RISK-BASED PROACTIVE AVAILABILITY MANAGEMENT - ATTAINING HIGH PERFORMANCE AND RESILIENCE WITH DYNAMIC SELF-MANAGEMENT IN ENTERPRISE DISTRIBUTED SYSTEMS

A Thesis
Presented to
The Academic Faculty

by

Zhongtang Cai

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
College of Computing

Georgia Institute of Technology
April 2008

RISK-BASED PROACTIVE AVAILABILITY MANAGEMENT - ATTAINING HIGH PERFORMANCE AND RESILIENCE WITH DYNAMIC SELF-MANAGEMENT IN ENTERPRISE DISTRIBUTED SYSTEMS

Approved by:

Dr. Karsten Schwan, Committee Chair
College of Computing
Georgia Institute of Technology

Dr. Mustaque Ahamad
College of Computing
Georgia Institute of Technology

Dr. Greg Eisenhauer
College of Computing
Georgia Institute of Technology

Dr. Dejan Milojicic
Hewlett Packard Labs
Hewlett Packard Company

Dr. Calton Pu
College of Computing
Georgia Institute of Technology

Date Approved: 10 December 2007

ACKNOWLEDGEMENTS

This thesis would not have been possible without the generous help, support and encourage from many individuals. I would like to express my gratitude to all of them.

First and foremost, my deepest gratitude is to my advisor, Professor Karsten Schwan, who led me all the way during my Ph.D. study. Karsten Schwan provided me tremendous support, advice and guidance for my study, research, and career development. I am indebted for all the countless hours and tireless effort he has spent for me. He has set a perfect example I can follow through my career life: to work hard, be kind and wise, and become an insightful scholar and a successful leader as he is.

I would like to thank my committee members Professor Mustaque Ahamad, Dr. Greg Eisenhauer, Dr. Dejan Milojicic, and Professor Calton Pu for their constructive feedbacks of my work that have greatly improved the quality of this thesis. Special thanks to Dejan Milojicic for his bountiful support and advice for our joint work with HP Labs, which became one of the core parts of the thesis.

I feel privileged to have had the opportunities to work with a number of experienced professors and scholars: Professor Constantine Dovrolis, Professor Jim Xu, Professor Brian Cooper, Professor Mostafa Ammar, Dr. Matt Wolf, Dr. Ada Gavrilovska, Professor Umakishore Ramachandran, and Professor Doug Blough.

I would like to thank all of my friends and colleagues at Georgia Tech, Yuan Chen, Hao Wu, Sandip Agarwala, Wenrui Zhao, Patrick Widener, Qi He, Yi Ma, Jiantao Kong, Mohammed Mansour, and Jinpeng Wei, just to mention a few.

I wish to thank my entire family for providing a loving environment for me. And most importantly, I want to thank my parents, Zhenfang Cai and Xiaowei Jin, for their endless love, understanding and gratuitous support for me. To them I dedicate this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This thesis addresses complex distributed systems such as distributed information flow systems that continuously acquire, manipulate and disseminate information across an enterprise's distributed sites and machines, and distributed server applications co-deployed in one or multiple shared data centers, each with different and dynamic performance/availability requirements. A basic requirement imposed on such systems is the need to provide timely and sustained/reliable delivery and processing of service requests. This remains a difficult task despite more than 30 years of progress in distributed computer connectivity, availability and reliability [8], for multiple reasons. These include the increasing complexity of enterprise scale computing infrastructure; the distributed nature of these systems which make them prone to failures, e.g., because of inevitable Heisenbugs in these complex distributed systems; the need to consider diverse and complex business objectives and policies, including risk preferences and attitudes in enterprise computing; the issues of performance and availability conflicts; the varying importance of sub-systems that compete for resources in an enterprise's IT infrastructure; and the best effort nature of resources like networks, which implies that resource availability itself can be an issue.

This thesis proposes a novel approach to business policy-driven, risk-based, automated availability management, which uses an automated decision engine to make availability decisions and meet business policies while optimizing overall system utility, uses utility theory to capture users' risk attitudes, and addresses the potentially conflicting business goals and resource demands in enterprise scale distributed systems. For critical and complex enterprise applications, since a key contributor to application utility is the time taken to recover from failures, we develop a novel proactive fault tolerance approach, which uses online methods for failure prediction to dynamically determine the acceptable amounts of

additional processing and communication resources to be used (i.e., costs) to attain certain levels of utility and acceptable delays in failure recovery. Since resource availability itself is often not guaranteed in typical shared enterprise IT environments, this thesis provides IQ-Paths with probabilistic service guarantee, to address the dynamic network behavior in realistic enterprise computing environment. The risk-based formulation is used as an effective way to link the operational guarantees expressed by utility and enforced by the PGOS algorithm with the higher level business objectives sought by end users.

In its totality, this thesis develops a novel availability management framework and methods for large-scale enterprise applications and systems, with the goal to provide different levels of performance/availability guarantees for multiple applications and sub-systems in a complex shared distributed computing infrastructure. More specifically, this thesis addresses the following problems. For data center environments, (1) how to provide availability management for applications and systems that vary in both resource requirements and in their importance to the enterprise, based both on operational level quantities and on business level objectives; (2) how to deal with managerial policies such as risk attitude; and (3) how to deal with the tradeoff between performance and availability, given limited resources in a typical data center. Since realistic business settings extend beyond single data centers, a second set of problems addressed in this thesis concerns predictable and reliable operation in wide area settings. For such systems, we explore (4) how to provide high availability in widely distributed operational systems with low cost fault tolerance mechanisms, and (5) how to provide probabilistic service guarantees given best effort network resources.

CHAPTER I

INTRODUCTION

1.1 Motivation

Modern enterprises rely critically on the timely and sustained/reliable delivery and processing of information, using distributed computing systems [35, 36, 53] resident in large data centers and/or spanning many such sites or endpoints connected via wide area networks. An important class of applications in this context is that of distributed information flows systems [35,53], which continuously acquire, manipulate, and disseminate information across an enterprise's distributed sites and machines. Another important class of applications is distributed server applications which are co-located in one or multiple shared data centers, with each of them having different performance/availability requirement which could also vary over time and competing with each other for the shared resources. For applications like these, two key goals are the dependable and timely delivery of information, despite potential hardware/software failures and/or fluctuations in underlying platform resources. Beyond the need to reliably deliver information, also important, of course are applications' abilities to carry out the computations necessary for services, commonly done in data center environments.

System failures can have dire consequences for the enterprise, including loss of productivity, unhappy customers, or serious financial implications. In fact, as reported, the average cost of downtime for financial companies is up to 6.5 million dollars per hour and the average cost of downtime for retail companies is hundreds of thousands of dollars (Table 1) [40]. This has resulted in a strong demand for enterprise distributed systems that are available almost continuously. At the same time, however, high availability often

contradicts the performance quest, partially because of the relatively high costs of fault tolerance services, particularly when they must support the sustained delivery of information 24 hours a day, 7 days a week.

Providing high availability in enterprise-scale distributed systems is complex for multiple reasons. First, the sizes and the distributed nature of these systems make them prone to failures. In a data center with hundreds to thousands of servers running thousands to tens of thousands of processes [27, 36], both software and hardware failures are frequent [66, 79]. Second, for large IT systems, improvements in availability must consider diverse and complex business objective and policies, with issues including tradeoffs in performance vs. availability, choices concerning different fault tolerance mechanism, and runtime changes in business priorities. Third, for applications with high data rates coupled with high processing demands, replication of all runtime data may not be a cost-effective option for attaining high availability. Fourth is the need for small or negligible recovery times, to limit losses to the enterprise. Fifth, based on our experience working with industry partners like Delta Air Lines and Worldspan, systems must be able to cope with both transient and non-transient failures (e.g., failures that will recur unless some root cause is addressed). Finally, different sub-systems in an enterprise's distributed infrastructure typically are of varying importance to the enterprise, but they also compete for the resources needed to provide required levels of performance and availability. In fact, with increased acceptance of service-oriented middleware, there is an increased need for being able to make global choices and tradeoffs between performance and availability, across different applications, subsystems, and software and system components.

The continued growth in scale and complexity of these applications and their IT infrastructures makes it difficult, if not impossible for human end users to continuously maintain and improve their availability. Figure 1 demonstrates a subset of a shared data center running multiple multi-tier applications in one company with which we have interacted. When the number of hosts are on the order of tens to hundreds, it may still be feasible to manually

fine tune applications running on these hosts and determine suitable policies and parameters for availability managements. However, this data center has over 10,000 hosts and over 20,000 applications that run on those hosts, which make it very inefficient, if not impossible, for administrators to correctly and efficiently manage the infrastructure. Furthermore, availability issues experienced or caused by a certain machine will have likely effects beyond the violation of a single application's service level agreements, potentially causing large losses in utility for other applications and for the data center's operator. Other factors contributing to management complexity include the diverse and complex business objectives and policies sought by different applications, and changing run-time environments as well as changing demand behavior. An important goal, therefore, is to automate availability management in large enterprise systems, both to provide higher levels of availability and performance and to reduce management cost [90].

Availability management must consider risk and risk attitudes [44]. An intuitive example of risk attitude policies given in 'A Research Agenda for Business-Driven Information Technology' [44] is the following: "given a choice between good performance and mediocre downtime, or mediocre performance and good downtime, I'd pick the second one (or some other customers will pick the first one)." Consider the Operational Information System (OIS) run by one of our industrial partners, Delta, a major U.S. airline. The OIS is responsible for tasks that range from facilitating passenger check-in, to baggage handling, to flight updates, and even supporting the website that allows online check-in and ticket sales. From an operational point of view, different components of the OIS exhibit different levels of risk. A flight positioning subsystem, using FAA inputs, for example, can tolerate some loss of state since the FAA feed periodically updates each flight's positions [35]. Conversely, the subsystem performing passenger check-in must be highly reliable. Another attributes of the system is that risk is not a static quantity. An OIS sub-component providing services to a flight ready to depart in 5 minutes is more critical, i.e., it is less risk-tolerant, than the sub-component serving a flight that is preparing for departure in 50

minutes, for instance. Quantifying such runtime changes in risk (criticality) would allow an enterprise to allocate more resources to a sub-component that is currently at a higher level of risk.

Table 1: Financial Impact of IT system failures [40]

Industry	**Business Operation**	**Average Cost per Hour of Downtime**
Financial	Brokerage operations	$6.5 million
Financial	Credit card/sales authorization	$2.6 million
Media	Pay-per-view television	$1.1 million
Retail	Home shopping (TV)	$113.0 thousand
Retail	Home catalog sales	$900 thousand
Transportation	Airline reservations	$89.5 thousand

1.2 Problem Statement

The main problem addressed by this thesis is availability management for large-scale enterprise applications and systems, so as to provide different levels of performance/availability guarantees for multiple applications and sub-systems in a shared distributed computing infrastructure. More specifically, this thesis addresses the following problems. For data center environments, (1) how to provide availability management for applications and systems that vary in both resource requirements and in their importance to the enterprise, based both on operational level quantities and on business level objectives; (2) how to deal with managerial policies such as risk attitude; and (3) how to deal with the tradeoff between performance and availability, given limited resource in a typical data center. Since realistic business settings extend beyond single data centers, a second set of problems addressed in this thesis concerns predictable and reliable operation in wide area settings. For such systems, we explore (4) how to provide high availability in widely distributed operational systems with low cost fault tolerance mechanisms, and (5) how to provide probabilistic service guarantees given best effort network resources.

The following section provides additional background and overview of current solution

approaches. This is followed by a description of the solution approaches pursued in this thesis.

1.3 Solution Approach – Self-Managing Systems

1.3.1 Autonomic Computing and Self-Healing

Autonomic computing or self-managing systems are motivated by the ever growing complexity of enterprise computing systems. Specific issues include the difficulty of managing these large-scale systems, the need to integrate heterogeneous environments into corporate-wide computing infrastructures, and current and future efforts to extend these infrastructures beyond company boundaries into the Internet. The idea of autonomic computing is to have systems manage themselves in accordance with high-level objectives stated by administrators [45], thereby extending the number of operations these administrators can perform, i.e., to use self-management to free administrators from routine system operation and maintenance tasks. The hope is to reduce management costs, improve management efficiency, and offer improved scalability. The four basic aspects of self-management are:

- Self-configuration of components that include servers, routers, databases, and other technologies, across different platforms potentially provided by different vendors;
- Self-Optimization to tune hundreds of nonlinear tuning parameters, in order to improve performance and efficiency.
- Self-Healing, which automatically detects software and hardware failures in complex systems, diagnoses causes, isolate failures, and recovers from them.
- Self-Protection defends against malicious attacks.

This thesis focuses on self-healing, with self-optimization addressed implicitly, in that performance optimization is closely related with availability management. More specifically, we propose a risk-based analysis and policy decision framework to automatically manage availability according to business level objectives and risk policies. The framework

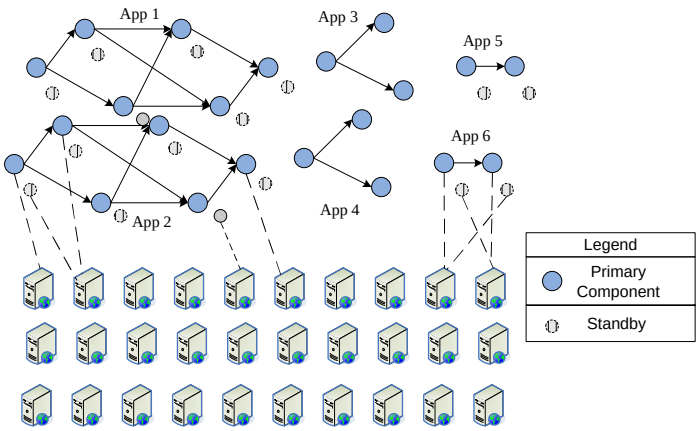


Figure 1: Shared Data Center

is implemented with a set of proactive fault-tolerance mechanisms, with novel methods to recover from both transient and non-transient failures, and with ways to optimize performance given specific availability requirements and policies to be maintained in shared computing infrastructures.

1.3.2 Performability, the Relationship of Performance and Availability

Performance and reliability are interrelated, particularly in modern enterprise systems. This is because these systems are typically *degradable*, meaning that they are able to continue to operate in the presence of faults or errors, at reduced levels of performance or quality of service, hence the term 'performability' [89]. The methods used for these purposes include active/passive standbys, check-pointing, logging, synchronization, and others [8].

Concerning performability, different users/customers will have different preferences about the tradeoff between performance and reliability. Consider again the example given by Kephart and White [44]: given a choice between good performance and mediocre downtime, or mediocre performance and good downtime, some customers will pick the latter, and others will pick the former. For example, in Figure 1, webserver App1 is used for brokerage operations services and requires very high availability (e.g., 'five nines'), but larger service times are acceptable. The internal enterprise search service App3 requires

lower response times, but 'five nines availability' is not necessary if that requires too many standbys given limited resources. Managerial attitudes like these can be related to different levels of risk attitudes or risk tolerances. Given the tradeoff between performance and downtime, low risk tolerance might choose lower downtime values, whereas high risk tolerance might choose higher performance. Rick tolerance changes over time too. For example, policy may directly prohibit patching App3 under conditions of high load, i.e., low risk tolerance. More generally, in the operational information systems like the one run by one of our industry partners [35], risk tolerance is an approximation of a combination of factors, including peak vs. non-peak operational time, proximity to delivery time for certain subsystem output, and others. Risk tolerance, therefore, is an aggregate measure of managerial policy.

The novel approach advocated in this thesis is to use risk tolerance to adjust system performance/reliability characteristics in response to different requirements and system changes. One specific case study considers patch application or service replacement for multi-tier web services. Another study considers the aforementioned aggregate measure of risk. For both cases, we demonstrate that entirely different tradeoffs are made for risk-averse vs. risk-tolerant scenarios, as explained in more detail next.

1.4 Solution Approach: Risk-Based Proactive Availability Management

This thesis enriches enterprise middleware with a new availability management framework and with new availability methods for applications operating on shared distributed computing and communication infrastructures. Specifically, we propose to develop risk-based 'proactive availability management' techniques that (1) consider performance and high-availability requirements in a uniform policy-driven automated management framework, to address potential conflicts in the realization of both, with consideration of users' risk attitudes and preference structures, (2) reduce the cost of high-availability with new low cost fault-tolerance services, and (3) implement proactive self-management methods so that

information flows can continuously and dynamically adjust their run-time fault-tolerance services to meet both their performance and availability requirements, according to current system stability level and resource constraints. (4) Another contribution of this work is the ability of middleware to distinguish the different information flows used in highly available information flow systems. A concrete example of the latter is to distinguish original from replicated information flow elements. By then differentially providing services to original vs. replicated flows, e.g., assigning different levels of importance and time constraints to each, middleware can extend performability guarantees to applications beyond those available in current systems.

The uniform framework for jointly considering performance and high-availability requirements used in this thesis employs a policy-based approach for managing both, the intent being to reduce or minimize the human engagement necessary while still complying with business level performance and availability objectives/policies. Business policies for availability management are formalized and operational policies are derived automatically based on utility objectives. Specifically, using utility functions, we capture tradeoffs between performance and availability, and tradeoffs between the use of different fault tolerance mechanism are captured with queuing system modeling and Markov chain availability analysis. Risk is formalized with von Neumann-Morgenstern utility theory, capturing users' risk attitude, resulting in risk attitude policies and risk-sensitive decision making (e.g., risk averse, risk neutral and risk seeking) that can be automated. The outcome is an automated decision engine for risk-based availability management.

This thesis also proposes and evaluates novel methods for reducing the costs of achieving the levels of availability desired by different users. Toward that end, we extend the known active-passive approach to reliable operation for distributed applications with new methods that can dynamically tune the tradeoff experienced across normal operations cost vs. recovery time. In particular, in this approach, the passive replica will be periodically refreshed with checkpoints. These checkpoints transfer the current state from the active

node to the passive node (passive standby). If the passive replica has been recently brought up to date by a soft-checkpoint, then recovery will be relatively fast. The resulting tradeoff between the cost of checkpointing and recovery delay is tuned by changing the frequency at which soft-checkpoints are transmitted during normal operation. Such tuning is based on user-provided expressions of information utility, and it takes advantage of runtime methods for failure prediction and dynamic availability self-management, hence the term 'proactive availability management'. With proactivity, therefore, it becomes possible to manage a system that experiences different levels of stability during its execution (e.g., a heavy memory load could mean an imminent failure). The prerequisite for such proactive management is that the 'current stability' of the system can be quantified, in order to then increase or decrease the resources expended to ensure desired levels of availability.

As stated earlier, a specific class of enterprise applications addressed by our work is Operational Information Systems. For the data transfers, i.e., the information flow services, required by such wide area applications, this thesis presents and experiments with middleware-based management methods. These methods permit us to differentiate how certain information flow services are realized, via the 'IQ-Paths' overlay network management and flow mapping and scheduling techniques. IQ-Paths implements self-regulating overlay streams, by assessing, predicting, judiciously using available network paths, and dynamically choosing alternate or exploiting concurrent paths. Self-regulation is based on (1) the dynamic and continuous assessment of the quality of each overlay path, (2) the use of online network monitoring and statistical analyses that provide probabilistic guarantees about available path bandwidth, loss rate, and RTT, and (3) self-management, via an efficient packet routing and scheduling algorithm that dynamically schedules data packets to different overlay paths in accordance with their available bandwidths. IQ-Paths offers probabilistic guarantees for application-level specifications of stream utility, based on statistical predictions of available network bandwidth. This affords applications with the ability to distinguish reliability-preserving from original data flows, and/or to send critical data

across overlay paths that offer strong guarantees for future bandwidth vs. less important data across less guaranteed paths. Further, demonstrating the generality of the IQ-Paths methods, they are also used to (1) help a data-driven interactive high performance code meet its user-defined utility requirements and (2) implement a more efficient version of the popular Grid-FTP application.

1.5 Thesis Statement

Enterprise applications exhibit wide variations in their performance and availability requirements, expressed implicitly by variations in workload and use and/or explicitly by differences in their service level agreements. The resulting challenges in managing these applications are compounded by their use of shared computational and network resources. Our thesis is twofold: (1) the effective online management of distributed enterprise applications running on shared IT infrastructures requires the combined use of multiple models of application and infrastructure behavior, including performance, availability, and resource usage models, and (2) a risk-based method based on utility theory constitutes an effective approach to combining these multiple models and to linking higher level (business) objectives to operational requirements and policies.

1.6 Thesis Contributions

The first set of thesis contributions concerns effective online management in shared data center settings.

Business Policy-Driven Automated Availability Management: Leveraging systems' performance / availability (i.e., performability) tradeoffs, we develop a novel policy-driven approach to address the potentially conflicting resource demands of these requirements. The approach uses utility theory to capture users' risk attitudes and then develops an automated decision engine that makes performability tradeoffs to meet those attitudes while optimizing overall system utility. In this fashion, IT users can

adjust system performance and availability to the risks tolerable by current business objectives. The approach is demonstrated with representative business applications on shared datacenter IT resources.

Utility-Driven Proactive Availability Management: For the critical enterprise applications considered in this thesis, a key contributor to application utility is the time taken to recover from failures, measured as Mean Time to Recovery (MTTR) or expressed by a Recovery Time Objective (RTO). Recovery techniques like disk-based logging [37] exhibit low runtime overheads but have large recovery times. Active replicas exhibit low recovery times but have high runtime costs [68]. Active-passive pairs can reduce runtime costs and offer suitable recovery times, but do not link these to current business objectives and/or incidence of failure. To address these issues, we develop a hybrid approach, termed proactive fault tolerance, which uses online methods for failure prediction to dynamically determine the acceptable amounts of additional processing and communication resources to be used (i.e., costs) to attain certain levels of utility and acceptable delays in failure recovery.

