out to be a performance bottleneck. From the analysis of P2P traffic collected at border routers at the University of Washington, Saroiu et al. [195] report that a small number of Kazaa [14] servers are responsible for serving the majority of requests for content. Their traces indicate that over 80% of all download requests are rejected because of the saturation of server capacity. Similarly, another study of P2P workload by the same group [107] shows that object downloading in Kazaa can be extremely slow, with 50% of all requests for large objects (>100MB) taking more than one day and nearly 20% taking over one week to complete!

These results clearly argue for taking a closer look at the server-side of peers, and this appendix reports on our initial steps. This work focuses on the scheduling problem, and the goal is to design efficient and fair scheduling algorithms for P2P servers that result in a lower average response time (a.k.a. sojourn time) for serving downloading requests of client peers. Despite the similarity in purpose with research on scheduling algorithms for web servers [25, 66, 34, 155], a closer look at the characteristics of P2P request traces indicate that many findings from the web context are not directly applicable to our problem:

- The *fetch-at-most-once* behavior of P2P client makes the distribution of object popularity decidedly *not* conform to Zipf or other power-laws [107].

- Requests to P2P servers are often not for the whole object, but instead for only a small chunk (with the remaining parts downloaded from other servers). In fact, as our traces show, the amount of data actually served is often just a fraction of the requested size.

- While web servers can reasonably assume full control over resources, P2P servers are commonly configured with quite conservative upper bounds for re-

source consumption to control their impact on their users' other tasks.

- Although web servers often experience high load [1], close to 1, they are typically not overloaded. Popular P2P servers, on the other hand, normally operate overloaded [195] due in part to low resource availability and, on average, large object sizes.

This chapter starts by characterizing server workload through trace collection and analysis. The traces of download requests were collected from a set of P2P servers behind 100Mbps and cable modem connections. To the best of our knowledge, ours is the first attempt at characterizing server workload on P2P systems.

This chapter studies the performance and fairness of different scheduling policies using our workload characterization and trace-driven simulations. The results show that average response time can be dramatically reduced by scheduling jobs on the server-side of P2P systems using policies based on preemptive *Shortest-Remaining-Processing-Time (SRPT)*.

B.2 Trace Collection

For the study of server workload characterization and the subsequent analysis of scheduling policies for P2P servers, this chapter makes use of set of traces collected from Gnutella [13], a popular data-sharing P2P system.

The workload traces were collected using a set of *honey-pots*, peers offering a large number of popular files to other peers in the Gnutella network. At each honey-pot and for each incoming download request, the request arrival time, object name,

[1] In this appendix, the load is defined as mean job arrival rate over mean service rate, as is the standard definition for load in queuing theory [197].

size of requested and served data chunk and transfer finish time were recorded for further analysis. Each of the honey-pots was built on a modified open-source Gnutella client [15].

To avoid potential bias in data collection, multiple honey-pots at different hosts were employed, each serving its own collection of shared objects, and each configured with different upper bounds for outgoing bandwidth and number of threads serving requests. To ensure that the behavior of busy server peers is caught, most of these limits were set much higher than their default settings. In order to capture potential differences due to bandwidth classes, traces were also collected using a peer behind a cable connection. Some key parameters of the traces are summarized in Figure B.1.

Connection Type	Number of Threads	Number of Objects	Number of Requests
100Mbps Ethernet	200	1,533	300,000
100Mbps Ethernet	100	1,533	150,000
100Mbps Ethernet	50	500	80,000
Cable Modem	20	1,533	40,000

Figure B.1: Key parameters of collected traces from P2P servers. *Number of Threads* is the number of available server threads.

B.3 Server Workload Characterization

Server workload characterization forms the basis for any work on scheduling policies. This section addresses the following questions for the case of data-sharing P2P servers. The terms "job" and "request" are used interchangeably.

- What is the distribution of job interarrival time?
- Are the job arrivals independent?

- What is the distribution of job size and job service time[2]?
- What is the likely performance bottleneck?
- What are the implications of our findings on P2P system scheduling?

B.3.1 Job Arrivals Form a Poisson Process

The job interarrival times is characterized for P2P servers based on the collected traces. Figure B.2 gives the complementary cumulative distribution function (CCDF) of job interarrival time at a P2P server for a typical trace. Notice that the vertical axis is logarithmic; the straight line of the CCDF curve strongly indicates that the arrival process can be modeled by an exponential. The least-squares curve-fitting using an exponential function, indicated by the dash-line in Figure B.2, with coefficient of determination $R^2 = 0.9943$ quantifies our argument.

The independence of job arrivals is tested by computing the serial correlation of their interarrival times, as shown in Figure B.3. Clearly, the correlation between any two separate interarrival times is effectively nil. Since each of our traces exhibits similar behavior, job interarrivals for a P2P server can be well modeled as independent of each other, clearly a significant difference from the web server case. Exponentially-distributed, independent interarrival times are the definition of a Poisson process.

Previous research [175, 72] has shown that Poisson processes are valid for modeling the arrival of user-initiated TCP sessions such as TELNET and FTP connections. HTTP arrivals, on the other hand, have been shown not to be Poisson. Deng et al [72] point out that the aggregated interarrival times of HTTP requests can better

[2]In this appendix, The job service time is defined as the wall clock time it takes a server to finish sending data to a client over the Internet given the bounded outgoing bandwidth for the job. Similarly, the response time of a job is the sum of its service time and its total waiting time in the queue.