The second set of thesis contributions extends management techniques beyond single data centers into the distributed computing and communication infrastructures commonly used by modern service-oriented systems.

Probabilistic Guarantees for Network Services: A specific issue faced by distributed enterprise applications is how to provide service guarantees across potentially unreliable best effort network infrastructures. Here, the presence of dynamic network behavior and of multiple available network paths make it imperative for enterprise middleware to assist end user applications in best utilizing the available network resource. More specifically, middleware can help by providing to applications different levels of guarantees for data movement. A specific instance in which such guarantees are useful is when moving an application's original data vs. replicated data, the latter

done to meet certain reliability goals. The IQ-Paths approach to self-regulating data streaming across shared network infrastructures offers such guarantees. IQ-Paths dynamically measures and then, also statistically predicts the available bandwidth profiles on network links. Using these methods, IQ-Paths then automates the movement of data traffic across different overlay paths. This includes splitting a single data stream across multiple paths to improve performance through parallelism, and to improve desired end-to-end behavior by dynamically differentiating the amounts and kinds of data traffic imposed onto different paths. Self-regulating data movement and differentiation use a dynamic packet scheduling algorithm named PGOS that automatically maps packets to paths to match application-level utility specifications.

Risk-based Management: A risk-based formulation is again shown to be an effective way to link the operational guarantees expressed by utility and enforced by the PGOS algorithm with the higher level business objectives sought by end users.

1.7 Thesis Organization

The remainder of the thesis is organized as follows. We discuss the risk-based availability management framework and its methods for self-optimization of both performance and availability in Chapter 2. For critical and complex enterprise applications, since a key contributor to application utility is the time taken to recover from failures, efficient proactive fault tolerance mechanisms able to reduce run-time cost and improve recovery time are discussed in Chapter 3. Chapter 4 proposes the IQ-Paths approach with probabilistic service guarantee, to address the dynamic network behavior in realistic enterprise computing environment, and provide better network resource availability and network service guarantee. The risk-based formulation is used as an effective way to link the operational guarantees expressed by utility and enforced by the PGOS algorithm, with the higher level business objectives sought by end users. Related work in availability management is described in Chapter 5, followed by Conclusions and Future Work.

CHAPTER II

RISK-BASED AUTOMATED AVAILABILITY MANAGEMENT DRIVEN BY BUSINESS POLICIES

2.1 Introduction

As businesses are increasingly dependent on their IT environments for critical business function, IT service management solution driven by business policies (including objectives policies) is taking on a crucial role. In IT service management driven by business policies, the underlying IT systems are designed to maximize the business values of the services offered by IT and to continuously change as business needs change. Policy-driven service management helps reduce IT management cost and keeps the service management aligned with business objectives. A key goal of IT service management driven by business policies is to use business policy to guide resource management, and allocations in the IT infrastructures are used to carry out business tasks. In the datacenter environment, for instance, business policy has been shown important for guiding provisioning for the different applications that share the center's computing resources [71]. In the high performance domain, batch schedulers routinely use high level policies to determine the allocation of parallel machines to applications [31]. In operational information systems, tradeoffs exist with respect to the performance vs. reliability of business applications, recovery time (MTTR) being a key metric.

While most of the previous research focuses on performance/resource management, little work has addressed the area of availability management driven by business policies. This is a critically important task in enterprise IT, because (1) a single failure in enterprise IT could cause large business loss and policy violations [40], (2) the interrelation between performance and availability implies that policies for both must be considered jointly [89],

and (3) decision making is often risk-sensitive [44, 97]. It is still unclear how we can automate availability management in a highly dynamic and complex system according to business level objectives for performance and risk attitude/preference. As a consequence, users cannot manage the availability/performance ratio to match their risk tolerance.

In this chapter, we propose a policy-driven approach to automating run-time availability management in IT systems, according to high level availability and performance objectives [14]. We further apply von Neumann-Morgenstern utility theory to deal with users' risk attitude and preference. Based on the proposed approach, we implement an automated decision engine for availability management. The initial evaluation of the solution illustrates the significance of the policy-driven approach and it demonstrates its applicability for availability management in complex IT environments. In this fashion, IT users can customize their availability to the risks tolerable by business objectives.

The research presented in this work makes several contributions to the domain of IT management driven by business policies:

- rigorous methods and an associated management framework relate business objectives to the IT services that implement certain business tasks and to the performance implications of provisioning changes for IT services - performance model and availability model;

- an automated decision engine continuously optimizes IT availability management and controls service provisioning based on expected utility value in order to maximize high level business objectives;

- a new *risk-based* formulation of business policy combines the Service Level Agreements (SLAs) used in prior work with notions of risk tolerance to better capture current operational needs and requirements; and

- novel management methods based on risk tolerance and SLAs are shown useful for runtime guidance and control of performance/reliability tradeoffs in two different

business environments: (1) multi-tier business applications and (2) operational information systems.

Our focus on reliability is driven by multiple facts. First, failures are a key threat to enterprise systems, since they can result in unacceptable levels of service unavailability and lead to substantial revenue loss [91].

Second, handling failures manually is both a difficult and error-prone task, constituting a strong motivation for automating system management. This complexity is evident both for the datacenter environment [36] evaluated in this work and the complex, distributed service-based systems now being developed and deployed in industry [47]. Complexity derives from several factors. First, modern enterprise services typically comprise multiple systems and sub services, and often interact with other services. In addition, the growing scale of modern enterprise services (e.g., tens of thousands of services instances running in HP's consolidated data center) make manual availability management almost impossible. At the same time, the business objectives or policies are diverse and complicated, typically involving a combination of performance, availability, security, and other aspects. While the operational policies are the actual policies used in practice to manage IT services, instead of business policy, it is hard to assess the business value of service management using operational policies, and align them to high level business objectives and policies. Finally, fault tolerance mechanisms (e.g., active standby and passive standby) themselves are complex and different mechanisms are applicable under different situations since they may offer different levels of reliability, recovery times, as well incur different overhead or performance penalty.

Third, there are some well-known causes of failures, one being increased failure rates under high loads, another being failures caused by external interventions such as the application of system patches or change of configurations [65]. Since they occur frequently in large-scale enterprise data centers, automated management is strongly indicated to improve service availability in these situations.

Fourth, it is possible to directly relate managerial attitudes concerning failures to different levels of risk tolerance. Given the tradeoff between performance and downtime, low risk tolerance might choose lower downtime values, whereas high risk tolerance might choose higher performance. For example, policy may directly prohibit patching under conditions of high load - i.e., low risk tolerance. More generally, in the operational information systems run by one of our industry partners [35], risk tolerance is an approximation of a combination of factors, including peak vs. non-peak operational time, proximity to delivery time for certain subsystem output, and others. Risk tolerance, therefore, is an aggregate measure of managerial policy. Specific research results presented in this work use risk tolerance to adjust system performance/reliability characteristics in response to system changes. One case considers patch application or service replacement for multi-tier web services. Another case considers the aforementioned aggregate measure of risk. Results demonstrate entirely different tradeoffs made for risk-averse vs. risk-tolerant scenarios. They also show the effects of different risk vs. utility curves.

The remainder of this chapter is organized as follows. Section 2.2 motivates this research, emphasizing importance of availability management in enterprise computing. In Section 2.3, we present our approach, including the framework, models and automated decision engine. Section 2.4 discusses how to apply utility theory to deal with users' risk attitude. Experimental evaluations and conclusion with lessons learned, are given in Section 2.5 and 2.6.

2.2 Motivating Example

Failures in enterprise systems [35, 36] can result in substantial revenue loss. For example, it is reported that the average cost per hour of downtime for financial organizations can be up to 6.5 million US dollars, and for retail systems such as the home shopping industry, the cost can be up to 113 thousand US dollars per hour of downtime [91]. The continued growth in scale and complexity of these applications and their IT infrastructures,

however, make it difficult, if not impossible for human end users to continuously maintain and improve their availability. Factors contributing to this difficulty include the diverse and complex business objectives and policies, and changing run-time environments as well as changing demand behaviors. An important goal, therefore, is to automate availability management in large enterprise systems, both to provide higher levels of availability and performance and to reduce management cost [90].

Availability management must consider risk and risk attitude policies [44]. One intuitive example of risk attitude policies given in [44] is, "given a choice between good performance and mediocre downtime, or mediocre performance and good downtime, I'd pick the second one (or some other customers will pick the first one)." Consider the Operational Information System (OIS) run by one of our industrial partners, a major U.S. airline. OIS is responsible for a wide array of tasks that range from facilitating passengers check-in, to baggage handling, to flight updates, and even supporting the website that allows online check-in, and ticket sales. From an operational point of view, different components of the OIS exhibit different levels of risk. A flight positioning subsystem, using FAA inputs, for example, can tolerate some loss of state since the FAA feed periodically updates each flight's positions [35]. Conversely, the subsystem performing passenger check-in must be highly reliable. At the same time, risk is not a static quantity. An OIS sub-component providing services to a flight ready to depart in 5 minutes is more critical, i.e., it is less risk-tolerant, than the sub-component serving a flight that is preparing for departure in 50 minutes, for instance. Quantifying such runtime changes in risk (criticality) would allow an enterprise to allocate more resources to a sub-component that is currently at a higher level of risk.

Knowledge about risk can guide management of subsystems as well as that of individual components. An OIS example is perceived risk for the web server used to book flights, perform check-ins, etc. The two important attributes for this server are its response-time and availability. Unfortunately, when resources are constrained, improved availability through

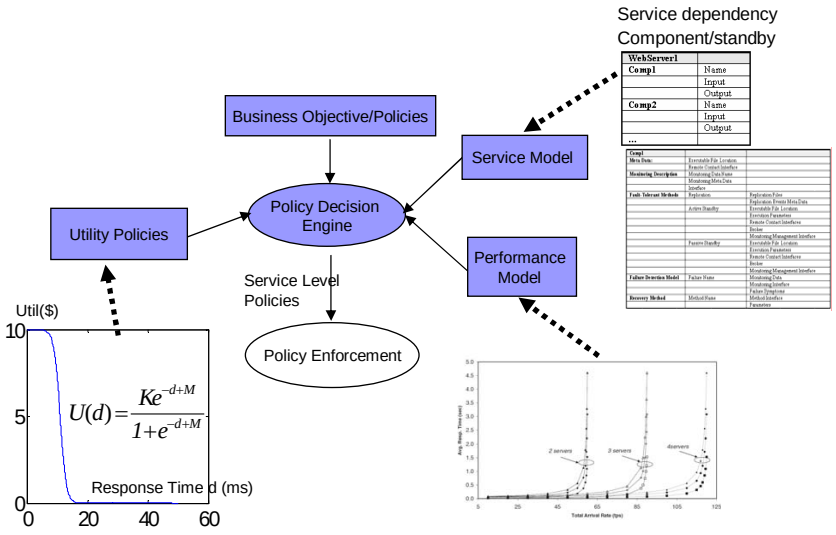


Figure 2: Availability Management Framework

methods such as active/passive standbys typically implies increased response-times. The resulting tradeoff in availability vs. response-time must be guided by its effect on the users (potential customers) accessing the web-site. Risk tolerance, i.e., the willingness to tolerate risk, is high, for instance, when loads are high. In this case, as in other enterprise systems, fault-tolerance modulated based on currently perceived risk, can exploit the fact that risk may be quantified as potential loss of revenue. A high risk state is one in which a failure in that state would cause a high loss of revenue or the amount of potential revenue loss is high. As a result, enterprise policy might dictate a preference for higher availability in such conditions.

2.3 Availability Management Driven by Business Policies

2.3.1 Framework

Our availability management leverages a policy-driven approach to automate the run-time availability management in IT systems, according to high level availability and performance objectives. The framework is shown in Figure 2. Business objective is the business level metrics including utility being maximized. The business policies specify a set of

business level regulations. Examples of such policies are:

1. Customer with revenue greater than $100K/Year should be classified as gold customers (Class 1 customers).

2. Services for gold customers should have at least four-nines availability.

3. Services for bronze customers should not consume more than 20% of the utility data center (UDC) resources.

While business policies are the high level policies to which automated service management should adhere, the operational policies are the actual refined policies used to manage service availability at run time. One typical example of an operational policies in the availability management domain is to allocate a passive standby to every second instance of the second tier, and to checkpoint every 10 minutes.

To 'map' the business policies to operational policies, a set of service models is used to specify the application topology, dependency between different sub systems or tiers, and setup information of services and their components. The service model is used for availability modeling, performance modeling, operational policies generation, and cascading failures.

A set of utility policies, which can also be viewed as part of the business policies (sometimes they are considered as business objectives, as they dictate customer satisfactions), express preferences for a variety of performance metrics, as well as metrics describing availability, security, and any other service attributes of interest. Often, utility function maps the performance metrics (e.g., response time) to monetary measurement (e.g., revenue). A widely used exponential utility functions is in the form of:

$$U(d) = \frac{Ke^{-d+M}}{1 + e^{-d+M}},$$

where d is the response time of the service requests and K and M is are constants which are specific to different services offered in the data center. The good property of this utility

function family is that it reasonably well models the common business requirement for service response time, i.e., when the response time is less than a threshold M, the utility is almost constant (although it still decreases very slowly when the response time increases), and when the response time is higher than the acceptable threshold, when it is considered the response time is unacceptable, the utility will drop very quickly to almost 0. Other utility functions can also be used, including discrete utility functions. Note that different services for different customers will have different constant and the utility function could change over time. One concrete example in Delta Airline's IT infrastructure is that one sub-system requires high performance (low response time) during a certain period of the day (from Midnight to early in the morning), and it has much lower requirement on the performance during other times. Our methodology can accommodate this kind of utility policies which change over time.

The performance profile is used to estimate the relationship between performance and resource allocation, under current and predicted work load. Service model is used to estimate the availability of various services using different fault-tolerance methods, including active standby, passive standby, and proactive standby [13].

Based on business policies, objectives, service model, and performance profiles, the policy decision engine (see Figure 3) first obtains the performance model and availability model, then forms a Mixed-Integer Programming (MIP) problem automatically which optimizes the service availability management according to high level business and utility policies, by finding the suitable availability operational policies. Many efficient MIP solvers can be used to solve this MIP problem, and we use GAMS with CPLEX [34]. How the decision engine obtains the performance/availability models and optimizes availability management is discussed in the next two subsections.

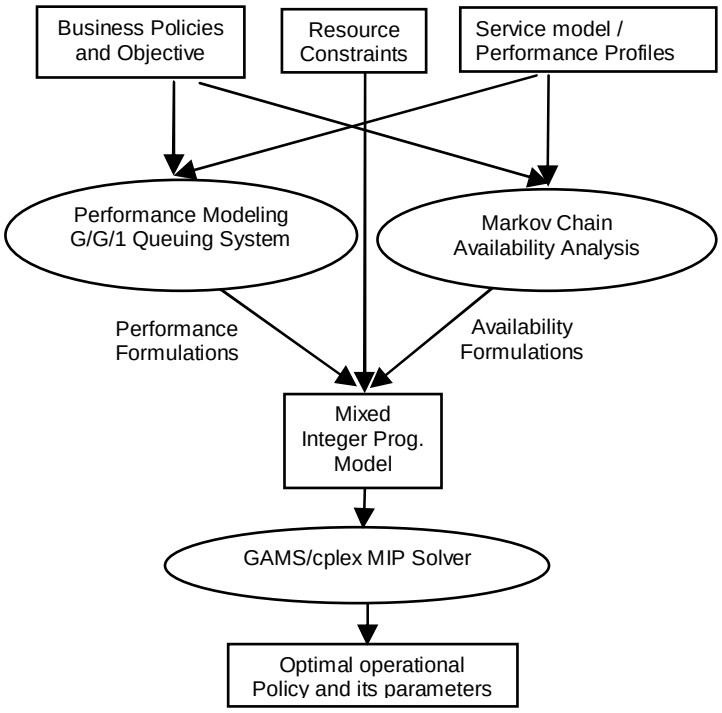


Figure 3: Policy Decision Engine

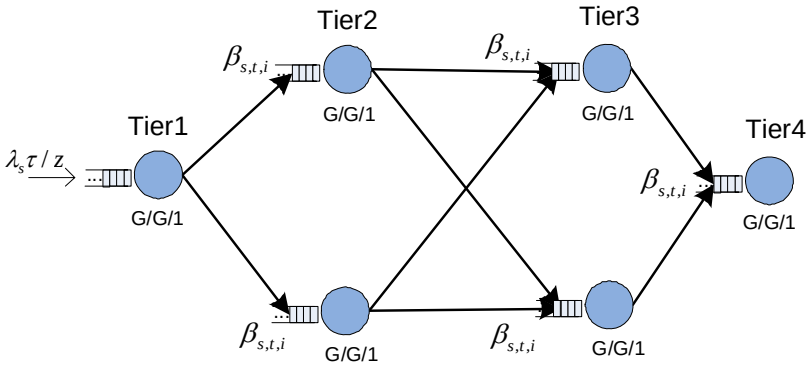


Figure 4: G/G/1Queuing System Modeling of Multi-Tier Applications

2.3.2 Performance Model

Although a session arrival process is often modeled as a Poisson process, this is not suitable for the request arrival rate for each component of a multi-tier application, as it largely depends on the load balancing/scheduling algorithms being used [80]. We therefore, use a G/G/1 queuing system (see Figure 4) for modeling the performance of this class of applications [95]. Let λ_s be the session arrival rate of application s, τ_s be the average session length, and $z_s$be the average session think-time. Then, by applying Little's Law, the arrival rate of each component can be calculated as:

$$\lambda_{s,t,i} = \beta_{s,t,i}\lambda_s\tau_s/z_s,$$

where $\beta_{s,t,i}$ is the component-specific constant for the *ith* component in tier t of application s. This constant depends on the load balancing and scheduling algorithm used in each tier and can be measured at run time.

Assume the response time of customers' request $d_s$is broken down into per-tier response times, then we have:

$$d_s = \sum_t (\sum_i \beta_{s,t,i} d_{s,t,i} / \sum_i \beta_{s,t,i}),$$

where $d_{s,t,i}$ is the average response time of each component, under its current workload and with its currently allocated resources.

To estimate $d_{s,t,i}$, we use a tight bound for G/G/1 waiting time:

$$w^u_{s,t,i} - \frac{1+\rho_{s,t,i}}{2\lambda_{s,t,i}} \leq w_{s,t,i} \leq w^u_{s,t,i},$$

where $\rho_{s,t,i} = \lambda_{s,t,i}S_{s,t,i}$, and $w^u_{s,t,i}$ is the upper bound of $w_{s,t,i}$:

$$w^u_{s,t,i} = \lambda_{s,t,i}(\sigma^2_{s,t,i} + \sigma'^2_{s,t,i})/(2 - 2\rho_{s,t,i}).$$

To see the tightness of the above bounds, we can convert this waiting time bounds to the bounds for the mean number of requests waiting in queue, and the difference between the latter lower and upper bounds is$\frac{1+\rho_{s,t,i}}{2}$, which is between 0.5 and 1 (events in the queue), because $0 < \rho_{s,t,i} < 1$ always holds.

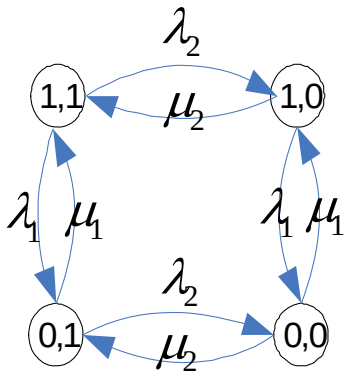


Figure 5: Markov Chain for a Simple Two-Component Application

Now the response time of each component can be estimated by:

$$d_{s,t,i} = (w^u_{s,t,i} + w^u_{s,t,i} - \frac{1+\rho_{s,t,i}}{2\lambda_{s,t,i}})/2 + S_{s,t,i}$$

Several parameters are needed in the above formulations. The average service time under some workload $\lambda_{s,t,i}$ with allocated resource $R_{s,t,i}$, $S_{s,t,i} = S_{s,t,i}(R_{s,t,i})$, is determined using off-line profiling (performance model profiling). The variation of service time $\sigma'^2_{s,t,i}$ is determined with a similar method. The variation of request inter-arrival time can be measured on-line. Currently we use a simple moving average predictor to predict the inter-arrival time variation in the next time slot, but more sophisticated predictors can be used.

2.3.3 Availability Model

The availability model is used to derive the probability P_s of the system when it is in state s, if P_s is not directly available. Each state s, $s = 1, 2, ...S$ is the state that one or several components of the system are down and the remaining components are functioning properly. There are several availability evaluation engines, such as Sharpe and Avanto [43] to derive P_s, either use traditional Markov Chain Model or simulation. We use the traditional Markov Chain Model. Figure 5 represents the Markov Chain Model of a two-tier application with only two components. For simplicity, we only consider software failures, as

software failures are usually the dominant failures in commercial utility data center [65], although the Markov Chain Model can be extended easily to consider the hardware failures. For limited space, we omit the formulations of Markov Chain analysis, referring interested readers to [32] for additional details.