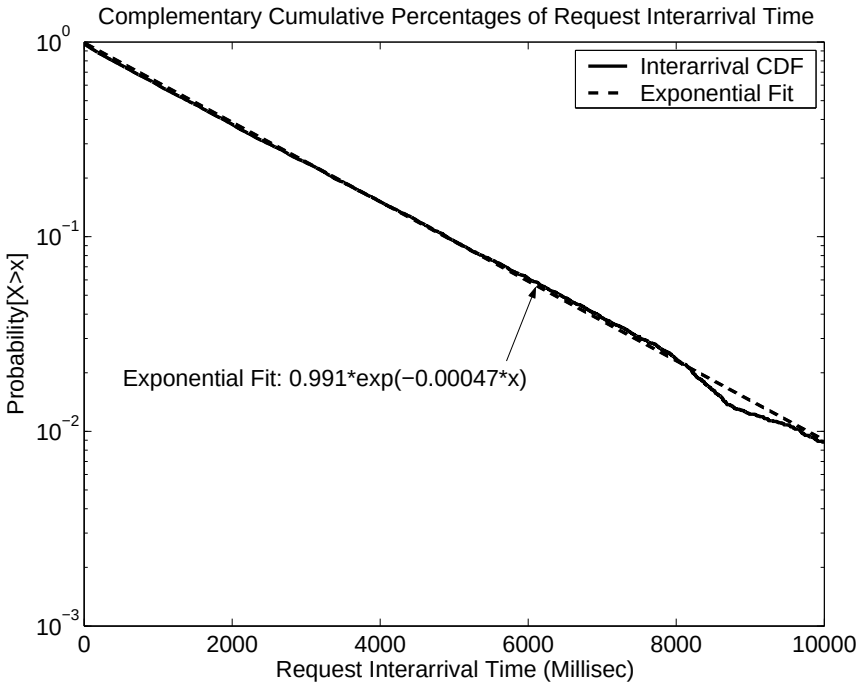


Figure B.2: CCDF of interarrival time of requests to P2P server

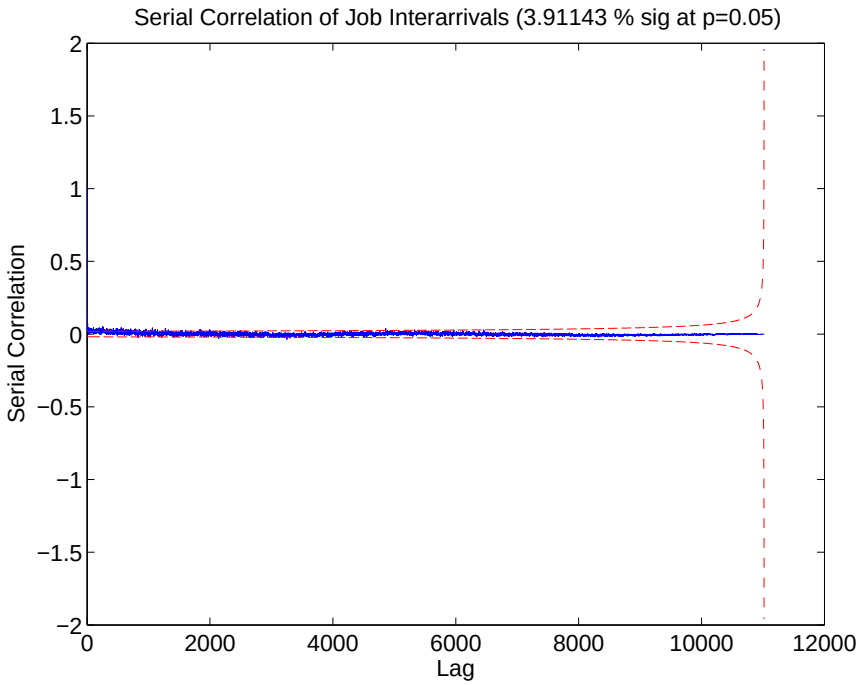


Figure B.3: Serial correlation of interarrival time of requests to P2P server

be modeled by a heavy-tailed Weibull distribution. This is because HTTP document transmissions are not entirely initiated by the user; some are automatically generated by the browser (requesting embedded files), resulting in a more bursty process.

Although P2P server requests, like web requests, are not solely initiated by the users, there are some interesting peculiarities of client peers that may explain the observed differences. For example, a client searching for a given object collects a set of candidate servers from which it later initiates parallel downloads. In addition, clients can abandon (switch) servers in the middle of a download, after finding an alternative source with higher available bandwidth [15, 12, 38].

B.3.2 Job Sizes are Pareto

Job size is an important property for queuing models. Interestingly, for P2P scheduling, there are three different possible definitions for job size: *full object size*, *requested data chunk size*, and *served data chunk size*. While the full object size is usually very large, most requested data chunks are small, covering only a small fraction of the whole object. More importantly, there is usually also a significant difference between the requested data chunk size and the actual served data chunk size. Some possible explanations are discussed for this difference in Section B.4.

The CCDFs of the three job sizes are depicted in log-log scale in Figure B.4. As it is clear from the graph, the three often differ by several orders of magnitude. This clearly distinguishes P2P server requests from web requests and supports the argument for taking a closer look at the server side of P2P systems. The remainder of this appendix focuses mainly on the requested and served data chunk sizes as these two are the main determinants of P2P server performance. For all of the traces, the distribution of these sizes can be modeled as a Pareto with high R^2 values (0.9293 and 0.9452 for the example in Figure B.4).