The allocated standbys help increase the repair rate of the system under consideration. For example, active standbys can decrease the repair time to almost zero, while passive standbys reduce the repair time significantly by using checkpointing and recovery based on most recent checkpointed session states. At the same time, active and passive standbys require additional resources, thus causing performance penalties before failures. A specific resource considered in this work is the CPU, which is normally the primary bottleneck for multi-tier applications in utility data center [3]. We also assume that virtualization methods and Work Load Manager (WLM) (e.g., such as those used in HP's utility data center [36]) make it easy to adjust primary and standby resource allocations.

Specific causes of failures considered in our work are configuration changes and updates such as software patching. In web server applications, these are known to cause up to 40% of all failures [65]. For availability management, then, we model the change process as four phases: standby initialization, patching/change, failure detection, and recovery. During the second phase, the primary component is typically un-available if it is patched, but it may still provide service if its configuration is being changed. If failure occurs in the failure detection phase, recovery phase is initiated immediately. The resources required by the standbys in each phase are profiled off-line, to determine the resource available in the performance model [13], and the performance model determines the utility obtained in each phase.

2.3.4 Resource Allocation

Given a set of components, let $A_{s,t,i}$ be the 0-1 variable that equals to 1 if an active standby is to be allocated for component $C_{s,t,i}$ or 0 if no active standby is to be allocated for this

component. $P_{s,t,i}$ is 0-1 variable that equals to 1 if a passive standby is to be allocated for $C_{s,t,i}$. Finally, $N_{s,t,i}$ is 0-1 variable that equals to 1 if no standby is to be allocated for $C_{s,t,i}$. Then, we have the following constraints that state only one standby can be allocated for one component:

$$if \quad A_{s,t,i} = 1$$
$$then \quad \sum_{k} A_{s,t,i,k} = 1$$

$$if \quad P_{s,t,i} = 1$$
$$then \quad \sum_{k} P_{s,t,i,k} = 1$$

and

$$A_{s,t,i} + P_{s,t,i} + N_{s,t,i} = 1, \textit{for all} s, t, i.$$

The following constraints guarantee that one standby is allocated to only one host:

$$A_{s,t,i}(\sum_{k=1}^{H} A_{s,t,i,h}) = 1, \quad and$$
$$P_{s,t,i}(\sum_{k=1}^{H} P_{s,t,i,h}) = 1,$$

where $A_{s,t,i,h}$ and $P_{s,t,i,h}$ are 0-1 variable, and they are equal to 1 if active standby $A_{s,t,i}$ (or passive standby $P_{s,t,i}$) is allocated at host h. Sometimes data center administrators or customers have specific restrictions on where the standbys should be placed, for which the following additional constraints can be applied:

$$if \quad A_{s,t,i} = 1$$
$$then \quad A_{s,t,i,1} + A_{s,t,i,5} + ... = 1$$

The primaries and standbys allocated on one particular host can use up to 100% of the CPU resource. Therefore, resource allocation constraints can be formulated as:

$$if \quad A_{s,t,i,h} = 0$$
$$then \quad CPU_{h,s,t,i,a} = 0$$
$$\sum_{s,t,i,b} CPU_{h,s,t,i,b} = 1, \quad for \quad all \quad hosts \quad h$$

The objective function of this MIP problem is then:

$$Max(U) = Max(\sum_{s \in S} P_s U_s),$$

where U_s is the average utility in each possible state in the Markov Chain, and P_s is the limiting probability of the corresponding state s. This MIP formulation maximizes the expected utility of the data center. In the next section, we will consider the risk attitude policies under uncertainty, which result in a different objective function. All of the constraints of the new MIP problem are the same as the constraints in this MIP problem.

2.4 Risk-attitude Sensitive Availability Management

2.4.1 von Neumann-Morgenstern Utility Theory

Availability management usually involves uncertainty, and decision making in such circumstances should take into account a user's preference structure, that is, how a user compares different outcomes of his or her decisions. In terms of availability management, how differently the outcome with failure and the outcome with no failure are valued implies the decision maker's preference structure or risk attitude, e.g., good performance and mediocre reliability versus mediocre performance and good reliability. Different preference structure or risk attitudes will result in different decisions. Utility theory developed by von Neumann and Morgenstern can be used to deal with such decisions under uncertainly. We will informally describe this theory below. A comprehensive introduction of this theory can be found in [97].

Let W be a set of possible outcomes of lotteries and w_i is one outcome in terms of money. A lottery L is defined as $\{(w_1, P_1), (w_2, P_2), \ldots (w_n, P_n)\}$. Where P_i is the probability that outcome w_i would happen. The von Neumann and Morgenstern theory suggests that if the preference structure satisfies certain primitive axioms, then L_1 is preferred over L_2 if and only if $vNMU(L_1) > vNMU(L_2)$, where $vNUM(L) = \sum_{i=1}^{n} P_i \cdot vNMU(w_i)$ is the expected *von Neumann-Morgenstern utility* of L.

In other words, a lottery is preferred over another one if and only if the preferred lottery has a lager expected utility. Based on this theory, a decision making becomes a procedure to find an alternative with maximal expected utility. The utility function *vNMU* is the value of a monotonically increasing function of the wealth level w. The function intuitively reflects how happy the decision maker is with its current wealth level. Consider a utility function, U defined over wealth w. Let $MU(w) = dU(w)/dw$. For everyone, regardless of their attitude, it is natural to assume that $MU(w) > 0$ since a person's utility always increases in the amount of wealth that he has. We can then determine a person's risk attitude through their marginal utility function ***MU(w)*** as follows.

$1) dMU(w)/dw > 0 \Rightarrow risk - seaking.$ In other words, the additional utility he gets from one more dollar is larger when he already has a larger amount of initial wealth. Loosely speaking, a risk-seeking person cares more about the upside potential than the downside risk.

$2) dMU(w)/dw < 0 \Rightarrow risk - averse.$ In other words, the additional utility he gets from one dollar becomes less with the increase of his wealth. Loosely speaking, a risk-averse person cares more about the downside risk than upside potential.

$3) dMU(w)/dw = 0 \Rightarrow risk - neutral.$ In other works, the additional utility is independent of the wealth level. Loosely, a risk-averse person cares equally about downside risk and upside potential.

Risk attitudes explain why people buy insurance even though the insurance premium is usually much larger than the expected loss from the insurance cause (i.e., risk-averse), and also why people buy lottery even though the money spent on lottery is usually much larger than the expected lottery prize (i.e., risk-seeking).

Exponential utility functions are one of the most commonly used type of risk-sensitive utility functions. In this work, we consider a very popular exponential utility function in economics: Constant Relative Risk Aversion (CRRA) von Neumann-Morgenstern (vNM) utility function.

2.4.2 Using Risk Formulations in Availability Management

Next, we use an example to discuss how to apply the utility theory described above to deal with uncertainty and user's preferences in availability management. Our approach uses risk-sensitive utility function such as CRRA to capture users' preferences and then optimize availability management by maximizing the expected utility.

Table 2: Lottery Outcomes

Options	Performance	Availability
$L1$: (pP,hA)	55% of Max	99.999%
$L2$: (hP,pA)	99.5%	97.25%
$L3$: (gP,gA)	92%	99.95%

Table 3: Preferences of Outcomes

Customer	Preference
$C1$: ('like high availability')	*L1>L3>L2*
$C2$: ('like high performance')	*L2>L3>L1*
$C3$: ('mediocre perf. and avail.')	*L3> L1>L2*

To manage availability according to a user's preference structure, we first need to find the user's risk attitude under uncertainty. In our example, we use three typical possible outcomes ('lottery outcomes' in game theory) to elicit a user's risk attitudes (Table 1). Intuitively, $L1$ represents an outcome with very high availability (five nines) and with poor performance, $L2$ represents an outcome with very high performance with poor availability, and $L3$ represents an outcome with mediocre performance and mediocre availability. Different customers have different risk attitudes, for example, customer $C1$ (Table 2) prefers $L1$ to$L3$, and prefers $L3$ to $L2$, since he likes really high availability (very risk-averse), while customer $C2$ prefers $L2$ to $L3$, and prefers $L3$ to $L1$. Their different risk attitudes are captured by the CRRA vNM utility function, which has the form of:

$$vNMU(L_i) = \frac{PU^{\alpha}}{\alpha},$$

where U is the performance outcome (e.g., 55%), and P is the probability that the performance outcome could happen (e.g., 99.999%). It's straightforward to show (1) $0 < \alpha < 1 \Rightarrow risk - averse$ and the smaller α is, the more risk averse the person is; (2) $\alpha = 1 \Rightarrow risk - neutral$; and (3) $\alpha > 1 \Rightarrow risk - seeking$, the bigger α is, the more risk-seeking the person is. To estimate the value of α for $C1$, we use the outcome preference of this user, i.e., *L1>L3> L2,* to find the estimated range of α. Given additional preferences of the customer, we can further narrow down the range, and when the range is narrow enough, the average of the upper and lower bounds is sufficiently good to represent the user's risk attitude [97]. There exist other techniques to fix the value ofα, if that is preferred. One technique is to find the equivalent outcomes, which states that two outcomes make no difference to the customer. It results in an equation with α as the single variable to be determined. Another technique is to find the limit of the user's risk attitude. For example, for customer $C1$, if he states further that $L1$'s five-nines availability is good enough for him, and other outcomes with even lower performance and higher availability are not as attractive to him as outcome $L1$, then the lower bound of α we found previously (which is 0.001) is the value that represents this customer's risk attitude. Similarly, we found thatα for customer $C2$ is 0.37. The α for customer $C3$ is estimated at 0.03, by averaging the lower and upper bounds derived by the preferences stated in Table 2 (we can refine the α for customer $C3$ by asking more preference questions, but the lower and upper bounds turn out to be sufficient to differentiate the outcomes in the experiments). To manage availability according to customer's risk attitude under uncertainty, we simply replace the objective function of the previous MIP problem with the new objective function, with all the constraints presented in Section 2.3:

$$Max(vNMU) = Max(\sum_{s \in S} \frac{P_s U_s^{\alpha}}{\alpha}).$$

Note that the risk-formulation discussed in this section is more general than the multi-tier applications in shared data centers example used in this Chapter. In fact, this risk-based approach can be used in many other situations, when risk or uncertainty is unavoidable.

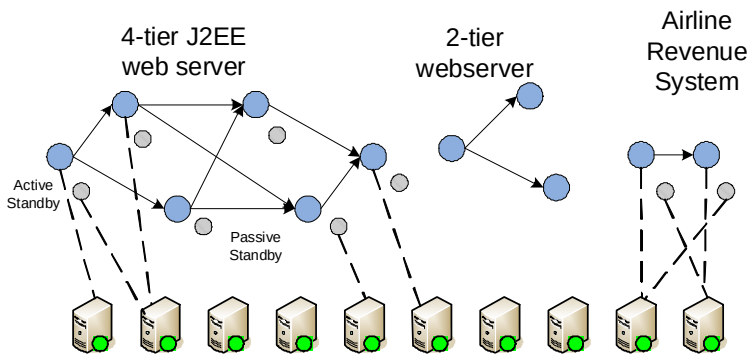


Figure 6: Example Services in a Utility Data Center

For such situation, the methods to find the probability P_s a system is in state s) and the utility of this state U_s could be different. In Chapter 4, we present more details on how to use the risk-based approach for the network resource availability issues.

2.5 Experiments

We evaluate our approach in a utility data center scenario, as shown in Figure 6, with three multi-tier applications. The first one is a 4-tier J2EE web server application with a load balancer tier, Apache HTTP server tier, Tomcat Servlet server tier, and MySQL database server tier. The second application is a 2-tier web server with one load balancer tier and one Apache HTTP server tier. The third application is a two-tier airline revenue sub-system.

Experiments are conducted to validate our approach in two aspects. The first experiment concerns availability management during change [65]. The second experiment is availability management according to customers' risk attitudes. To simulate failures, we use a trace-based queuing system simulator, which has as inputs the user request arrival time traces for each server and the service time traces for each component as input. The simulation engine is similar to the simulation engine used by Janakiraman et al. [43]. In our work, each component processes the request according to the time logged in the service time traces, while the simulation engine used in their work is based on estimated service

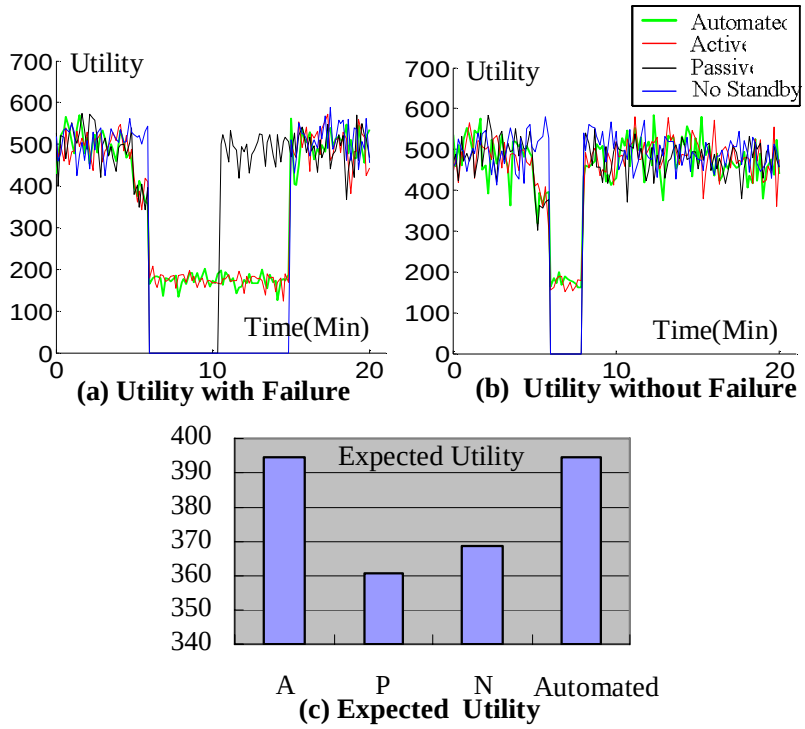


Figure 7: Utility during Patching Component $C_{1,1,1}$. Patch/change starts at t=5min, and ends at 15min. Graph (a) is the utility when failure occurs, (b) is the utility when failure doesn't occur, and (c) is the expected utility using active standby (denoted by A), passive standby (P), no standby (N), and automated standby configuration.

time distribution. The load balancing algorithm for each tier is the widely used round-robin algorithm (e.g., used in Linux Virtual Servers (LVS) [108]).

2.5.1 Availability Management During Change

To illustrate the importance of automated availability management for IT infrastructures, and to show how to optimize availability configurations to maximize expected utility, we introduce three different change scenarios in the first experiment: (1) patching component $C_{1,1,1}$ (Server 1, Tier1, Component 1, and the probability that the component will fail after patch is 0.1, (2) patching component $C_{1,2,2}$ and the probability that the component will fail after patch is 0.1, and (3) changing the configuration of component $C_{1,1,1}$, and because of the risk of this change, the probability of failure is 0.7. The results are illustrated in

Figure 8: Utility during Patching Component $C_{1,2,2}$.

Figure 7 to Figure 9. Figure 7 depicts the utility achieved by the J2EE web server under two different situations: patch applied and the new component failed vs. patch applied without resultant failure. The expected utility, which we want to maximize, is calculated by $P_f U_f + P_{nf} U_{nf}$, where P_f is the probability the new component will fail, and U_f is the average utility from the time patch is applied till the time the recovery is completed (20Mins). P_{nf} and U_{nf} are the probability the new component will not fail and the average utility in the same time period. Expected utilities under three possible configurations (active standby, passive standby, no standby) are shown in Figure 7(c). In this experiment, our availability management automatically determines the active standby is the optimal configuration in this situation.

While optimal in this scenario, active standbys are not always desirable. For example, in Figure 8, the active standby provides higher levels of availability than needed and therefore, it has the worst expected utility as compared with other two possible configurations.

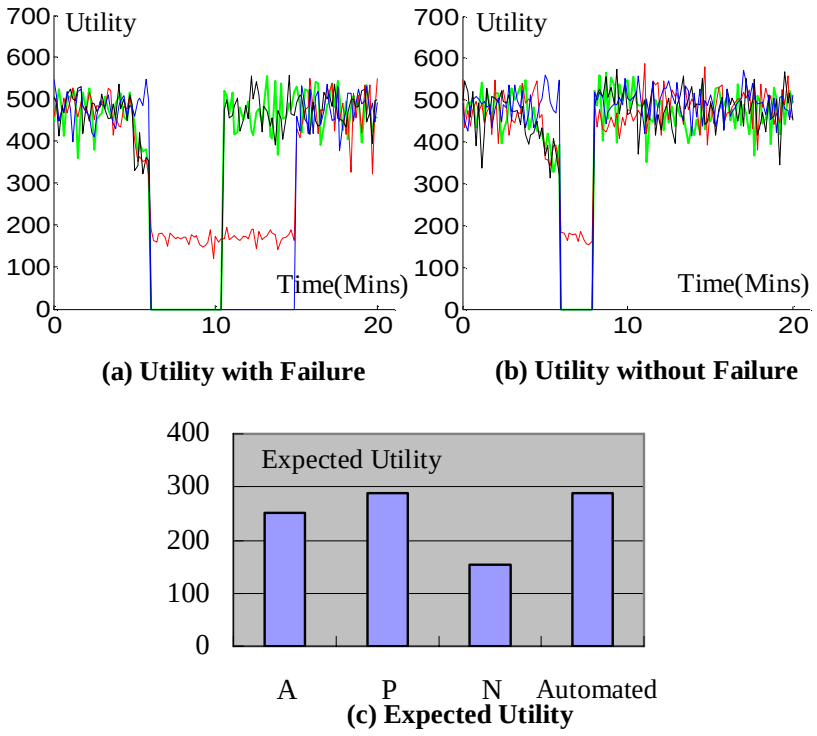


Figure 9: Utility during Reconfiguration of Component $C_{1,1,1}$.

The intuitive reason here is that Tier two has two replicated components and the failure probability that the patch will fail is low. In this case, it is better not to allocate standbys, in order to maximize the expected utility during change.

Again, while the no-standby configuration is the best configuration in this scenario, it is not the best configuration for the third scenario (actually it is the worst one among three configurations). Instead, the passive standby becomes the best configuration (see Figure 9). The insight gained from these experiments is that although current availability management either doesn't take any preventive procedures before change, or uses static/same configuration for different situations, one particular availability configuration can result in totally different behaviors in different situations, depending on many factors, including the failure probability, current workload, available resource, patch time, and recovery time, etc. Since one optimal (or close to optimal) configuration for one particular scenario could easily become the worst configuration in other scenario, it is important to automate availability

management so the appropriate configuration (operational policies) can be determined and executed automatically.

2.5.2 Risk-attitude Sensitive Availability Management

The second experiment deals with risk attitude policies, to validate that it is possible to optimize availability management according to customers' preference structure/risk attitudes. Here we consider availability management for the two-tier airline revenue application, with each tier of the application having one component (see the third application in Figure 6), for three different customers (Table 1 and Table 2). To illustrate the results, the configurations for each component are limited to (1) No standby, (2) Active, (3) Passive 3Min, (4) Passive 5Min, (5) Passive 15Min, and (6) Passive 30Min, resulting in a total of 36 possible configurations for the two-tier application. As discussed in the Section 2.4, the three customers $C1, C2,$ and $C3$ have different risk attitudes which are characterized by the CRRA vNM utility functions.

Results obtained by using different CRRA vNM utility functions for these three customers, and additional results using risk-neutral and risk-seeking vNM utility functions are given in Table 3. In this table, the first row is the value α. The first column is the utilities under five representative configurations. "A-A" means to allocate active standby for both components, and "P-P 3" means to allocate passive standby for both components, with checkpointing interval chosen to be 3Mins.

The configuration with higher vNM utility better matches customer's risk attitude. This means that the only thing that matters is the ordering vNM utility. For ease of comprehension, we convert the results from Table 3 to normalized vNM utility (orders of vNM utility) as shown in Figure 10. Experiment results show that the availability management is optimized according to the user's risk attitudes (preference structures). For example, for Customer 1, we use CRRA utility function with α=0.001, and active standbys are chosen for the two components in the server. Intuitively this is true, as customer $C1$ is very

risk averse and availability configuration A-A is the most conservative one. More accurately, the actual performance and availability of the application using this configuration is (54.9%, 99.995%), which is very close to the top choice of customer $C1$ ($L1$ in Table 1). Similarly, the performance and availability achieved for customer $C2$ and $C3$ are (92.5%, 99.7%) and (97.5%, 98.3%), which are also very close to their top preference, $L2$ and $L3$ respectively.

Another important observation is that different risk attitudes (different values of alpha for CRRA vNM utility function) result in entirely different configurations. Therefore, it is important to consider the users' risk attitude when managing availability. Existing policy-based methods only optimize expected utility, which is actually one special case in our framework. By optimizing the expected utility, we are actually treating all users as risk-neutral (α=1). The reality is however, most users are risk-averse (including customer $C2$ in our example who only requires one nine to two nines availability and 'seems' risk-seeking). If we simply optimize the expected utility, the resulted configuration could be significantly different from what the user actually expected.

Table 4: vNM Utility of Five Representative Configurations

	0.001	0.03	0.5	1	3	10
A-A	1000.1	33.4	2.09	1.10	0.44	0.25
P-P 3	997.2	33.8	2.71	1.85	2.09	46.58
P-P 15	983.6	33.4	2.74	1.92	2.42	77.74
N-P 15	933.3	32.1	2.66	1.88	2.47	91.45
N-N	902.8	30.7	2.55	1.80	2.39	91.89

2.6 Conclusion and Lesson Learnt

This chapter presents an approach to automating availability of management in IT systems driven by business policy. We implement a policy engine that dynamically optimizes expected utility according to high level availability and performance objectives. We further study how to apply utility theory such as von Neumann-Morgenstern utility function to

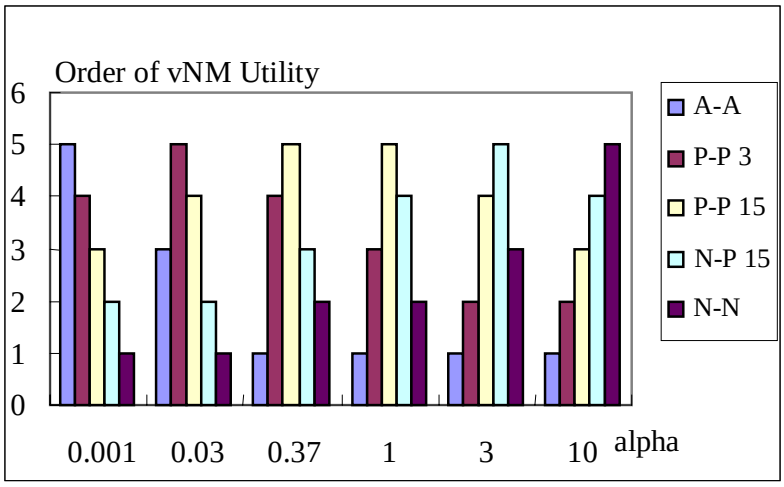


Figure 10: Normalized vNM Utility of Five Representative Configurations

deal with users' risk attitude and preference and enable users to customize their availability to the risks tolerable by business objectives. The evaluation of the proposed solution demonstrates that our approach is applicable for availability management of complex IT environments and the significant difference to use our approach to consider risk attitudes in both decisions made and the actual results.

Some of the lessons we learnt through this work are:

(1) Current policy specification standards, such as WS-Agreement, are relatively simple and SLAs are not sufficiently rich to capture the different functional and expression-based formulations needed in modern service-based applications and systems. For example, WS-Agreement can be used to express different valuations for configurations, however, only with discrete attributes. It lacks formal semantics to express preferences for a variety of performance metrics, availability, security, and any other service attributes of interest in terms of utility functions. Risk attitudes have not been studied in current SLA policy specification.

(2) Business objectives or policies may involve performance, availability, security and other aspects defined as traditional SLA or complex utility functions. Systems and IT services themselves are complicated, as well. Automated IT management solution should

support multiple classes of enterprise applications, and different utility formulations used in such applications. The key to success are accurate system models and optimization algorithms.

(3) Availability management involves uncertainty and the decision making under uncertainty often involves decision maker's risk attitude. Utility theory is a useful tool to deal with risk-sensitive policies in IT management.

CHAPTER III

UTILITY-DRIVEN PROACTIVE MANAGEMENT OF AVAILABILITY IN ENTERPRISE-SCALE INFORMATION FLOWS

3.1 Introduction

Modern enterprises rely critically on timely and sustained delivery of information. An important class of applications in this context is a company's distributed operational information system, which continuously acquires, manipulates, and disseminates information across the enterprise's distributed sites and machines. For applications like these, a key attribute is their availability, or in other words, the ability to reduce failures and ability to recover from failure fast, which are key contributors to application utility. As discussed before, system failures can have dire consequences, including loss of productivity, unhappy customers, or serious financial implications. The average cost of downtime for financial companies, as reported in [40], is up to 6.5 million dollars per hour and hundreds of thousands of dollars per hour for retail companies. This has resulted in a strong demand that these critical distributed systems should be available almost continuously.

Providing high availability in widely distributed operational information systems or distributed systems in large data center is complex for multiple reasons. First, because information flows, hosts, applications, and components are distributed, they are difficult to manage, and failures at any of a number of distributed components or sites can reduce availability. Second, multiple data flows may use the same distributed resources, thereby increasing the complexity of the system and the difficulty of managing and preventing failures. Third, such systems often have high data rates and/or intensive processing requirements, and there are frequently insufficient system resources to replicate all this data

and processing to achieve high reliability. Fourth, information flows must have negligible recovery times to limit losses to the enterprise. Finally, based on our experience working with industry partners like Delta Air Lines and Worldspan, systems must recover not only from transient failures but also from non-transient ones (e.g., failures that will recur unless some root cause is addressed) [35]. While Chapter 2 proposed the risk-based availability management framework and its methods for self-optimization of both performance and availability, how can we provide high availability for critical distributed systems and critical subsystems of enterprise IT services, given all of these requirements? Traditional techniques such as recovery from disk-based logs [37] may have recovery times that are unacceptable for the domain in question. Using active replicas [68] imposes high additional communication and processing overheads (since all data flow and processing is replicated) and therefore, may not be an economically viable option. Another option is to use an active-passive pair [68], where a passive replica of a component can be brought up to date by retransmitting messages that had gone to the failed, active one. This option reduces communication costs, since messages are only sent to the passive component at failure time. Unfortunately, this may result in long recovery times. A better solution would be a hybrid of the above approaches, accepting dynamically determined levels of additional processing and communication during normal operation in order to reduce recovery times when failures occur.

In this chapter, we extend the active-passive approach to dynamically and proactively tune the tradeoff between normal operation cost and recovery time [13, 16]. In particular, the passive replica will be periodically refreshed with 'soft-checkpoints': these checkpoints transfer the current state from the active node to the passive node (passive standby), but are not required for correctness (hence, 'soft-checkpoints'). If the passive replica has been recently brought up to date by a soft-checkpoint, then recovery will be relatively fast. The tradeoff between cost and recovery is tuned by changing the frequency at which

soft-checkpoints are transmitted during normal operation. Such tuning is based on user-provided expressions of information utility, and it takes advantage of the following methods for failure prediction and dynamic availability self-management:

- *Availability-Aware Self-Configuration* – a user-supplied per information flow 'benefit-function' drives the level of additional resources used to guarantee availability. This ensures preferential treatment of flows that offer more benefit to the enterprise, with the aim of maximizing benefit across the system.
- *Proactive Availability Management* – during its execution, a system may be at different levels of stability (e.g., a heavy memory load could mean an imminent failure). In many cases, the 'current stability' of the system can be quantified in order increase or decrease the resources expended to ensure desired levels of availability.
- *Handling Non-Transient Failures* – some failures will recur if the same sequence of messages that caused the failure is resent during recovery. In this case, we must use application-level knowledge to avoid fault recurrence. We present several techniques, based on real-world case studies, to deal with such faults.
- *Differentiate service guarantee for original vs. replicated flows* – the data transport middleware distinguishes the different information flows used in highly available information flow systems, e.g., the original information flows vs. replicated flow elements used for attaining high availability. By differentially providing services to original vs. replicated flows, assigning different levels of importance and time constraints to each, middleware can extend performance guarantees to applications beyond those available in current high-availability frameworks.

Proactive availability management techniques have been integrated into IFLOW, a high performance information flow middleware described in [49]. The outcome is a flexible, distributed middleware for running large-scale information flows and for managing their availability. In fact, experimentation shows that proactive availability management not only

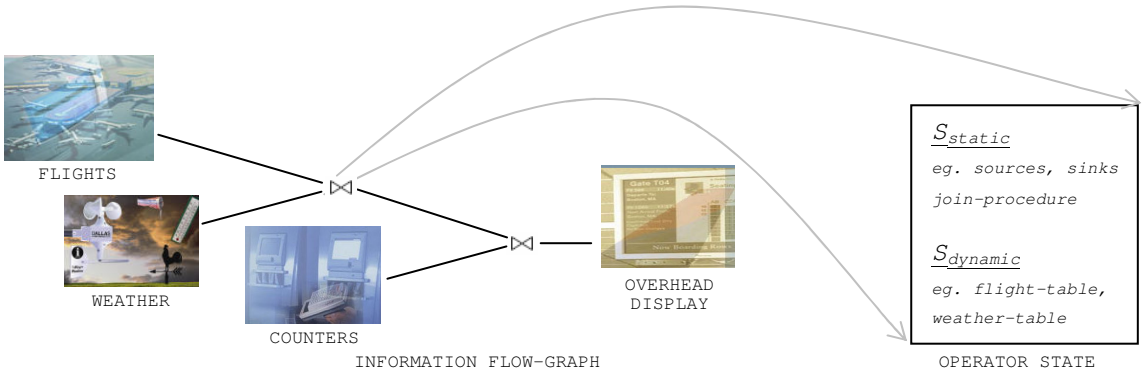


Figure 11: Information Flow-Graph and Operator State

imposes low additional communication and processing overheads on distributed information flows, but also, that proactive fault tolerance is an effective technique for recovering from failures, with a low recovery time of 2.5 seconds for an enterprise-scale information flow running on a representative distributed computing platform. Experiments further show that utility-based availability management offers 1.5 times the net-utility of the basic active replica approach.

3.1.1 Example: Operational Information System

An operational information system (OIS) [35] is a large-scale, distributed system that provides continuous support for a company or organization's daily operations. One example of such a system we have been studying is the OIS run by Delta Air Lines, which provides the company with up-to-date information about all of their flight operations, including crews, passengers and baggage. Delta's OIS combines three different sets of functionality:

- *Continuous data capture* – for information like crew dispositions, passengers, airplanes and their current locations determined from FAA radar data.
- *Continuous status updates* – for low-end devices like airport flight displays, for the PCs used by gate agents, and even for large databases in which operational state changes are recorded for logging purposes.

- *Responses to client requests* – an OIS must also respond to explicit client requests, such as pulling up information regarding a particular passenger, and it may generate additional updates for events like changes in flights, crews or passengers.

In this thesis, we model the information acquisition, manipulation, and dissemination done by such an OIS as an information flow graph (a sample flow-graph is shown in Figure 11). We then present techniques, based on this flow-graph formalization, to proactively manage OIS availability such that the net-utility achieved by the system is maximized. This is done by assigning per information flow availability guarantees which are aligned with the benefit that is derived from the information flow, and by proactively responding to perceived changes in system stability. We also present additional techniques, based on real-world case studies, which can help a system recover from non-transient failures. Differentiate information flow service guarantee is realized through the IQ-Paths overlay network management and flow mapping and scheduling techniques, which is addressed in more details in Chapter 4

3.2 System Overview

This section describes a model of the information flows under consideration, and it elaborates the fault model used for the proactive availability management techniques explained later.

3.2.1 Information Flow Model

An information flow is represented as a directed acyclic graph $G(V_g, E_g, U_{net})$ with each vertex in V_g representing an information-source, an information-sink or a flow-operator that processes the information i.e. $V_g = V_{sources} \cup V_{sinks} \cup V_{operators}$. Edges E_g in the graph represent the flow of information, and may span multiple intermediate edges and nodes in the underlying network. The utility-function U_{net} is defined as:

$$U_{net} = Benefit - Cost \tag{1}$$

Both benefit and cost are expressed in terms of some unit of value delivered per unit time (e.g., dollars/second). Benefit is a user-supplied function that maps the delay, availability, etc. of the information flow to its corresponding value to the enterprise. Cost is also a user-supplied function; it maps resources such as CPU usage and bandwidth consumed to the expense incurred by the enterprise. We will expand the terms of this seemingly simple equation in upcoming sections.

3.2.2 Fault Model

We are concerned with failures that occur after the information flow has been deployed. In particular, we consider fail-stop failures of operators that process events. Such failures could result from problems in the operator code or in the underlying physical node. Other factors might also cause failures, but are not considered here, including problems with sources, problems with the sink, or link failures between nodes. While such issues can cause user-perceived failures, they must be addressed with other techniques. For example, link failures could be managed by retransmission or re-routing at the network level.

For the purpose of failure recovery, we assume that each flow-operator consists of a *static-state* S_{static} that contains the information about the edges connected to the operator and the enterprise logic embedded in the operator; in contrast, the *dynamic-state* $S_{dynamic}$ is the information that is a result of all the updates that have been processed by this operator (shown in Figure 11). Recovery therefore, is dependent upon the correct retrieval of the states S_{static} and $S_{dynamic}$, which jointly contain the information necessary for re-instantiation of flow-operator and information flow edges. However, as described next, simply recovering these states may not prevent the recurrence of a failure.

3.2.2.1 Transient Faults.

A fault can be caused by a condition that is transient in nature (e.g., a memory overload due to a mis-behaving process). Such faults will not typically recur after system recovery. In our formulation, a transient fault would cause the failure of an operator, and correct

retrieval of the two states associated with the operator would ensure permanent recovery from this fault. The techniques proposed in this work are capable of effectively handling faults of this nature.

3.2.2.2 Non-Transient Faults.

Non-transient faults may be caused by some bugs in the code or some unhandled conditions. For information flows, this may mean recurrence of the fault even after recovery, particularly when recovery entails repeating the same sequence of messages that caused the fault. To deal with faults of this nature, we note that the output produced by a flow-operator in response to an input event E depends on the existing dynamic-state $S_{dynamic}$, the operator logic encoded as S_{static}, and the event E itself. Therefore, the failure of an operator on arrival of an event E is a result of the 3-tuple $< S_{dynamic}, S_{static}, E >$. Thus, any technique that aims to deal with non-transient failures must have application-level methods for retrieving and appropriately modifying this 3-tuple. Our prior work presents examples of such methods [35], and we generalize such techniques here.

3.3 Utility-Driven Proactive Availability Management

Traditional techniques for availability management typically rely on undo-redo logs, active-replicas, or active-passive pairs. A new set of problems is presented by information flows that form the backbone of an enterprise. For instance, using traditional on-disk undo-redo logs for information flows would lead to unacceptable recovery times for the enterprise domain in face of machine or disk failures. The other end of the availability management spectrum, which uses active replicas, would impose large additional communication and processing overheads due to the high arrival rate of updates, typically making it economically infeasible for the enterprise to use this option. In response, we take the active-passive pair algorithm [68], and customize it for enterprise-scale information flows. To do this, we will incorporate our previous work on soft-checkpoints [86], and add the ability to dynamically choose checkpointing intervals to reduce communication and processing overheads.

For completeness, we first describe the existing active-passive pair and soft-checkpoint techniques, and then describe our enhancements.

3.3.1 Basic Active-Passive Pair Algorithm

To ensure high-availability for the flow-operator, in its simplest form, the active-passive pair replication requires:

- A *passive node* containing the static-state S_{static} of the flow-operator hosted on the active node.
- An *event log* at the flow-graph vertices directly upstream to the flow-operator in question.
- A mechanism to *detect duplicates* at the flow-graph vertices directly downstream to the flow-operator.
- A *failure detection* mechanism for the active node hosting the primary flow-operator.

In case of a failure, recovery proceeds as follows: the failure detection mechanism detects the failure and reports it to the passive node. On receipt of the failure message, the passive node instantiates the flow-operator, making use of the static-state, S_{static}, already available at the node. The instantiated operator then contacts the upstream vertices for retransmission of the events in their event log. The newly instantiated operator node processes these re-transmitted events in a normal fashion, generating output events, and leaving it to the downstream nodes to detect the resulting duplicates. Once the retransmission of the event log has been completed, the resulting dynamic-state, $S_{dynamic}$, will be recovered to the state of the failed operator, and normal operations can resume. Unfortunately, this simple algorithm can lead to long recovery times, large event logs at the upstream nodes, and large associated retransmission costs. The remedy to these problems is the 'soft-checkpoint' technique, described next.

The event logs at the upstream nodes and their retransmission to the recovered operator are required for reconstructing the dynamic-state $S_{dynamic}$, of the failed operator. However, in practice, it is advantageous to retain additional stable state at the passive node in order to avoid the need to re-transmit the entire event log. Such state saving is called soft-checkpointing, because it is not needed for correctness. Soft checkpoints can be updated on an intermittent basis in the background. Once taken, the component receiving the checkpoint no longer requires the events on which the state depends for reconstructing $S_{dynamic}$. This in turn permits upstream nodes to discard the event logs for which the soft-checkpoint has been taken. Soft-checkpointing, therefore, is an optimization that reduces worst-case recovery time and permits the reclamation of logs.

The introduction of soft-checkpoints requires small modifications to the recovery mechanism described earlier in this section. The flow-operator at the active node in the duration prior to failure would intermittently send messages to the passive node that contain information about the incremental change to its dynamic-state since the last message. The passive node, after the receipt of complete state update message from the active node, applies the incremental modifications to the state it holds and then sends a message to the flow-operator's upstream neighbors about the most recent event contained in the message from the active node. The upstream nodes can use such information to purge their event logs. In case of a failure, the algorithm proceeds exactly as described earlier, but only a small fraction of the events needs to be re-transmitted and processed.

3.3.2 Availability-Utility Formulation

In this section, we use a basic availability formulation to better describe the effects and trade-offs in soft-checkpoint-based active-passive replication. Availability $\mathcal{A}_{\mathcal{I}}$ is described in terms of Mean-Time Between Failure, $MTBF$ and Mean-Time To Repair, $MTTR$.

$$\mathcal{A}_{\mathcal{I}} = \frac{MTBF}{MTBF + MTTR} \tag{2}$$

As stated earlier, our approach contributes to a reduction in recovery time and also reduces the processing and communication overhead imposed as a result of ensuring a certain level of availability. The reduction in recovery time results in lower MTTR and a reduction in associated overheads diminishes cost. Jointly, both result in higher net-utility U_{net}, which is the actual utility provided by the system.

With our approach, MTTR depends on two factors: (1) the time to detect a failure, and (2) the time to reconstruct the dynamic-state of the operator. Failure detection mechanisms generally rely on time-outs to detect failures and therefore, depend on the coarseness of the timer used for this purpose. Some research in the domain of fault-tolerance has focused on multi-resolution timeouts [85], but to simplify analysis, henceforth, we assume that the time to detect a failure is a constant. The second factor contributing to MTTR depends on the soft-checkpoint algorithm. Specifically, a higher frequency f_{cp}, expressed in per unit time, of such checkpoints would lead to a smaller number of events required to reconstruct $S_{dynamic}$ in case of a failure. Therefore:

$$MTTR \propto \frac{1}{f_{cp}} \tag{3}$$

For simplicity, we next derive the availability-utility formulation for a single information flow (self-configuration across multiple information flows is addressed in Section 3.3.3), and we assume that the $Benefit$ and $Cost$ depend only on availability. In this case, in general, the benefit derived from a system is directly proportional to its availability. Thus:

$$Benefit \propto \frac{MTBF}{MTBF + k_1/f_{cp}} \tag{4}$$

The above formulation may lead one to believe that a higher f_{cp} is good for the system. Unfortunately, a higher f_{cp} also means more cost to propagate checkpoints from the active node to the passive node. Therefore:

$$Cost \propto f_{cp} \tag{5}$$

Note that a higher f_{cp} also results in fewer events retransmitted per soft-checkpoint; however, for large values of MTBF this effect is minor compared to the effects described above

(increase in benefit due to better availability, and compared to the increase in cost due to a higher frequency of checkpoints). Experiments reported in Section 3.5.2.4 study the effects of soft-checkpoint frequency on the cost and availability of information flows.

Combining equations 1, 4, 5, and replacing proportionality using constants, we arrive at:

$$U_{net} = \frac{k_2 \times MTBF}{MTBF + k_1/f_{cp}} - k_3 \times f_{cp}, \tag{6}$$

which represents the business-utility calculation model and the constants are determined by business level objectives [49, 99], or using more detailed formulation described later. This equation expresses the key insight that net-utility depends not only on MTBF, but also on the soft-checkpoint frequency used in a system, the latter both positively contributing to net-utility (by reducing the denominator) and directly reducing net-utility (by increasing the term being subtracted). Intuitively, this means that frequent checkpointing can improve utility by reducing MTBF, but that it can also reduce utility by using resources that would otherwise directly benefit the information flow.

3.3.3 Availability-Aware Self-Configuration

Ideally, we would like to maximize the availability of an information flow, but given that there is an associated cost, our actual goal is to choose a value of availability that maximizes its net-utility. In our algorithm and its mathematical formulation, f_{cp} is the factor that governs availability. By setting the derivative of equation 6 equal to zero, we find that the value of f_{cp} that maximizes net-utility is:

$$f_{cp} = \sqrt{\frac{k_1 \times k_2}{k_3 \times MTBF}} - \frac{k_1}{MTBF} \tag{7}$$

In the presence of multiple information flows, each with a different benefit-function, the resource assignment for availability is driven by the need to maximize net-utility across all deployed information flows. Total net-utility of the entire system, then, is the sum of individual net-utilities of information flows. For a system with n information flows,

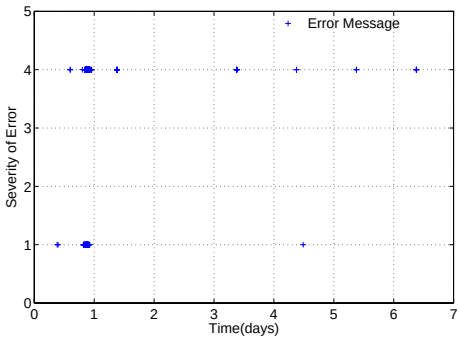


Figure 12: Enterprise Error-Log Showing Predictable Behavior of Failures

we will need to calculate $\{f_{cp}^1, f_{cp}^2, ..., f_{cp}^n\}$, which will automatically determine resource assignments. The value of f_{cp} for each information flow can be calculated using partial differentials, and the involved calculations are omitted due to space constraints.