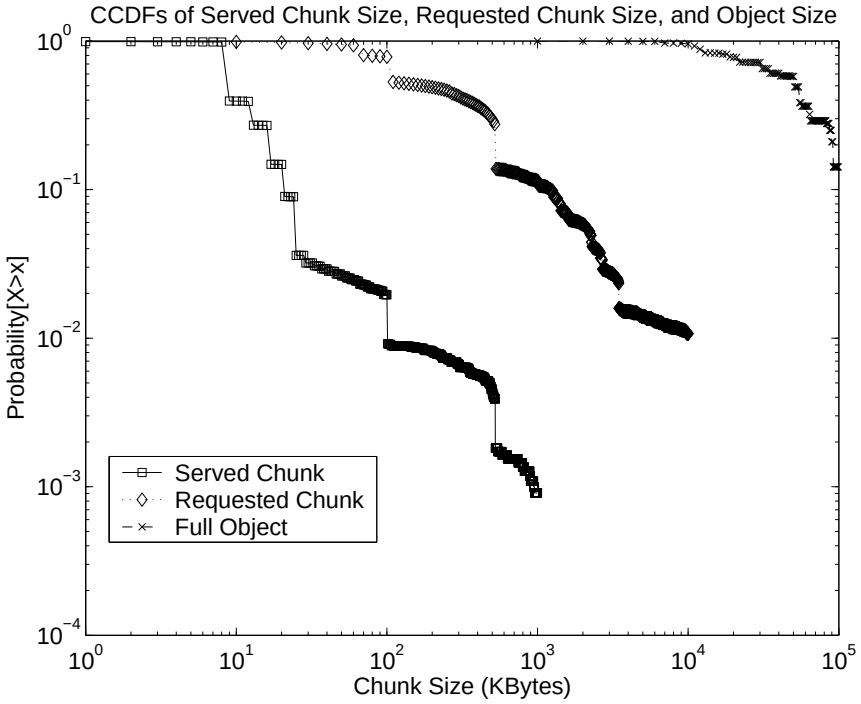


Figure B.4: CCDFs of served data chunk size, requested data chunk size, and full object size

B.3.3 Job Service Times Are Pareto

Job service time is another aspect of the workload that will help us understand the potential benefits of different scheduling policies. Due to the different download speeds of clients, job service time is not directly proportional to any of the three job sizes, but can be well approximated by a Pareto distribution, one of the simplest forms of heavy-tailed distributions.

B.3.4 Server Resource Utilization

Despite their apparent similarities with web servers, the resource utilization of P2P servers could be quite different. Web servers typically try to serve requests as quickly as possible and, as it has been shown, their bottleneck resource is commonly the limited bandwidth of the outgoing link [34]. P2P servers, on the other hand, are normally run on the background of common users' machines and are thus more conservative in their use of resources.

The honey-pots [3] was instrumented to periodically (every three seconds) record different metrics such as CPU and memory usage. The traces show that even when the servers support 200 concurrent downloads and use up to 2 MBytes/second of bandwidth, CPU utilization is always between 1.2% and 20%, and memory usage is consistently below 20 MBytes. Thus, unlike the web server case, neither CPU, memory, nor bandwidth turn out to be the performance bottleneck for a P2P server, not even for the most popular of our honey-pots.

These low resources utilization can be largely attributed to the user-defined upper-bounds on bandwidth usage and number of concurrent server threads. These same

[3]Each of our servers is a dual 1 GHz Pentium III machine with 1 GB RAM and two 30GB IDE disks running Red Hat Linux 7.3.

tight upper-bounds are what make P2P servers the performance bottleneck of the whole system [195, 107].

It is clear from this analysis that the bottleneck resource to schedule is the set of server threads on a server, i.e., the collection of concurrent jobs that a server can serve. Our scheduling problem can then be formulated as follows: *Given the total number of concurrent jobs that a server can take, how to schedule the incoming jobs so that their mean response time is minimized?*

B.4 Evaluation of Scheduling Policies

B.4.1 Scheduling Policies

A good scheduling policy should minimize the average waiting time without starving any jobs. Fairness is another important metric for evaluation of a scheduling policy. Fairness has several metrics, with the most recent work [34] using slowdown – defined as a request's response time divided by the time it would require if it were the sole request in the system.