3.3.4 Proactive Availability Management

We have established that net-utility depends on checkpoint frequency and MTBF. However, the MTBF in a real system is not a constant. Instead, the rate of failures fluctuates, with more failures occurring when the system is in an unstable state. For example, during periods of extreme overload, the system is likely to experience many component failures. If we can better approximate the current MTBF, and in particular predict when there will be many failures, we can make better decisions about checkpointing, increasing the checkpoint rate when the current MTBF is low (and failures are imminent.)

3.3.4.1 Failure Prediction.

An effective way to estimate the current MTBF is to use failure prediction techniques to generate 'early alarms' when a failure seems to be imminent. By using failure prediction, our approach can be 'better prepared' for an imminent failure, by taking more frequent soft-checkpoints. Analysis logs provided to us by one of our industry partners strengthens our belief in the usefulness of dynamic failure prediction. These logs contain error messages and warnings that were recorded at a middleware broker over a period of 7 days, along

with their time-stamps. Figure 12 shows the distribution and severity of errors recorded at the broker node. One interesting observation of these logs is that errors recur at almost the same time (around 9:00am as read from the log time-stamp) beginning from the 2nd day. Another interesting observation about the same set of logs is that 128 errors of severity level 1 occurred from 7:30pm in the first day before a series of level 4 errors occurred from 8pm. Based on such logs, it would be reasonable, therefore, to assume lower MTBF (i.e., predict imminent failures) for the 9am time period and the period when a large number of less severe errors occur, than for other time periods in which this application executes. We note that similar time- or load-dependent behaviors have been observed for other distributed applications [25].

We implemented the Sequential Probability Ratio Test(SPRT) used in MSET [38, 106] failure prediction method, to predict failures injected by the FIMD [10] failure injection tool, including timing delay, omission, message corruption datatype, message corruption length, message corruption destination, message corruption tag, message corruption data, memory leak, and invalid memory access. The SPRT method is a run-time statistical hypothesis test that can detect statistical changes in noisy process signals at the earliest possible time, e.g., before the process crashes or when severe service degradation occurs. SPRT has been applied successfully to monitor nuclear power plants, and it has recently been used for software aging problems, e.g., for the database latch contention problem, memory leaks, unreleased file locks, data corruption, etc. For example, an early warning may be raised about 30 seconds (the 'early warning capability') before a memory leak fault causes the service to degrade dramatically or the process crashes. For database shared-memory-pool latch contention failures, early warning capabilities of 5 minutes to 2 hours have been observed. For additional information about SPRT and associated MSET method, please refer to [38] and to an extended version of this text in a technical report [16].

3.3.4.2 *Modulating Checkpoint Frequency.*

The idea behind proactive availability management is to use failure prediction to modulate f_{cp}. We first provide the important yet simple guideline regarding checkpoint frequency modulation, we then develop a detailed formulation for enterprise-scale information flows, and finally, present a formulation and method to meet some specific availability requirement while also maximizing net-utility.

General guidelines. Intuitively, if a failure prediction turns out to be correct, the system 'benefits' because of reduced $MTTR$; if a prediction turns out to be a false-positive, the system still operates correctly, but it pays the extra 'cost' due to increased f_{cp}. Stated more formally, let:

$$\begin{aligned} \alpha &= \textit{prediction false–positive rate} \\ \beta &= \textit{prediction false–negative rate} \\ f'_{cp} &= \textit{modulated checkpoint frequency after a failure is predicted} \\ T_{proactive} &= \textit{duration of increased checkpoint frequency} \\ k &= \textit{timeout after which an operator is concluded to have failed} \end{aligned}$$

Earlier, $Cost$ was shown to be proportional to soft-checkpoint frequency. The new cost, $Cost'$, due to modulated f'_{cp}, is:

$$Cost' = Cost \times f'_{cp}/f_{cp} \tag{8}$$

This increased cost is incurred for a duration equal to $T_{proactive}$, and it is incurred each time a prediction is made. Therefore, the additional cost incurred per prediction is:

$$\delta Cost = (f'_{cp}/f_{cp} - 1) \times Cost \times T_{proactive} \tag{9}$$

The increase in f_{cp} also affects the availability of the system and therefore, the benefit, $Benefit'$, derived from the system. Using equation 4, we have:

$$Benefit' = \frac{MTBF + k_1/f'_{cp}}{MTBF + k_1/f_{cp}} \times Benefit \tag{10}$$

Therefore, the increase in benefit due to a correct prediction that affects a period equal to $MTBF$ is:

$$\delta Benefit = (Benefit' - Benefit) \times MTBF \tag{11}$$

Since λ is the fraction of false-positives and because there is no increase in benefit due to a false positive, the following condition expresses when proactive availability management based on failure prediction is beneficial for an entire system:

$$\delta Cost < (1 - \alpha) \times \delta Benefit \tag{12}$$

Proactive availability management. Different systems could have different types and formulations of benefit and cost, and the above analysis provides the general guideline regarding proactive availability management. For the enterprise information flow system targeted by this thesis, the proactive availability management problem can be formulated in more details as follows. Proactive availability management regulates checkpoint frequency based on stability predictions to maximize net business utility. By considering 'total cost', including the cost of checkpointing and the utility loss because of a failure (i.e. the extra utility the system could offer if there had been no failure), the problem of maximizing net-utility can be converted to the problem of minimizing total cost. This total cost consists of the cost of normal checkpointing (at frequency f_{cp}), $Cost^{cp}$, the cost due to false-positive failure prediction (i.e., the failure predictor raises a false alarm), $Cost^{fp}$, the cost due to false-negative failure prediction (i.e., a failure is not predicted successfully), $Cost^{fn}$, and finally, the cost associated with failure recovery when a failure is successfully predicted, $Cost^{ps}$.

These four types of cost are summarized in Table 5. For the cost of normal checkpoints, $Cost^{cp}$, C_1 is the cost for each checkpoint update (e.g., the communication cost), and P is the possibility an operator could fail from any time t to t+1 (seconds). Here, $P(1-\beta+\alpha)$ is the fraction of time when the checkpoint frequency is $f_{cp}^{'}$, due to correct failure predictions and false alarms. For the cost of false-positive failure prediction, t_o is the average time

Table 5: Four Types of Cost

$Cost^{cp}$	=	$[1 - P(1 - \beta + \alpha)]f_{cp}C_1$,
$Cost^{fp}$	=	$\alpha P f'_{cp} t_o C_1$,
$Cost^{fn}$	=	$\beta P C_2/(2f_{cp}) + \beta P(k+1/(2f_{cp}))C_3$,
$Cost^{ps}$	=	$(1-\beta)P[C_2/(2f'_{cp}) + (k+1/(2f'_{cp}))C_3 + f'_{cp}t_0 C_1]$.

a predictor raises an early alarm for a severe failure. In the equation for the cost due to false-negative prediction, $Cost^{fn}$, the first term is the total state recovery cost, i.e., the cost for the passive node to recover from the latest checkpointed state to the state when the failure occurred, including retransmission cost and re-computation cost. C_2 is the average recovery cost per unit time($\$/sec$). The second term is the total utility loss from the time when failure occurs to the time when the system recovers to normal operational status. In other words, this term represents the utility the system could provide if there had been no such failure. C_3 is the utility the system provides per second($\$/sec$) if there is no failure. The cost associated with failure recovery when a failure is successfully predicted, $Cost^{ps}$, is determined in a similar manner as $Cost^{fn}$.

To regulate checkpoint frequency, proactive fault tolerance finds the best checkpoint frequency, f_{cp}, when there is no failure predicted, and the best checkpoint frequency, f'_{cp}, after the time a failure is predicted. This is done by minimizing the total cost.

Meet specific availability requirement. Often, enterprises have specific requirements for system availability. For example, a 365 x 24 system with maximum allowed average down-time of 8.76 hours (i.e., 525 minutes) per year requires 99.9 percent availability, while a system with only 3 minutes of service outage must have at least a 99.999 percent availability. To achieve such availability is difficult due to the high cost of fault tolerance services and equipments. Proactive availability management is able to strike a balance between these two factors by jointly considering availability and utility when regulating checkpoint frequency. Notice that MTTR can be expressed as:

$$MTTR = (1/2f_{cp} + k)\,\beta + \left(1/2f'_{cp} + k\right)(1 - \beta), \tag{13}$$

where k is the timeout after which we conclude that a module actually failed, the availability is given by:

$$\begin{aligned} A_I &= \frac{MTBF}{MTBF + MTTR} = \frac{1 - P \cdot MTTR}{1} \\ &= 1 - p[(1/2f_{cp} + k)\,\beta + \left(1/2f'_{cp} + k\right)(1 - \beta)] \end{aligned} \tag{14}$$

Proactive fault tolerance meets the minimum availability requirement and also maximizes net utility by solving the following equation:

$$Minimize\{Cost = Cost^{cp} + Cost^{fp} + Cost^{fn} + Cost^{ps}\}, \text{ subject to:}$$

$$1 - p[(1/2f_{cp} + k)\,\beta + \left(1/2f'_{cp} + k\right)(1 - \beta)] \geq A_I^{required} \tag{15}$$

This optimization problem is of small size with two variables and one constraint, and is solved using standard Quasi-Newton method with inverse barriers.

3.3.5 Handling Non-Transient Faults

Non-transient failures are a result of bugs or unhandled conditions in operator code. Traditional techniques for ensuring high-availability that use undo/redo logs [37, 86] are useful for transient failures, but for non-transient failures, they may result in recurrence of faults during recovery. The same applies to replication-based approaches [5], for which all replicas would fail simultaneously for non-transient faults.

As described in Section 3.2.2.2, a non-transient failure of the information flow in our model is a result of the 3-tuple $< S_{static}, S_{dynamic}, E >$. The active-passive pair approach for ensuring high-availability has sufficient information during recovery to change this 3-tuple. The passive-node during recovery has access to S_{static}, a stale state $S'_{dynamic}$, and

a set of updates T from the upstream nodes that when applied to $S'_{dynamic}$, would lead to $S_{dynamic}$. The rationale behind our approach to avoid non-transient failures is simple: avoid the 3-tuple that caused the failure. This can be done in a number of ways, and the retransmitted updates T along with application-level knowledge holds the key:

- *Dropping Updates*: the simplest solution to avoid recurrence of a fault is to avoid processing the update that caused the failure. Our earlier work on 'poison messages' used this technique [35].
- *Update Reordering*: changing the order in which updates are applied to $S'_{dynamic}$ during recovery can avoid $S_{dynamic}$. This makes use of application-level knowledge to ensure correctness.
- *Update Fusion*: combining updates to avoid an intermediate state could be an option. A simple example of this approach could be the use of this technique to avoid 'division by zero' error.
- *Update Decomposition*: decomposing an update into a number of equivalent updates can be an option with several applications, and this can potentially avoid the fault.

While seemingly simple, the techniques described above are often successful in realistic settings. For example, one of our collaborators, reported an occasional surge in the usage of resources connected to their Operational Information System (OIS) [55] that traced back to a particular uncommon message type. The resulting performance hit caused other subsystem's requests to build up, including those from the front ends used by clients, ultimately threatening operational failure (e.g., inappropriately long response times) or revenue loss (e.g., clients going to alternate sites). Such uncommon request/message, termed 'Poison Messages', were later found to be identifiable by certain characteristics. The solution then adopted was to either drop or re-route the poison message in order to maintain operational integrity.

3.4 Middleware Implementation

IFLOW is an information flow middleware developed at Georgia Tech. IFLOW implements the information flow abstraction of Section 2.1 and provides methods to deploy and then optimize (by migrating operators) the information flow. For more details please refer to [48, 49, 81].

We now briefly describe the features that enable proactive availability management in the IFLOW middleware. These features are implemented both at the *control plane* and the *data plane* of this middleware infrastructure.

3.4.1 Control Plane

The control plane in IFLOW is the basis for managing information flows. Self-management methods involve running a self-configuration and a self-optimization algorithm, carried out by exchanging control messages between physical nodes that are external to the data fast paths used to transport IFLOW data. Control actions involve operations like flow-control, operator re-instantiation, etc. The main new features of the IFLOW control plane that are used for proactive fault tolerance are described below:

- *Availability-aware self-configuration module*: the benefit-formulation in IFLOW allows for availability goals to be specified, and determines the best value of f_{cp} by using the formulation described in Section 3.3.2.
- *Failure detection & prediction*: IFLOW attempts to use the regular traffic from a node to determine its liveness, but it switches to specific detection messages if there is no regular traffic from the node to the monitoring node. We also have a provision for multi-resolution timeouts to reduce the load imposed by the failure detection algorithm. Finally, state can be maintained to use failure history for predicting failures, but we have not yet implemented any specific technique into IFLOW.
- *Control messages*: SOAP calls are used to notify active-node failure, to communicate log purge points to upstream vertices, etc.

- *Update re-direction in case of failure*: a simple control mechanism exists at the upstream vertices to re-direct updates to the passive node in case of failure. The connection between upstream vertices and the passive node is created at the time of flow deployment.

3.4.2 Data Plane

A *fast data-path* is one of the key design philosophies of the IFLOW middleware. We have taken care that the features required for proactive availability management have minimal impact on the data-path. In order to ensure proactive availability management, the state of an operator on the data plane needs to be soft-checkpointed and the changes need to be periodically communicated to the passive-node. The fact that a soft-checkpoint is not necessary for correctness of proactive availability management ensures minimal impact on the data-path. Specifically, the active-node can transfer the soft-checkpoint to the passive node asynchronously (e.g., when load is low), and this will not compromise the correctness of our algorithm. The specific features required for proactive availability management are described below:

- *Logging at upstream vertices*: any update that is sent out from the source vertex is logged to enable retransmission in case of failure. Additional logs can be established at intermediate nodes (an operator vertex is a source for downstream vertices) to enable faster recovery. The log module also implements a mechanism to purge the log when a message is received from the downstream node after a soft-checkpoint is completed.
- *Soft-checkpoint module at operator vertices*: the soft-checkpoint module tracks the changes in $S_{dynamic}$ since the last soft-checkpoint. It is also responsible for sending soft-checkpoints to the passive node.
- *Duplicate detection at the downstream node*: the duplicate detection mechanism is based on the monotonic update system proposed in our earlier work [86]. When the updates cannot be ordered using the contained attributes, a monotonically increasing

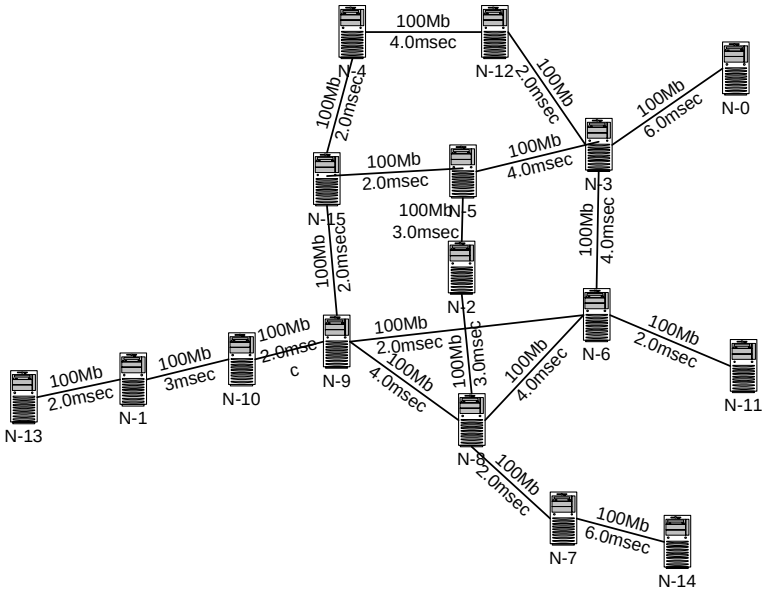


Figure 13: Sample Testbed. The Testbed Topology is Generated using GT-ITM and is Configured at Emulab Facility.

attribute (e.g., the real-time clock) is appended to the out-going update that uniquely identifies this update.

- *Additional edge between active-passive pair*: a supplementary data-flow between the active-passive pair delivers the soft-checkpoints to the passive vertex.
- *Maintaining checkpoints at passive-node*: the passive vertex contains the logic that applies an incoming soft-checkpoint to the recorded active node state.

3.5 Experiments

Experiments are designed to evaluate the performance our proactive availability management techniques. First, simulations are used to better understand the behavior of the self-configuration module that determines the availability requirement based on the user-supplied benefit function. Next, an end-to-end setup is created on Emulab [93], representing an enterprise-scale information flow to compare our approach against the traditional approaches and to study the effect of different soft-checkpoint intervals and proactivity on

aspects like MTTR, recovery cost, and net-utility. Results show that proactive availability management is effective at providing low-cost failure resilience for information flow applications, while also maximizing the application's net-utility.

3.5.1 Simulation Study

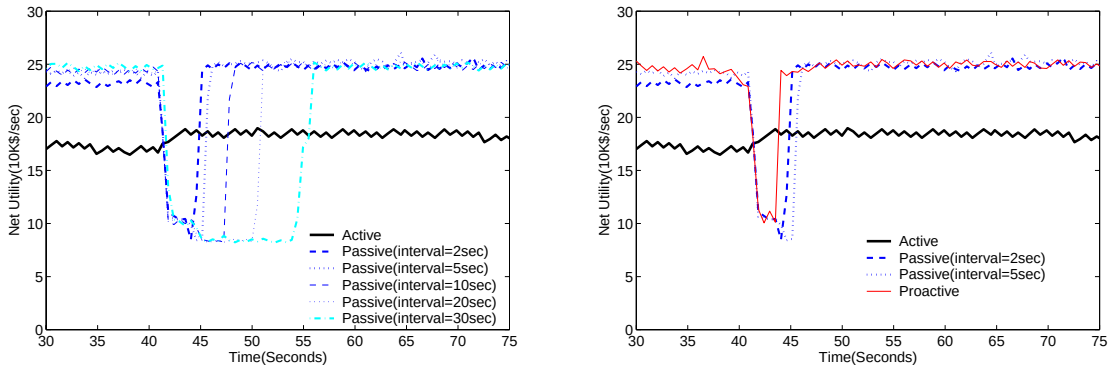
A simulation study is used to compare utility-based availability management to simple approaches that are not availability-aware. The study uses a 128 node topology generated with the GT-ITM internetwork topology generator [107]. The formulation of net-utility U_{net} determines benefit as: $benefit = k_1 \times (k_2 - delay)^2 \times availability \times availableBandwidth / requiredBandwidth$, and cost is calculated as: $cost = dataRate \times bandwidthCostPerByte$. Random costs are assigned to the network links, expressed in dollars per byte. We substitute ($k_1 = 1.0$, $k_2 = 150.0$) in the benefit formulation for this specific simulation [49]. The MTBF is assumed to be 86400sec. and the MTTR is assumed to be 864sec. for a f_{cp} value of 0.01Hz. (Many values are possible for these variables. However, we must choose some values when conducting our simulations, and the ones we chose are reasonable for the enterprise environment.) We first deploy the flow-graph using the net-utility specification from equation 1 as the optimization criteria, and the results are shown in Table 6 under the column labeled 'Utility'. The results show a high achieved net-utility with acceptable values for delay, f_{cp} and availability. The second deployment (under 'Cost') focuses instead on minimizing the cost, and it uses $1/cost$ as the optimization criteria. The effect of choosing different criteria is evident in the reduced cost, achieved by allowing a higher delay and a lower availability (resulting from lower f_{cp}). The final experiment uses $1/delay$ to drive the deployment. This results in a reduction of delay achieved for the flow-graph, but at the expense of net-utility and availability.

3.5.2 Testbed Experiments using IFLOW

This set of experiments is conducted on Emulab [93], and the network topology is again generated using the GT-ITM internetwork topology generator. In many cases, enterprises

Table 6: Self-Determining Availability Based on Benefit

Optimization Criterion	**Utility**	**Cost**	**Delay**
`Net-Utility (dollars/sec)`	431991	52670	2160
`Cost (dollars/sec)`	79875	14771	80315
`Delay (msec)`	222	444	191
f_{cp} `(sec`$^{-1}$`)`	0.050	0.018	0.020
`Availability (percent)`	99.88	99.66	99.70



(a) ***Active and passive approach(various intervals)***

(b) ***Proactive, active, and passive approach***$(interval = 2s, 5s)$

Figure 14: Net Utility Rate Variations using Active, Passive or Proactive Fault Tolerance Approaches. A failure is injected into one operator node at the time $t = 40s$.

would hand tune their topology for availability and performance, instead of using an arbitrary topology. For example, an enterprise may explicitly designate a primary and secondary data center. An arbitrary topology is used in our experiments in order to understand how our techniques perform without the benefit of additional hand tuning. Figure 13 shows the testbed used for experimental evaluations. Background traffic is generated using cmu-scen-gen [96], injected into the testbed using rate-controlled udp connections. For the testbed depicted in Figure 13, background traffic is composed of 900 CBR connections. We use the utility formulation in Equation 15 to better study the net-utility and the costs associated with checkpointing and failures. Required availability is 99.9% if not stated otherwise.

3.5.2.1 *Variation of Net-Utility for Different Approaches.*

The first experiment studies the variation of net-utility with different availability management approaches in the presence of failures. For simplicity, only one failure is injected into the system. We conduct experiments with the active replication approach, the passive replication approach with varying soft-checkpoint intervals, and our proactive replication approach. Figure 14 clearly demonstrates that the active replication approach provides lowest net-utility. This is because of the high amount of replicated communication traffic when using this approach. After a failure, net-utility of the active approach increases slightly; there is less replication traffic, because the failed node no longer sends replicated output updates. The experiment also corroborates the analysis in Section 3.3.2: a lower soft-checkpoint interval for the passive approach imposes higher communication cost on the system and therefore, results in lower net-utility. Note that if availability were a predominant factor in the net-utility formulation, then a lower soft-checkpoint interval could have resulted in higher net-utility. The cost of soft-checkpoints is almost negligible when the interval is greater than 5 seconds, but its effect is evident for an interval of 2 seconds.

Our proactive approach provides the highest net-utility overall, as it modulates the soft-checkpoint interval and takes into account the perceived system to offer preventive fault tolerance. For instance, it switches to a smaller soft-checkpoint interval just before the failure and is therefore able to recover as fast as the passive approach with a 2 seconds update interval, while performing as well as the passive approach with a 30 seconds update interval at other times. We note that evaluation of failure prediction techniques is not the focus of this thesis (such kind of evaluations appear in [16]). To investigate how prediction accuracy affects the system, these experiments simulate a predictor for the proactive approach, with failure prediction statistically generated at various levels of accuracy. In particular, we notify the soft-checkpoint mechanism that a failure is imminent, no matter whether the prediction is correct or a false positive.

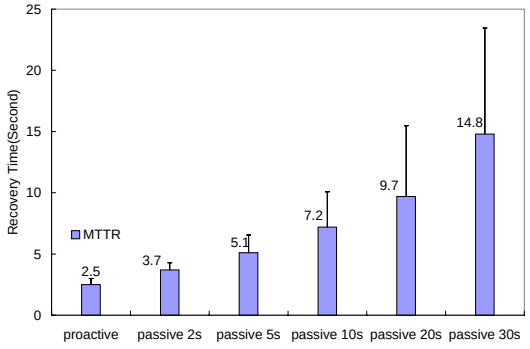


Figure 15: MTTR and Standard Deviation of Recovery Time under Three Replication Strategies. Standard deviation is represented by vertical error bars.

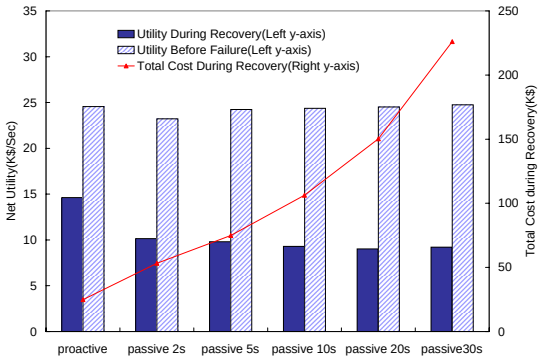


Figure 16: Utility Before Failure and During Recovery, and the Total Cost to Recover from one Failure.

3.5.2.2 Variation of MTTR for Different Approaches.

The variation of MTTR and its standard deviation with different approaches are shown in Figure 15. For each approach, nine experiments are used to obtain the mean and standard deviation. The active replication approach (not shown in the graph) has no explicit recovery time. This is because the node downstream of the replicated operator continues to receive processed updates even after the failure of one active replica. On the other hand, the passive replication approach which attempts to avoid the high cost of active replication incurs recovery times that increase with the soft-checkpoint interval. The reason for this increase is the time taken for reconstructing the operator state: the higher the soft-checkpoint interval, the larger the number of updates required to rebuild the state. Recovery time for the passive replication approach depends on the soft-checkpoint interval. It ranges from 3.7 seconds (for a 2 second interval) to 14.8 seconds (for a 30 second interval). Our proactive approach, as expected, performs well as compared to other passive replication approaches, since it is able to change over to a very small soft-checkpoint interval just before the failure, and hence, has low MTTR. The experiment demonstrates the importance of choosing the right soft-checkpoint interval automatically to maximize availability at low cost and thereby maximize the net-utility of information flows.

3.5.2.3 *Cost & Net-Utility During Recovery.*

Our proactive availability management approach increases soft-checkpoint activity when a failure is predicted in the near future, but it maintains a low soft-checkpoint activity at other times. The analyses of net-utility value before failure, during failure recovery, and the total cost to recover from failure are summarized in Figure 16. Net-utility using proactive availability management is higher than any other approach, because it contains a very recent soft-checkpoint for the operator state and therefore, incurs the least cost during recovery. Note that passive replication with an interval of 2 seconds also incurs a low cost during recovery, but this is achieved by losing non-negligible net-utility at normal operation time.

3.5.2.4 *Effects of Checkpoint Frequency and Prediction Accuracy on Cost and Availability.*

The next experiment closely examines the effect of checkpoint frequency on the system, both in terms of system availability and the cost imposed to gain a unit amount of utility. As mentioned in Section 3.3.2, a higher f_{cp} leads to a higher number of soft-checkpoint messages from the active to the passive node, but it also leads to a smaller number of updates being required to reconstruct the operator state during recovery. The conflicting behavior of incurred cost due to f_{cp} is represented in Figure 17 by the two parabolic curves. Ideally, we would like to spend the minimum cost to achieve a unit amount of utility and would therefore, like to choose a value of f_{cp} that is located at the dip of the parabolic curve. Note that the cost/utility ratio is consistently higher for the passive vs. the proactive approach. We also show the effect of f_{cp} on the availability of the system: the change is in line with the formulation described in Equation 4. However, the interesting insight from this experiment is the direct correspondence between the lowest achievable cost/utility and the flattening of the availability curve.

Our final experiment studies the effect of prediction accuracy λ, on the achieved cost/utility

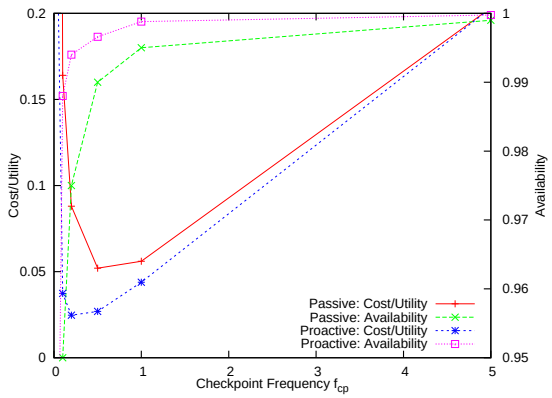


Figure 17: Effect of Checkpoint Frequency on Cost and Availability. Checkpoint frequency affects the cost (left y-axis) in a non-linear way, and it is important to optimize it. Note that there is also a sweet spot in the graph, where cost is minimized and availability (right y-axis) is also high. Our proactive approach can achieve the same level of availability with significantly less cost compared to the passive approach.

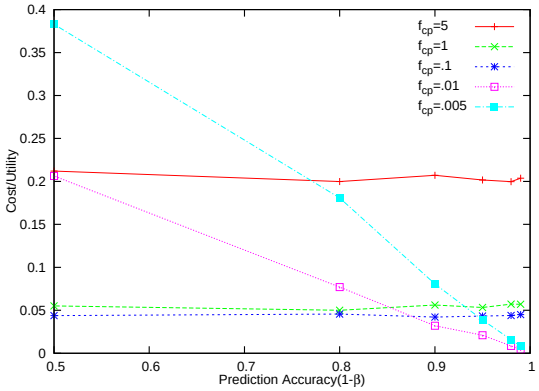


Figure 18: Effect of Prediction Accuracy on Cost of Ensuring Availability. Better prediction accuracy helps reduce the cost incurred for ensuring high-availability, especially when the checkpoint frequency is low (the curve with the deepest slope). When checkpoint frequency is sufficiently large (the four curves with $f_{cp} \geq 1Hz$), higher accuracy has less effect on the cost due to the fact that less time is required to recover from an unpredicted failure.

ratio. It is intuitive that better prediction accuracy would lead to lower cost/utility for proactive availability management, and this is clearly depicted in Figure 18. It is interesting to note the behavior of proactive availability management with a lower f_{cp} value. When prediction accuracy is low, a small f_{cp} leads to very high recovery times with low net-utility during that period. However, if f_{cp} is modulated properly to handle failures, recovery time decreases and a far lower cost/utility is achieved. Meanwhile, the effect of prediction accuracy is less prominent when a higher value of f_{cp} is used, as the recovery times don't improve much, even with a correct prediction.

CHAPTER IV

PREDICTABLY HIGH PERFORMANCE DATA STREAMS ACROSS DYNAMIC NETWORK OVERLAYS

4.1 Introduction

Data-driven high performance applications are important to many constituencies, including corporations in applications like real-time data mining or data integration [11], common end users in telepresence [54], and scientists or engineers in applications like remote data visualization [88] or instrument access [60]. A common characteristic of such applications is their need to meet quality of service (QoS) guarantees and/or offer utility-based services to end users (i.e., meet certain service-level objectives (SLOs)). However, excepting datacenter-based solutions [11] and the few dedicated, high end links with guaranteed network resources existing between select centers of excellence (e.g., via DOE's UltraScience Net [28] or the National Lambda Rail [62]), such guarantees must be provided across shared best-effort network infrastructures, where dynamic network behavior and best-effort nature make it imperative for middleware to assist end user applications in providing better *network resource availability* and best utilizing the available network resource. More specifically, when transporting and manipulating their data, applications should receive specific network resource guarantees from the overlay networks used by middleware, accommodating dynamic variations in network behavior: (1) high performance applications require consistent levels of end-to-end performance, such as limited delays for transporting online collaboration data [102] or small jitter for multimedia [29], (2) enterprise applications couple data transport and manipulation with application-level expressions of utility or cost [50], and (3) many application classes can utilize guarantees that differentiate across different traffic types, such as offering stronger guarantees for control vs. data traffic in

remote instrument access for high performance codes [70].

Previous work on middleware for data-intensive distributed applications has addressed limitations and runtime variations in network bandwidth with adaptive approaches to matching desired to available network resources. Examples include dynamically adjusting data transfer rates [15], varying compression levels in response to monitored changes in network bandwidth [101], or changing the nature of the data being sent [15, 46, 102]. Other research has sought to use alternative network connections or new network infrastructures to compensate for problematic connection behaviors [70].

Here we present the IQ-Paths [17, 19] approach to self-regulating high performance data streaming with defined quality requirements across wide area networks. IQ-Paths offers novel functionality that enhances and complements existing adaptive data streaming techniques. First, IQ-Paths dynamically measures [56] and then, also predicts the available bandwidth profiles on network links. Second, it extends such online monitoring and prediction to the multi-link paths in the overlay networks used by modern applications and middleware. Third, it offers automated methods for moving data traffic across overlay paths. These include splitting a single data stream across multiple paths to improve performance through parallelism and to improve desired end-to-end behavior by dynamically differentiating the amounts and kinds of data traffic imposed onto different paths. Such self-regulating data movement and differentiation utilizes a dynamic packet scheduling algorithm that automatically maps packets to paths to match application-level utility specifications. Finally, an important attribute of IQ-Paths is that unlike other methods for bandwidth prediction based on measurements of average bandwidth, it uses statistical techniques to capture the dynamic or noisy nature of available network bandwidth across overlay paths. This enables it to better map data with different desired utility – service guarantees – to the underlying best effort network infrastructure.

Our research uses IQ-Paths for both scientific and high end media applications. In the scientific domain, real-time remote data visualization for a molecular dynamics (MD)

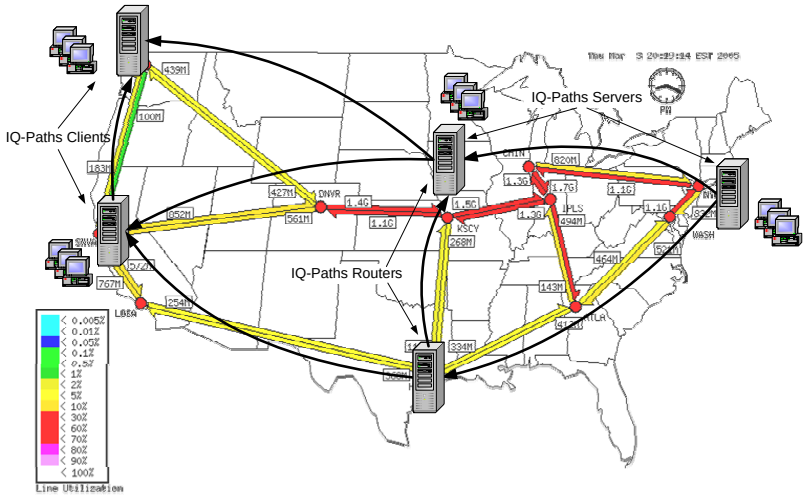


Figure 19: IQ-Paths Overlay Network: servers, routers, and clients continually assess the qualities of their logical links, admit and map data streams with different desired utility using a self-regulating packet routing and scheduling algorithm.

code benefits from IQ-Paths' ability to better meet its dynamic end user requirements. A specific example is to differentiate the transport of certain elements of the application's data streams, as with the atoms vs. bond forces visually depicted for each timestep of the MD application. Another example is to use network paths with more stable bandwidths for the critical 'control' traffic in the remote visualization software and also for the most time-sensitive data sets in large volume parallel data transfers. Stability is dynamically diagnosed and predicted via the aforementioned statistical techniques. In the multimedia domain, IQ-Paths is shown to deliver improved performance for different encoding levels of MPEG-4 video streams [18].

Results in Section 4.5 also demonstrate the advantages derived from IQ-Paths' statistical guarantees. Specifically, there are distinct improvements over earlier work on adaptive methods that provide QoS over wide-area networks by predicting future average network behavior from past history [52]. With such methods, quantities like RTT can be predicted well, but average available bandwidth or packet loss rate are not easily captured (e.g., using predictors like MA, AR, or more elaborate methods like ARMA and ARIMA) [109]). This is because noise is a large portion of the signal in the time

series of available bandwidth or packet loss rate. As a result, the values for predicted average bandwidths will have large prediction errors. For example, the results reported in [109], based on measurements at over 49 well-connected academic and research institutions, have prediction errors larger than 20% for more than 40% of the predicted values (i.e., $|predictedvalue/actualvalue| > 1.2$), and for 10% of the values, prediction error is larger than 50%. In comparison, IQ-Paths can provide an application with strong guarantees, stating that it will receive its required bandwidth 99% of the time or experience a deadline miss rate (i.e., jitter) of less than 0.1%, for example. Finally, other methods apply low frequency filters [23] to measured values, to reduce prediction error, but unfortunately, this means that they essentially eliminate the noisy nature of (i.e., dynamic variations experienced over) certain network paths. The outcome is that applications cannot adjust to or deal with such variations, by mapping less important or less delay-sensitive data to noisier connections, for example.

When large data streams are transferred over a shared network, it is often impossible to meet some required service level agreement at all times, i.e., risk is unavoidable in such environments. At the same time, when multiple paths are available, e.g., to make service more reliable, many enterprise IT infrastructures utilize multiple commercial networks, to provide higher availability in case of network downtime. Besides network bandwidth, the cost and benefit tradeoffs between multiple networks are other important factors to consider. The risk based management introduced in Chapter 2 provides a way to manage such risk, costs, and benefits in a unified framework, offering a method to scheduling and routing streams across alternative network paths to meet high level reliability and utility requirements when using IQ-Paths. In this chapter, we also illustrate how to apply risk based management to IQ-Paths.

Figure 19 illustrates an example of an IQ-Paths overlay, which utilizes automatic network resource profiling, admission control, and self-regulating data routing and scheduling to guarantee different streams' desired utility requirements. The overlay implemented by

IQ-Paths has multiple layers of abstraction. First, its *middleware underlay* – a middleware extension of the network underlay proposed in [59]) – implements the execution layer for overlay services. The underlay is comprised of processes running on the machines available to IQ-paths, connected by logical links and/or via intermediate processes acting as router nodes. Second, underlay nodes continually assess the qualities of their logical links as well as the available resources of the machines on which they reside. The service guarantees provided to applications are based on such dynamic resource measurements, on runtime admission control, resource mapping, and on a self-regulating packet routing and scheduling algorithm. This algorithm, termed PGOS (Predictive Guarantee Overlay Scheduling), provides probabilistic guarantees for the available bandwidth, packet loss rate, and RTT attainable across the best-effort network links in the underlay.

Key technical advantages of IQ-Paths and its PGOS algorithm include the following:

- *Probabilistic and 'violation bound' guarantees*: since the PGOS algorithm uses bandwidth distribution analysis and prediction to capture network dynamics, it can make service guarantees and provide prediction accuracies superior to those provided by prediction methods based on average network behavior: (1) it can ensure that applications receive the bandwidths they require with high levels of assurance (e.g., it can guarantee that an application receives its required bandwidth 99% of the time or that its deadline miss rate is less than 0.1%)); (2) in addition, PGOS can also provide deadline violation guarantees that bound the average number of packets that miss their guaranteed QoS (e.g., their deadlines).

- *Reduced jitter*: by reducing jitter in applications like remote data acquisition or display, buffering needs are reduced. This is particularly important for high volume data transfers in time- or delay-sensitive applications.

- *Differentiated streaming services*: different streams can receive different levels of guarantees. As a result, when applications use close to the total available bandwidths

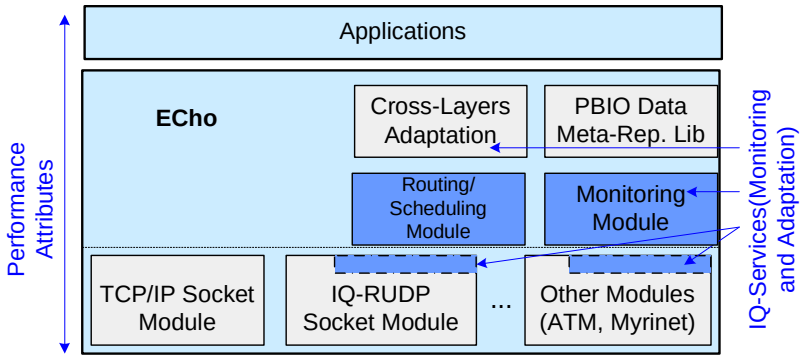


Figure 20: Middleware Architecture.

of all overlay paths, PGOS can ensure that high utility streams receive stronger service guarantees that others.

- *Full bandwidth utilization*: providing guarantees does not imply sacrificing the bandwidths available to applications (e.g., by purposely under-utilizing some link). Instead, PGOS has sufficiently low runtime overheads to satisfy the needs of even high bandwidth wide area network links.

The remainder of this chapter is organized as follows. of the IQ-Paths approach. We outline the software architecture of IQ-Paths in the next section, followed by descriptions of its bandwidth prediction methods and of the PGOS algorithm using these methods. Experimental evaluations on an emulated network testbed appear before the chapter's conclusions.

4.2 Software Architecture of the IQ-Paths Middleware

The software architecture of the IQ-Paths middleware is depicted in Figure 20. It is derived from our substantial experiences with the IQ-ECho [15] high performance publish/subscribe infrastructure implementing channel-based information subscriptions. IQ-Paths leverages IQ-ECho's support for multiple transport protocols (e.g., TCP, RUDP, SCTP) and its monitoring modules for measuring desired network metrics from middleware and in cooperation with certain transport modules (e.g., RUDP). PGOS routing/scheduling module aggregates such runtime measurements in order to schedule application packets across

multiple overlay paths. Unlike ECho, however, IQ-Paths is realized at a layer 'below' the publish/subscribe model of communication. Namely, IQ-Paths manipulates arbitrary application-level messages flowing from data sources to data sinks. Whether such messages are described as pub/sub events or in other forms is immaterial to the research described here. Similarly, IQ-Paths is not concerned with how source-to-sink links are established. It supports both direct source-to-sink links and more complex linkages that utilize overlay networks to route messages and process them 'in-flight' on their paths from sources to sinks. One way for end users to establish such linkages is via IQ-ECho's 'derived channel' abstraction. Another way is to use the deployment features implemented as part of the 'in-transit' information flow infrastructure described in [50]. A third way is to directly use IQ-Paths as the transport layer for applications, as with the IQ^{PG}-GridFTP implementation used in the evaluation section of this chapter.

The goal of IQ-Paths is to provide a general framework for routing, scheduling, and processing streams of application-level messages. Generality is established by layering IQ-Paths 'beneath' the different messaging models used by end users, including the IQ-ECho and in-transit models developed in our own research. A specific example is the IQ^{PG}-GridFTP described in this thesis, which (1) replaces its transport level with IQ-Paths and (2) interposes the IQ-Paths message routing and scheduling algorithm between GridFTP's parallel link layer and lower level message transports. As a result, IQ-GridFTP (1) retains its ability to exploit parallelism in data transport by simultaneously using multiple network links, while more importantly, (2) gaining the ability to adjust the volumes of data being transferred to the current behavior of each single network link between source and sink, and (3) using overlay paths and path bandwidth-sensitive message routing and scheduling to better control how data is streamed across multiple links from source to sink.