The most commonly used scheduling polices are *Processor Sharing (PS)* and *First Come First Serve (FCFS)*. PS is commonly employed for CPU scheduling and in the current Apache web server, while FCFS is used by common Gnutella implementations such as Mutella [15]. Neither of these policies makes use of other available information, such as size of a job, to improve performance. [4]

Shortest Remaining Processing Time (SRPT) has been studied since the 1960s [197]. For a general queuing system (G/G/1) Schrage [196] proved that SRPT is optimal in the sense that it yields – compared to any other conceivable strategy – the small-

[4] However, some P2P systems (such as eDonkey [12]) consider reputation (scores) as part of their scheduling policy.

est mean value of occupancy and thus also of minimum waiting and delay time. Perera [177] and Harchol-Balter, et al [34] evaluated SRPT in terms of fairness. Perera [177] studied the variance of delay time in the $M/G/1/SRPT$ queuing systems and concluded that the variance is lower than FIFO and LIFO [177], while in [34] the authors proved that SRPT also outperforms PS in terms of mean slowdown, their fairness metric. SRPT has been successfully applied to a number of application areas. Bux [48], for example, introduced SRPT into packet networks using the message size as the service time. More recently, Harchol-Balter et al. [34] proposed the use of SRPT in web servers, relying on file sizes as the estimator of service time.

This appendix introduces SRPT into P2P server-side scheduling. The adoption of SRPT faces some challenges, however. To begin with, ideal SRPT requires knowledge of requests' service times, something not available a priori. In addition, while it may be possible to estimate it [48, 34], the estimation is in itself challenging due to the dynamic characteristics of P2P systems.

B.4.2 SRPT Scheduling in P2P Systems

Since a typical P2P download request is for a specific chunk of the whole object, as described in Section B.3, we could use the requested chunk size as a rough estimate of service time, and as the metric for SRPT scheduling. Unfortunately, as Figure B.5 shows, there are only weak correlations between requested chunk size and either served chunk size or the real service time, which implies that requested chunk size may not be a good estimate of service time. This discrepancy between the requested and served chunk sizes could compromise the performance of SRPT [155].

Several characteristics of the P2P environment could help explain the weak correlation:

Statistics	Service Time	Served Chunk Size	Requested Chunk Size
Service Time	1.0000	0.7023	0.2833
Served Chunk Size	0.7023	1.0000	0.2339
Requested Chunk Size	0.2833	0.2339	1.0000

Figure B.5: Correlation coefficients between service time, served chunk size and requested chunk size.

- A client can exit at any time during the data transmission.
- As already discussed, a P2P client can switch servers for a given data chunk before the request is completed.
- Although each downloading process is supposed to share equal outgoing bandwidth from the P2P server, bandwidth bottlenecks along the path to the destination can make the individual download speed vary.

Figure B.5 also shows a much stronger correlation between served chunk size and service time, indicating that served chunk size can be a very good estimate for service time.

Despite the aforementioned discrepancies between requested and served chunk sizes and the weak correlation between requested chunk size and service time, it may be worthy to evaluate SRPT performance using requested chunk size as its scheduling metric. Lu et al. [156] have studied the behavior of size-based schedulers with inaccurate job size information and concluded that the SRPT can outperform PS given an effective job size estimator. They [155] showed that SRPT outperforms PS when $R > 0.15$ in the case of web server scheduling.

For comparison purposes, the scheduling performance for ideal SRPT is also presented. The three scheduling policies are denoted as SRPT-CS, SRPT-SS, and SRPT,