Important components of the IQ-Paths middleware described in this thesis are its Statistical Monitoring techniques and its Routing/Scheduling algorithms. Figure 21 illustrates the structure of each IQ-Paths overlay node and the dynamic interactions of these software

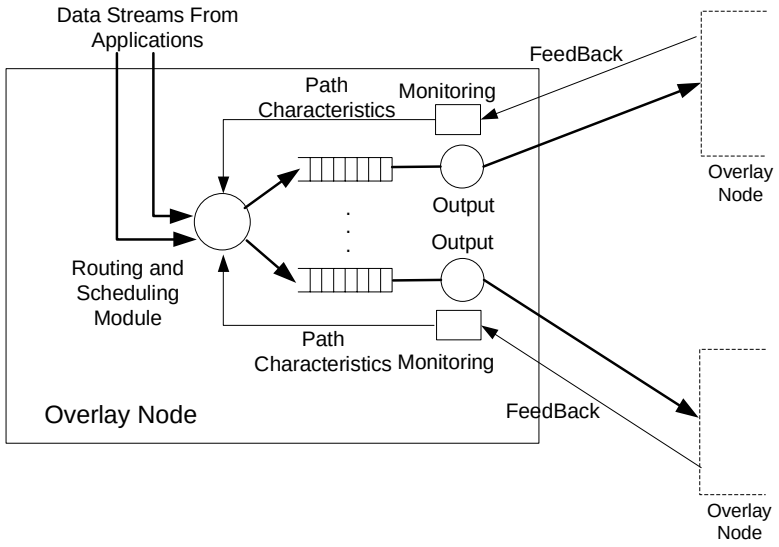


Figure 21: Structure of IQ-Paths Overlay Node.

components. Specifically, the Statistical Monitoring component monitors the bandwidth characteristics (i.e., bandwidth distribution) of each overlay path and shares this information with the Routing/Scheduling component. The latter routes applications' data streams and sub-streams to the appropriate overlay paths and in addition, for each path, it schedules the data packets mapped to it. The goal, of course, is to route and schedule application-level messages to continuously match the network loads imposed by the middleware to the available network bandwidths present in overlay paths, such that application-level metrics of stream utility are met (e.g., probabilistic guarantees on the timeliness of data delivery).

The remainder of this chapter ignores other components of the IQ-Paths middleware, referring the reader to a more complete description of the system in [15]. We next describe the manner in which bandwidth guarantees are attained.

4.3 Statistical Bandwidth Prediction and Guarantees

The PGOS algorithm presented in Section 4.4 provides to an end users predictive guarantees that with some large probability, application-level messages will experience certain levels of bandwidth across certain overlay paths. Toward this end, for each overlay path,

IQ-Paths network monitoring (1) tracks the past *distribution* of path bandwidth in the form of cumulative distribution function(CDF), and (2) uses the percentile points in that distribution as the bandwidth predictor, instead of using average bandwidth. The PGOS algorithm then uses these predictions to judiciously route and schedule streams across overlay paths by finding the path(or set of paths), which, for some large value of P_0, can ensure that the bandwidth allocated to the stream has the property of $P(bw \geq bw_0) \geq P_0$, where bw_0 is the required bandwidth.

For each specific overlay path, frequent bandwidth variation makes it difficult to predict the exact values of average available bandwidth in the near future, both for very short timescales like milliseconds and for the second timescales at which IQ-Paths operates. Statistical prediction is to leverage rather than suppress such variations, in order to provide to applications higher bandwidth guarantees. In other words, while predicting the exact value of future bandwidth is hard, statistical prediction relaxes the prediction requirement by asking if we can obtain certain amount of bandwidth with high probability. Because of the IID nature of available bandwidth, statistical prediction has much smaller prediction error than average bandwidth prediction. Furthermore, statistical prediction also retains more information including the variation and distribution of the signal, which is directly related with the service-level objectives of many applications.

Figure 22 illustrates the results of predicting average bandwidths vs. the statistical predictions used in our approach. Here, we analyze more than 8GB of IP header trace files from the National Laboratory for Applied Network Research, collected at a number of locations of the Abilene (Internet2) and the Auckland networks. The mean prediction error is the average relative error ($|(predicted\ value - actual\ value)/actual\ value|$) of several widely used average bandwidth predictors (i.e., MA, EWMA and SMA). ¿From Figure 22, the common average bandwidth predictors have a roughly 20% of prediction error. Similar error ranges are also reported in [109]. In contrast, our statistical prediction method (percentile prediction) achieves less than a 4% prediction failure rate. The percentile prediction

failure rate is the number of prediction failures divided by the total number of predictions. For these experiments, we first calculate the distribution of N (e.g., 500 and 1000) samples, where each sample is the bandwidth measured in 0.1 to 1 second. Then, since we are particularly interested in whether a path can guarantee certain throughput for 90% of the time (or for 80%, 70%, etc), we find distribution D's 10th percentile as X(Mbps), and test whether the next n (n=5 to 10) samples are larger than X. If they are, a successful prediction occurs, and if not, a prediction failure occurs.

From these experiments and for representative distributed applications, we determine two facts. First, in practice, an application is typically more interested in whether it can receive its required bandwidth consistently, than in the exact value of the bandwidth the network provides. This is precisely the question answered by our statistical bandwidth prediction methods. Second, such statistical guarantees are easier to make than guarantees about available average bandwidth, because the majority of available bandwidth or maximum throughput on Internet paths is IID [109]. As a result, the exact value of average bandwidth in the near future is hard to predict, but the statistical structure of bandwidth can be predicted well. Simply speaking, if in the last 5 mins., the 10th percentile of bandwidth is 10Mbps, then with a large probability, the bandwidth in the next 1 second will be higher than 10Mbps. The measured low prediction failure rate directly justifies our usage of percentile prediction. For high-performance computing, typically large amounts of data are transferred in some small time interval (e.g. 12.5 MB of data in 0.1 second on a 1Gbps link, or 125MB on a 10Gbps link). Mean prediction time series have a large number of outliers because of the aforementioned IID nature, if the measurement interval is small(e.g., 0.1 second). To avoid these outliers, one has to average the bandwidth over larger interval (e.g., 1 second), which loses the variation (or stability) nature of a particular network and could easily cause inappropriate data movement as illustrated in the next experiment. Statistical prediction arguably avoids this pitfall by profiling the bandwidth distribution and

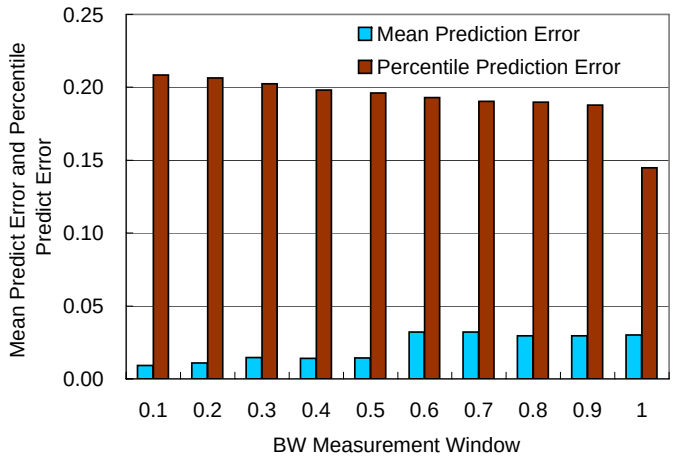


Figure 22: Bandwidth Prediction.

providing accurate predication without using lossy averaging. The outcome is a solid foundation for controlling how large amounts of data are moved around in every small time interval with high confidence levels.

4.4 The PGOS Overlay Path Guarantee and Scheduling/Routing Algorithm

This section describes the Predictive Guarantee Overlay Scheduling (PGOS) algorithm, first discussing the general algorithm framework, then clarifying the concept of predictive guarantees and describing the algorithm itself. Formal analysis of buffer size under PGOS is given in [18], which shows PGOS also reduces the server/client buffer size requirement and make data transfer less bursty, by using statistical prediction, as compared with using average bandwidth prediction.

4.4.1 General Framework

An overlay network like the one in Figure 19 may be represented as a graph $G = (V, E)$ with n overlay nodes and m edges. An overlay node may be a server (i.e., data source) running on some host, a client (i.e., data sink), or a daemon for data routing. There may exist multiple distinct paths $P^j, j = 1, 2, ..., L$, between each server and client, where

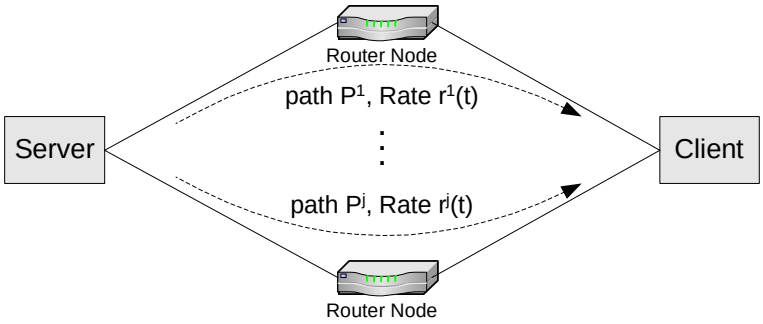


Figure 23: Overlay Routing and Scheduling Algorithm Framework.

$P^j = (V^j, E^j)$, $V^j = \{v_0, v_1, ...v_k, v_p \neq v_q \text{ if } p \neq q\}$ and $E^j = \{v_0v_1, ..., v_{k-1}v_k,$ where $v_pv_{p+1} \in E$, for all $0 \leq p \leq k-1\}$. As in [87], we make no assumptions about the placement of overlay nodes in the network. Rather, we assume that the middleware has determined some suitable placement.

For each overlay link, since network bandwidth varies over time, the service time of each application-level message is not known a priori and varies over time. The specific problem addressed by the PGOS algorithm is further illustrated in Figure 23, where multiple streams $S^j, j = 1, 2, ...N$ must be transmitted from Server s to Client c with 'best' predictive performance guarantees. Figure 24 illustrates a server that deliver multiple streams (in Queue 1, 2, ...) to a client via overlay paths 1, 2, etc. In this model, there is one scheduler and L path services (each service corresponds to one overlay path used to deliver packets, with service rate $r^j(t)$).

Applications specify stream utility in terms of the minimum bandwidths they require, or using Window-Constraints [100] requirement. A Window-Constraint is specified by the values x_i, and y_i, where y_i is the number of consecutive packet arrivals from stream S_i for every fixed window, and x_i is the minimum number of packets in the same stream that must be serviced in the window. The dynamics of the underlying network make it difficult to satisfy the minimum bandwidth guarantees required by the utility specifications described above, including for the guarantees associated with each scheduling window

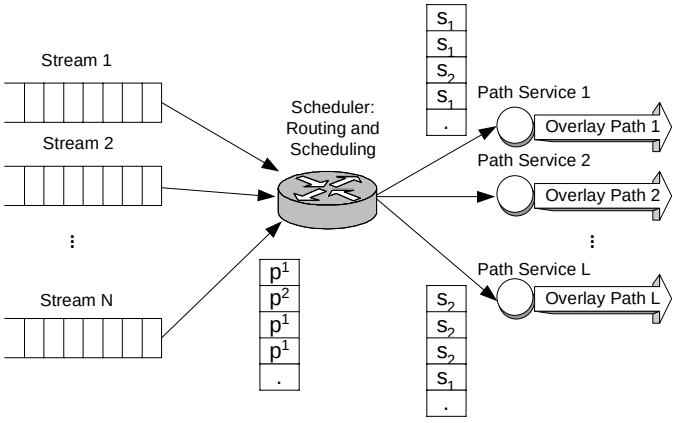


Figure 24: Routing and Scheduling on the Server.

t_w. We address this issue by asking applications to specify additional requirements of the following nature: ensure that the minimum bandwidth is met with some large probability P(e.g. 95%, 99%). This also means that 95% of the time, the window constraint will be satisfied. Given these specifications, assuming a packet size of s, and denoting the available bandwidth over a given path by $b^j(t)$, or simply b, (1) the available bandwidth distribution is described as the cumulative distribution function $F^j(b) = P\{avail\_bw \in (0, b)\}$, and (2) the service rate of the path service j is described as $r^j = r^j(t)$, where r^j varies over time.

4.4.2 Predictive Guarantee Overlay Scheduling/Routing Algorithm

The Predictive Guarantee Overlay Scheduling/Routing Algorithm (PGOS) supports two types of guarantees for stream utility specifications: probabilistic and 'violation bounded'. The former states that with some large probability P, stream S_i will receive the required bandwidth on the selected path. It also means that the stream S_i will receive the required bandwidth for at least $100P\%$ of the time. The latter states that the average number of packets that miss their constraint during each scheduling window can be bounded. Here, we first define a single path selection algorithm for predictive guarantees and then extend it to a scheduling algorithm that operates across multiple overlay paths.

4.4.2.1 Single Path Guarantee

The idea of single path selection is to choose the best path among all candidate paths for stream S_i, with some desired guarantee. Single path selection is important because there exist streams that are not easily mapped across multiple paths, an example being a stream with tight deadline/bandwidth requirements which would have to cope with synchronization issues and out of order arrivals when mapped across multiple paths.

Probabilistic guarantee The following is the probabilistic guarantee provided by the PGOS algorithm. Due to limit of space, proofs of Lemmas and Theorem appear in [18]:

Lemma 1 *Suppose during time $(t, t+t_w)$, where t_w is the length of the scheduling window, the available bandwidth distribution of server j is $F^j(b^j)$. Then, with probability $P = 1-F^j(x_i s/t_w)$, it is guaranteed that x_i packets will be served during the scheduling window t_w.*

Note that this guarantee essentially bounds the probability of insufficient throughput by $F^j(x_i s/t_w)$

'Violation bound' guarantees Another useful application-level utility specification is to bound some violation, such as the deadline miss rate. The following is the deadline 'violation bound' guarantee provided by PGOS, where Z is the number of packets that miss their deadlines during one scheduling window, given the rate distribution $G^j(r^j)$ in this scheduling window:

Lemma 2 *Given available bandwidth distribution $F^j(b)$, $E[Z]$ is bounded by $x_i \cdot F^j(b_0) - \frac{t_w}{s} \cdot M[b_0]$, where b_0 is the required bandwidth of Stream S_i, $b_0 = x_i s/t_w$, and $M[b_0]$ is the mean of b for all $b \leq b_0$. Both $F^j(b_0)$ and $M[b_0]$ can be easily computed from the available bandwidth distribution.*

Table 7: Precedence among Packets in Different Streams.

	Packet Ordering
1.	pkts scheduled on current path.
2.	pkts scheduled on other path:
2.1	Earliest deadline first.
2.2	Equal deadlines, highest window constraint first.
3.	pkts not scheduled:
3.1	Earliest deadline first.
3.2	Equal deadlines, highest window constraint first.

4.4.2.2 Guarantees for Multiple Overlay Paths

By combining the properties of multiple paths, PGOS can provide better guarantees to applications than those achievable on single paths. This is particularly relevant to large data transfers, where the parallelism achievable across multiple paths can be used to speed up data transfers as well as desired 'in flight' processing. Based on the two types of guarantees developed for each a single path, we now describe an overlay routing and scheduling algorithm that maps multiple streams across multiple paths (Figure 23). The algorithm schedules all packets of streams $S_i, i = 1, 2, ..., N$ such that the best guarantee is provided for the timely delivery of high utility streams, while other streams are delivered with less stringent guarantees. The PGOS algorithm, therefore, consists of two parts: (1) utility-based resource mapping and (2) path routing and packet scheduling.

Utility-Based Resource Mapping The resource mapping part of the PGOS scheduling algorithm (see Figure 25) finds the best proportion of stream S_i to be delivered via path P^j(**resource mapping**). The result is the generation of a *scheduling vector*, which is then used for routing and scheduling stream packets across multiple paths. The resource mapping step is executed when a new stream joins (or an existing stream terminates) or when the CDF of some path changes dramatically. A single resource mapping typically persists across many scheduling windows.

```
1     updateCDF(); /*update CDF using bandwidth/lossrate
              measurement in last scheduling window*/
2     if(previous scheduling vectors doesn't satisfy currentCDF){
        /*when new stream joins or CDF changes dramatically*/
3       Find best scheduling share Tp_i^j ; /*Tp_i^j is the number of
        packets of stream i scheduled to be sent on path j*/
        /*now rebuild scheduling vectors:*/
4       for( i = 1; i ≤ N; i++){
5         for( j = 1; j ≤ L; j++){
6           Tp^j += Tp_i^j; /*for path lookup vector*/
            /*Insert deadlines corresponding to Tpi(j) into VD^j*/
7           UpdatePathDeadlineVector(VD^j, Tp_i^j).
8         }
9       }
        /*build path lookup vector*/
10      VP=PathSchedVector(Tp^j);
        /*convert deadlines to stream scheduling vector*/
11      VS[]=StreamSchedVector(VD[]);
      }/*end of scheduling vectors update(when necessary)*/

12    while(in current scheduling window){
13      /*get next path according to V^p*/
        path=GetNextFreePath(V^p);
        /* get next packet to send based on V_s[p]:*/
14      if(getNextScheduledpkt(V_s[p]))
15        sendpkt(path, pkt);
16.     else if(pkt=getNextUnscheduledPkt(V_s)){
          /*other unscheduled pkt. Precedence rule 2 and 3.*/
17        sendpkt(path, pkt);
        }
      }
```

Figure 25: Scheduling Algorithm.

During each scheduling window, PGOS schedules packets based on the current scheduling vector and the stream precedence listed in Table 7. This table maintains the statistically optimal stream division scheme, while also utilizing additional available bandwidth whenever possible. For example, given two overlay paths' available bandwidth distribution $G^j(j = 1, 2)$, and two streams $S_i(i = 1, 2)$, the table is set up to divide each stream S_i into two sub-streams $S_i^1 + S_i^2$, where S_i^1 will be sent via path 1 and S_i^2 will be sent via path 2, such that their required performance guarantees are met. Note that S_i^j could be a null sub-stream, if necessary. We will send S_1^1 and S_2^1 via path 1, and send S_1^2 and S_2^2 through path 2.

Stream precedence is determined by the probabilities with which different streams' bandwidth requirements must be met. If streams S_i desire to receive their required bandwidths $100P_i\%$ of the time, then PGOS first finds the path that can satisfy the requirement of the most important stream (with highest P_i), then finds the path for the second most important stream, and so on. If there does not exist a single path that can satisfy stream S_i's requirement, then the stream S_i is divided into multiple parts S_i^j if this can satisfy stream S_i's requirement. If this still fails due to limited bandwidth, an upcall is made to inform the application that it is not possible to schedule this particular stream. The application can reduce its bandwidth requirement (e.g., from 95% to 90%) or try to adjust its behavior to the limited available bandwidth [15].

When a deadline violation bound guarantee is desired, PGOS works in a fashion similar to the probabilistic guarantees described above. When one or multiple streams join, PGOS begins with new stream with the highest deadline guarantee (i.e., with $\underset{i}{Minimum}[E[Z_i]]$), and attempts to find a path to meet its guarantee. If such a path does not exist, PGOS divides stream S_i(with x_i packets) into multiple parts S_i^j(with x_i^j packets) such that $\sum_{j=1}^{P} E[Z_i^j]\frac{x_i^j}{x^j} \leq E[Z_i]$, where $x_i = \sum_{j=1}^{P} x_i^j$ and $x^j = \sum_{i=1}^{N} x_i^j$. An alternative approach is to find a feasible division scheme without considering the ordering of $E[Z_i]$ and solve a mixed integer linear programming problem(MILP). However, this is not desirable since it may divide some

important stream (e.g., a control stream) into multiple sub-streams, thereby causing synchronization and delays due to potential packet re-ordering across multiple overlay paths. It is also an N-P hard problem. A detailed analysis of alternative approaches and their comparison are beyond the scope of this thesis.

Path Routing and Packet Scheduling While it may be computationally complex to find the best possible resource mapping, a more important issue is the affect of complex mappings on PGOS fast path performance. Here, during each scheduling window, PGOS needs to schedule packets according to the resource mappings encoded in scheduling vectors and according to the precedence table (see Table 7). The efficient data structures used by PGOS are depicted in Figure 25: the scheduler has a path routing vector VP, and each path service has one stream scheduling vector VS. The scheduling vector V encodes the currently best resource mapping scheme derived by the resource mapping step. The lookup vector VP is the vector the scheduler uses to switch between the different overlay paths. As derived in the resource mapping step, path j is assigned x^j packets, so path j is assigned x^j virtual deadlines $D_p[k] = t_w/x^j \cdot (k-1)$. Virtual deadlines are used to maintain the desired resource mapping proportion. That is, VP contains the ordering to be used for visiting each path, based on virtual deadlines.

To illustrate, consider a concrete example with two streams and two overlay paths. Stream S_1 has 5 packets in one scheduling window that are mapped to path 1. Stream S_2 has 10 packets in one schedule window, where 4 of them are mapped to path 1, while another 6 packets are mapped to path 2. In this example, path 1 has 9 packets to deliver, and path 2 has 6 packets to deliver. Thus, VP=[1,2,1,2,1,1,2,1,2,1,1,2,1,2,1]. When the scheduler switches between the overlay paths, the path lookup vector ensures that three fifths of the time, it will visit path 1, and two fifths of the time, it will visit path 2. Stated more generally, when the scheduler visits path j, it uses the stream scheduling lookup vectors VS^j to select the streams to which to send packets.(VS is essentially a lookup

table where each row corresponds to one path). The lookup vectors VS^j are based on the deadlines of all of the packets (from multiple streams) to be sent on path j. In the example, path 1 has nine packets, and the deadlines of these 9 packets are for S_1, S_2, S_1, S_2, S_1, S_2, S_1, S_2, and S_1 respectively. Thus, $VS^1 = [1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1]$.

While using VS for path mapping, PGOS schedules packets based on both VP and VS. That is, once it has selected path j, PGOS sends packets over it according to VS. Specifically, it selects a packet to send based on the stream scheduling lookup vector VS^j and the precedence table (Table 7). First, it sends the packet scheduled on the current path $j$some other path that has the earliest deadline. Equal deadlines are broken by the window constraint x/y (highest window constraint first) and further ties are broken arbitrarily. When all scheduled packets have been sent out and there are still free paths to utilize, PGOS sends out other unscheduled packets according to their deadlines and window constraints. Whenever a path is blocked, the scheduler switches to the next path immediately, in order to best utilize other available resources. Because of the high cost of blocking, timeouts and exponential backoff are used to avoid sending multiple packets to a blocked path.