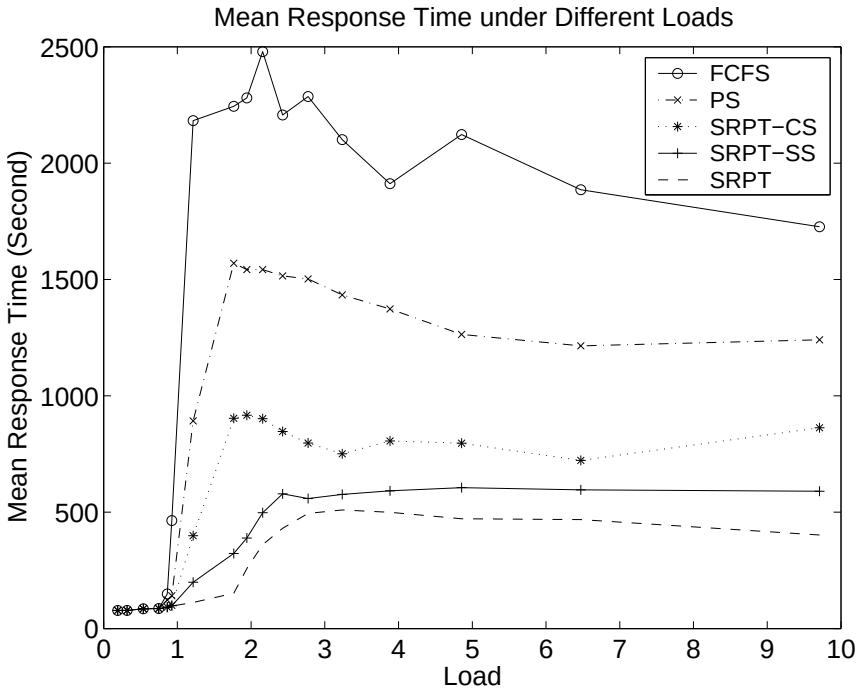


Figure B.6: Mean Response time for different scheduling policies under varying load

respectively. Notice that SRPT-CS can be directly implemented with current tools, while SPRT-SS would require an accurate estimator.

B.4.3 Performance Analysis

A general purpose queuing simulator is built to evaluate the performance of different policies, including PS, FCFS, SRPT-CS, SRPT-SS and ideal SRPT. All simulations were driven by the server-side request traces. For all of the simulations the queue capacity is set to 500. A time slice of 0.01 seconds is used for PS. Besides our own work [156], we are not aware of other previous research addressing SRPT performance with inaccurate job size information.

Figure B.6 gives the mean response time of the five scheduling policies handling

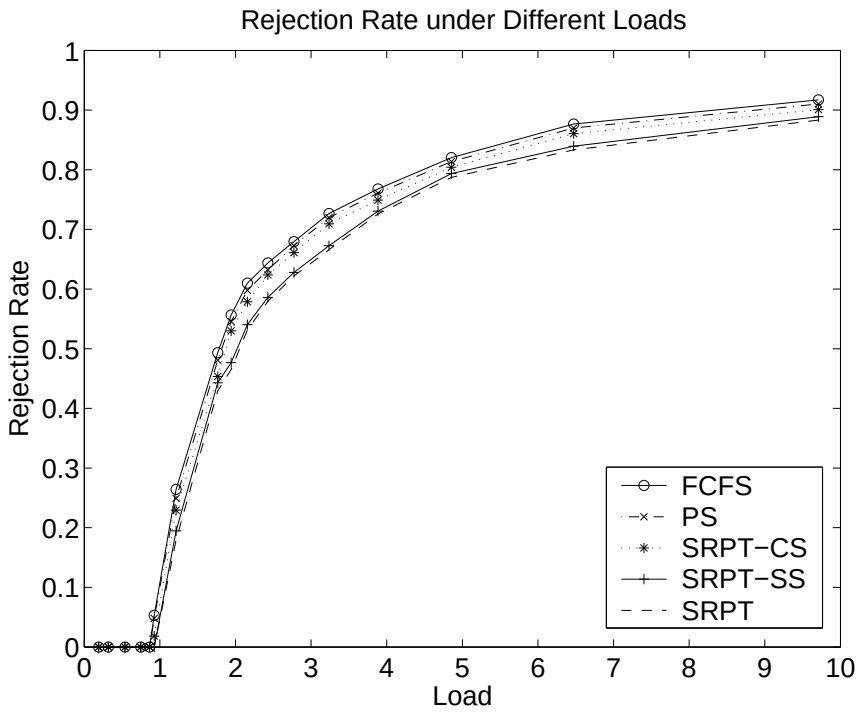


Figure B.7: Rejection rate for different scheduling policies under varying load

all requests for a P2P server, with the system load varying between 0.1 to 10. The advantages of the three SRPT-based policies over PS and FCFS are clear, especially when the load is close to or above 1[5]. When the load is 1.76, for instance, mean response time is 2244.08 seconds under FCFS and 1569.89 seconds under PS. For SRPT-CS, SRPT-SS, and SRPT, however, the number drops to 903.61 seconds, 322.621 seconds, and 151.451 seconds, respectively. This confirms our expectations of SRPT performance.

Similar to what can be observed for web servers [155], even with only a weak correlation between requested chunk size and actual service time, SRPT-CS achieves considerable performance gains over both PS and FCFS. As would be expected, due to the strong correlation between served chunk size and service time, SRPT-SS performs significantly better and even approaches the performance of ideal SRPT under several different system loads.

The actual served chunk size, upon which SRPT-SS relies, is not known until the request is completed. However, it should be possible to predict it fairly accurately. One possible way of achieving this is by finding correlations between object popularity and the level of discrepancy between the requested and served data chunk size. Another approach could be based on both the prediction of served chunk size and the download speed of the client. For the latter, the connection type and the network distance to the client, such a domain-based scheduling, could be potentially applied [155].

[5]In this appendix we are mostly interested in the case where server load is larger than 1, which is normal for a popular P2P server. Moreover, since job arrivals form a Poisson process and lack burstiness, the queue length shrinks abruptly when the load drops below 1. In all scheduling policies evaluated, for example, the mean queue length drops to around 0.10 when system load is 0.75. As can be seen in Figure B.6, SRPT-based scheduling policies still outperform FCFS and PS when the load is smaller than 1, as long as there are jobs waiting in the queue.

B.4.4 Fairness Concerns

One major concern with SRPT scheduling is that it is possible to design an adversarial workload in which SRPT leads to starvation of large jobs. That is, SRPT can be made to behave unfairly. Fortunately, previous research on M/G/1 queuing systems with comparable workloads have shown that the starvation does not occur [177, 34]. Perera [177] proves that the variance of delay time of ideal SRPT is smaller than that of FIFO and LIFO, while Harchol-Balter [34] shows that the mean slowdown of ideal SRPT is actually smaller than that of FCFS and PS. In the context of P2P server scheduling, we consider fairness issues of a scheduling policy from three different aspects: mean slowdown of *large* jobs, rejection rate of requests, and distribution of rejected job size.

Figure B.7 shows the rejection rates for the five policies under various system loads. It can be seen that SRPT actually results in the lowest rejection rate; SRPT-CS and SRPT-SS also reject fewer jobs than FCFS and PS. Our simulations also demonstrate that the distribution of rejected job size is almost identical for all evaluated scheduling policies. Moreover, under various system loads, SRPT-based scheduling policies yield lower mean slowdown for large jobs. When the system load is two, for instance, the mean slowdown for the top 10% largest jobs in the system are: 15.496 (FCFS), 25.615 (PS), 10.723 (SRPT-CS), 8.741 (SRPT-SS), and 7.707 (SRPT).

B.5 Conclusions and Future Work

The server-side of P2P systems often turns out to be the performance bottleneck. Surprisingly, it has received little attention from the research community. This appendix starts this exploration by looking at the problem of download request scheduling. The trace data of P2P download requests experienced by individual P2P servers

was collected and performed analysis and modeling of this server workload. Two SRPT-based scheduling policies were proposed and show their advantages through trace-driven simulations.

Analysis of several inherent characteristics of P2P server requests also reveals considerable room for improvement in estimating request service time, which would let us approach the performance of ideal SRPT. Two possible approaches for estimating service time include: predicting served data chunk size based on object popularity and requested chunk size, and predicting transfer rate based on client type and Internet path characteristics.

Other interesting directions of future work in P2P server-side scheduling are also identified:

- Deeper and more thorough analyses of fairness issues for various scheduling policies.
- P2P server trace collection from different hosts, increasing both the geographical and connection variety.
- Modeling and scheduling for cooperative uploading/downloading, as employed in [12, 11].
- Implementation and evaluation of various scheduling models in P2P software and its evaluation on large-scale Internet testbeds.