The following theorem states more precisely the guarantees provided by PGOS:

Theorem 1 *If there is a feasible schedule for PGOS to deliver streams $S_i, i = 1, 2, ..., N$ over paths $P^j, j = 1, 2, ...L$ during scheduling window $(t, t + t_w)$ with bandwidth guarantees, then stream S_i's window constraint will be met with probability P_i.*

4.4.3 Risk Based Resource Mapping

The PGOS algorithm involves two steps: resource mapping step which finds the best proportion of streams S_i to be delivered via path P^j, and the packet routing and scheduling step which does the fast routing and scheduling according to the scheduling vector and stream precedence. In the previous sections, we consider how to maximize the utility in the resource mapping step. In real world, maximizing utility is often not what is really desired when risk is another factor to consider. As demonstrated in Chapter 2, risk attitude affect

people's decision whether a high utility solution coupled with high risk is really better than a mediocre utility solution coupled with low risk. We extend the methodology proposed in previous sections with the risk-attitude sensitive availability management in this section to demonstrate how to consider risk attitude when dealing with network risks.

Consider multiple distinct network paths from the source to the destination $P^j, j = 1, 2, ..., L$, and we need to find out the right proportions S^j of stream S which is to be delivered by path P^j. The vector S^j is used in the path routing and packet scheduling step. Because of the fluctuation of the network resource, we don't know if the actual network resource will be enough to deliver the required bandwidth, however, we do know from statistical bandwidth prediction that with probability of $P_{S^j} = 1 - F^j(S^j)$, path P^j will provide at least S^j Mbps, which is the outcome of the lottery and $1 - F^j(S^j)$ is the probability that this outcome will happen.

Table 8: Lottery Outcomes

Options	Performance	Guarantee
$L1$: (pP,hG)	60% of S^j	99%
$L2$: (hP,pG)	99%	97%
$L3$: (gP,gG)	80%	98%

Table 9: Preferences of Outcomes

Customer	Preference
$C1$: ('like high guarantee')	*L1>L3>L2*
$C2$: ('like high performance')	*L2>L3>L1*
$C3$: ('mediocre perf. and guarantee')	*L3> L1>L2*

To map the resource based on risk preference/attitude, we first elicit the customer's risk attitudes using typical possible outcomes (see Table 8) and the customer's preferences for these outcomes (see Table 9). Here, $L1$ represents an outcome with low bandwidth but very high bandwidth guarantee, $L2$ represents an outcome with high bandwidth with poor bandwidth guarantee, and $L3$ represents an outcome with mediocre bandwidth with

mediocre guarantee. Again, different customers have different risk attitudes, for example, customer $C1$ prefer $L1$ to $L3$ and prefers $L3$ to $L2$, since he is more risk averse person.

Let $U(S^j)$ be the utility generated if path P^j can provide S^j Mbps bandwidth in the next second and define von Neumann-Morgenstern utility as $vnmU(S^j) = P_{S^j}U(S^j)^\alpha/\alpha$ using CRRA vNM utility function. According to the user's preference, we use the same method proposed in Chapter 2 to find the value of α. The optimal resource mapping for a customer with risk preference determined by α, should be $S^j, j = 1, 2, 3, ..., L$ which maximizes von Neumann-Morgenstern utility:

$$vnmU = Max\{\sum_{j=1,2,...,L} vnmU(S^j)\} = Max\{\sum_{j=1,2,...,L} P_{S^j}U(S^j)^\alpha/\alpha\}$$

4.5 Experimental Evaluation

In this section, we first demonstrate the importance of choosing the right path based on statistical predictions. Then, we evaluates and analyzes IQ-Paths with three types of applications: (1) the SmartPointer system [102] for distributed collaboration and interactive program steering, (2) the GridFTP [2], a high-performance and reliable data transfer protocol widely used in the Grid community, for reliable parallel data transmission in wide area networks, and (3) MPEG-4 Fine-Grained Scalable video streaming.

Our testbed emulates a realistic wide area setting, using the EmuLab facility [93]. NLANR traces are used to inject representative cross-traffic [61]. If not stated otherwise, the overlay server N-1 has two overlay paths to reach the client N-6(Figure 26), and the background traffic and data traffic share the common link between N-3 and N-5, and the link between N-2 and N-4. All link capacities are 100Mbps, which is the current up-limit of Emulab.

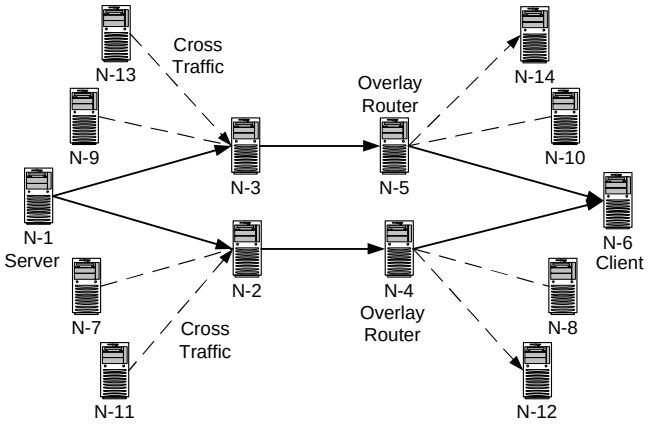


Figure 26: IQ-Paths Testbed. The link connecting each pair of nodes is fast Ethernet. Cross traffic is injected by Node N-9 to N-14. Overlay routers are placed at Node N-4 and N-5, so that overlay paths and cross traffic paths share the same bottleneck (N-3 to N-5 and N-2 to N-4).

Table 10: Importance of Choosing the Right Path. 'X%' means the Xth percentile point.

	Throughput($Mbps$)		Jitter(ms)	
	95%	99%	mean	std
Atom-pathb	2.47	1.93	1.70	6.1
Atom-patha	3.23	3.23	0.82	1.3
Bond1-pathb	17.23	13.02	1.70	6.1
Bond1-patha	22.04	22.03	0.83	1.3

4.5.1 PGOS Evaluations with SmartPointer

4.5.1.1 Importance of Choosing the Right Path

We first evaluate the importance of choosing the 'right' path for an application's data streams. In this evaluation, cross traffic based on trace files obtained from NLANR is injected into two overlay paths. The average available bandwidth on Path B is higher than that on Path A, but it has larger variation compared to Path A. Two streams of the SmartPointer (streams 'Atom' and 'Bond1') are transferred from Node1 to Node6 over either of these two overlay paths. In Table 10, the Atom-patha row and Bond1-patha row are the throughputs and jitters of streams Atom and Bond achieved if we utilize Path A, and Atom-pathb and Bond1-pathb are throughputs and jitters achieved if we use Path B. Although Path B

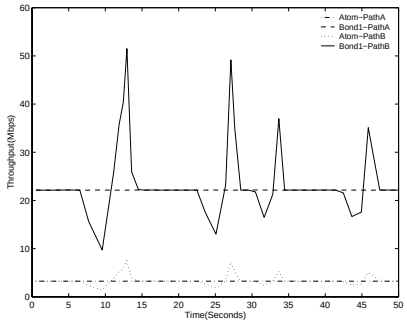


Figure 27: Throughput on Each Path. Although path A has less mean available bandwidth than path B, it is preferable for streams ‘atom’ and ‘bond1’

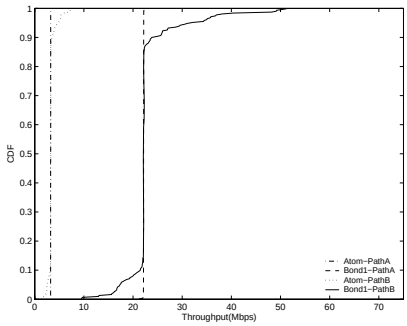
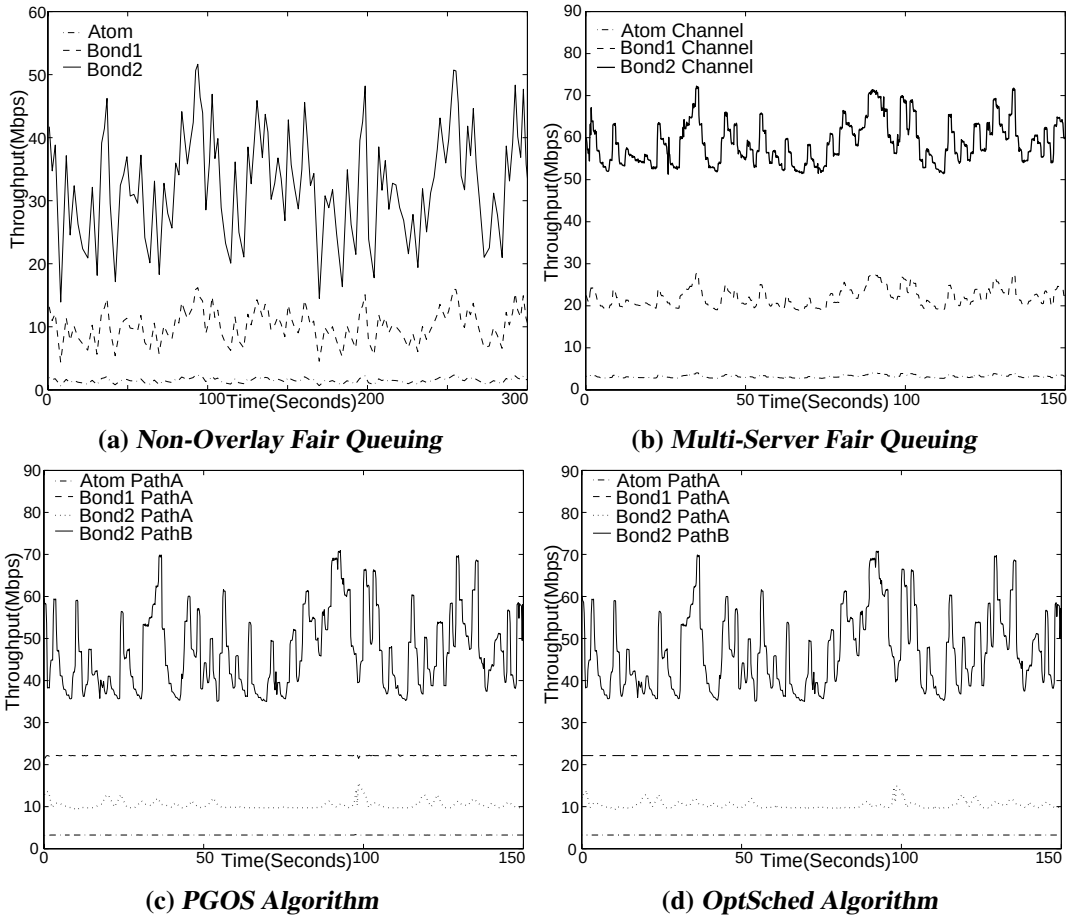


Figure 28: Throughput CDF on Each Path. Bandwidth on path B is more dynamic than bandwidth on path A.

has higher average bandwidth, its higher variation causes unstable bandwidth (e.g., for 95% of time, stream Atom can only obtain 2.47Mbps from Path B, while it can obtain 3.23Mbps from Path A.) Unstable bandwidth also results in bursty transmission behavior, large queue lengths on the server side, and higher jitter, none of which are desirable for this remote collaboration application. Note that low jitter is particularly important for real-time applications like remote scientific visualization, as it provides smoother data streaming and also reduces total buffering. As shown in Table 10, the two streams can achieve much lower average jitter and lower standard deviation in jitter if PathA is chosen instead of PathB (0.82 vs. 1.7 for average jitter and 1.3 vs. 6.1 for standard deviation.)

This experiment demonstrates the importance of assessing the distribution of available bandwidth to meet application-level service requirement vs. assessing average bandwidth

values. When we transfer large data volumes, average bandwidth is one important factor, but it is not a sufficient one. Specifically, by using the distribution of available bandwidth, PGOS can find the path to transfer application data that has the best predictive guarantee. We next discuss further improvements in attainable end-to-end bandwidth, by using PGOS to schedule traffic across concurrent network paths.

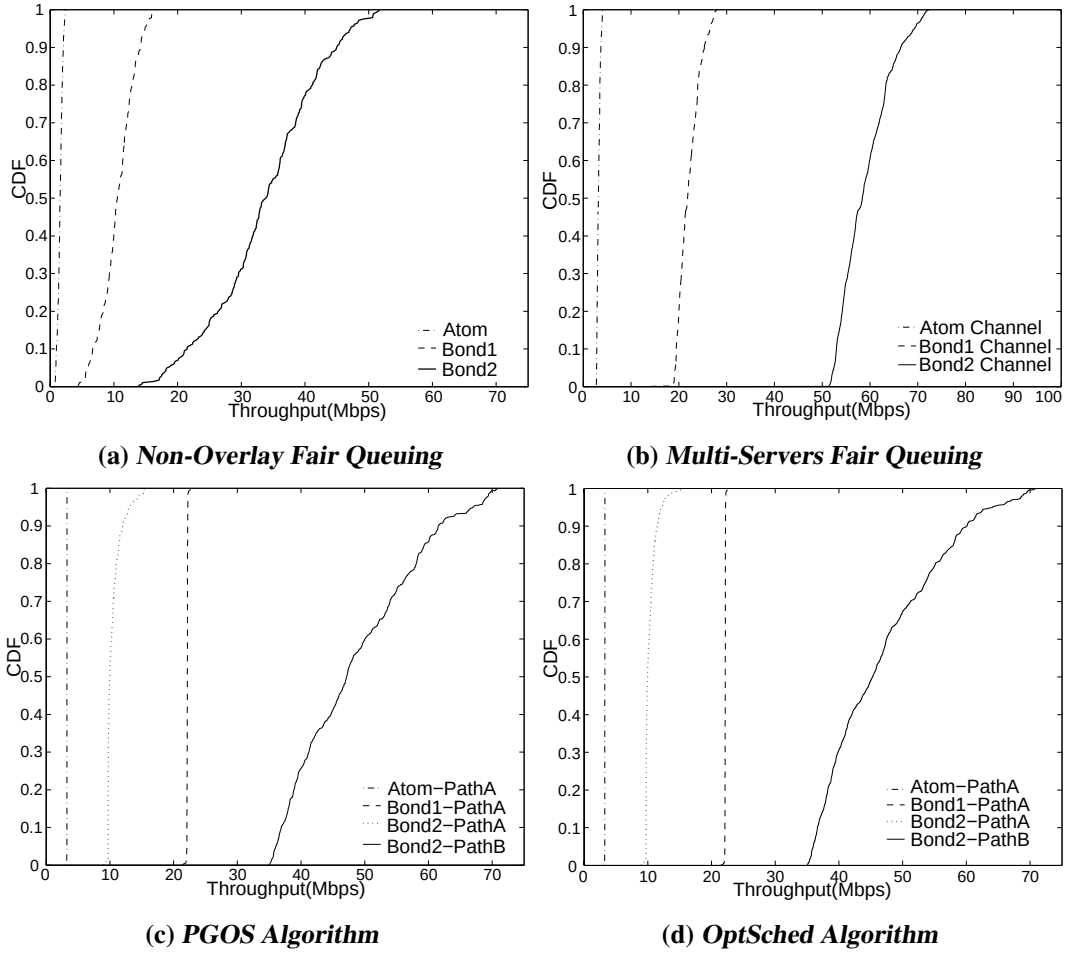


(a) *Non-Overlay Fair Queuing* **(b) *Multi-Server Fair Queuing***

(c) *PGOS Algorithm* **(d) *OptSched Algorithm***

Figure 29: Throughput Time Series Comparison of Three Algorithms.

This set of experiments evaluates PGOS' multi-path message routing and scheduling performance using the SmartPointer distributed collaboration application. The purpose is to see how the algorithm can guarantee some critical stream's required throughput while also providing high throughput to non-critical streams. Consider the SmartPointer server issuing three streams (Atom, Bond1, and Bond2) to remote clients. Streams Atom and Bond1 are



(a) *Non-Overlay Fair Queuing* **(b) *Multi-Servers Fair Queuing***

(c) *PGOS Algorithm* **(d) *OptSched Algorithm***

Figure 30: Throughput CDF Comparison of Three Algorithms.

data about all atoms and those bonds that are in the observer's immediate graphical view volume, whereas stream Bond2 contains the bonds outside the observer's current view. Therefore, Streams Atom and Bond1 are important and must be delivered in real-time (25 frame/sec) for effective collaboration, but stream Bond2 is less critical (e.g., it may be important when the observer rapidly changes his/her viewing angle.)

Three on-line message transfer algorithms are evaluated and compared to meet this application's needs: (1) transfer all messages over one single path based on normal Fair Queuing (WFQ), (2) transfer messages over two paths with multi-server Fair Queuing(MSFQ) [9], and (3) transfer messages over two paths using the proposed PGOS routing and scheduling algorithm. The input (utility requirements) to PGOS are 3.249Mbps with 95% predictive

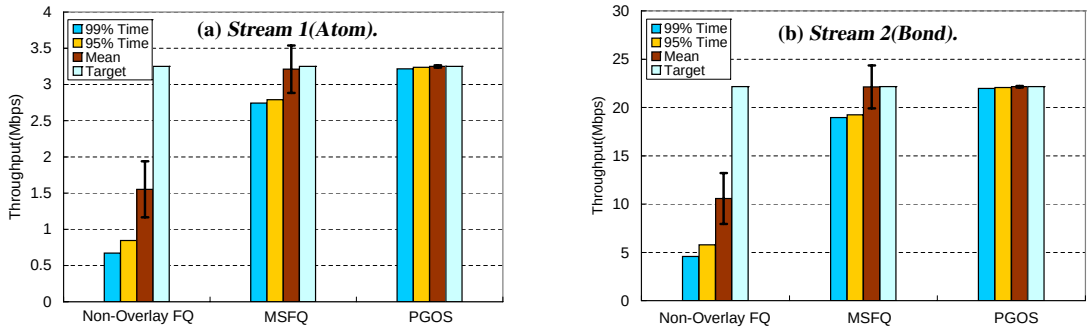


Figure 31: Throughput Achieved by Three Algorithms: Target, Mean, 95% of the time, 99% of the time, and Standard deviation (represented by the vertical bars).

guarantee for stream Atom and 22.148Mbps with 95% predictive guarantee for stream Bond1. We also compare these results with a near-optimal off-line algorithm, termed OptSched [18], which assumes that we know available bandwidth a priori. Although this off-line algorithm cannot be used in practice, it can be used to gauge the absolute performance of PGOS.

The results of using these four algorithms appear in Figure 29. Figure 29a depicts the throughput of 3 streams attained by the WFQ algorithm on Path A, which has higher available bandwidth than Path B with larger variance. Multi-Server Fair Queuing (MSFQ) can maintain the proportion of throughput shared by the three streams quite well (see Figure 29b), but because of its inaccurate average bandwidth prediction, it fails to provide the required throughput to the two critical streams Atom and Bond1. Both streams exhibit substantial throughput fluctuation. In comparison, the PGOS algorithm successfully provides very stable throughput to these two critical streams. Furthermore, note that in Figure 29c, the throughput of stream Bond2 is not compromised. This stream is divided by PGOS into two substreams (Bond2-PathA and Bond2-PathB), and the average throughput of stream Bond2 is almost the same as that achieved by MSFQ.

The cumulative distributions of throughput of the three streams under the three algorithms are given in Figure 30. PGOS provides the two critical streams at least 99.5% of their required bandwidth for 95% of the time. MSFQ can only provide about 87% of their

required bandwidth for 95% of the time. For example, stream Bond1 requires 22.148Mbps, and the actual 95th percentile of the achieved bandwidth is 22.068Mbps under PGOS, but it is only 19.248Mbps under MSFQ. The standard deviations of bandwidth experienced by the two critical streams appear in Figure 31. Although stream Bond2 has slightly larger standard deviation with PGOS, the two critical streams Atom and Bond1 experience much lower standard deviations.

Both Fair Queuing and Multi-Server Fair Queuing try to allocate bandwidth in a proportional based manner according to predicted bandwidth, but they require exact values of end-to-end bandwidth, which are hard to attain. As a result, although both of these two algorithms can successfully maintain the proportion of the bandwidth allocated to multiple streams, they cannot provide specific bandwidth to a particular stream. In comparison, PGOS relaxes the prediction assumption, only asking if we can obtain certain bandwidth with some high probability. This is not only easier to predict, but also directly provides the functionality needed by applications.

All three algorithms experience certain overheads when routing single streams over multiple paths, because of packet reordering and delays of head-of-line packets. PGOS reduces this overhead by using a single path for one stream whenever possible, especially for streams with higher priorities. Simply speaking, unlike MSFQ which provides the two critical streams less than required bandwidths when the network is congested and more than required bandwidths when the network is free of congestion, PGOS routes and schedules packets such that the two important streams obtain stable required bandwidths no matter whether or not one path is congested. As a result, the application frame jitter is also reduced from 2.0ms (with MSFQ) to 1.4ms (with PGOS).

In summary, these experiments show that with PGOS routing/scheduling, critical streams' required throughput can be guaranteed most of the time. This is done without compromising the average throughput experienced by non-critical streams. A case in point in our experiments is non-critical stream Bond2, which still receives almost the same average

Table 11: Comparing Three Routing/Scheduling Algorithms.

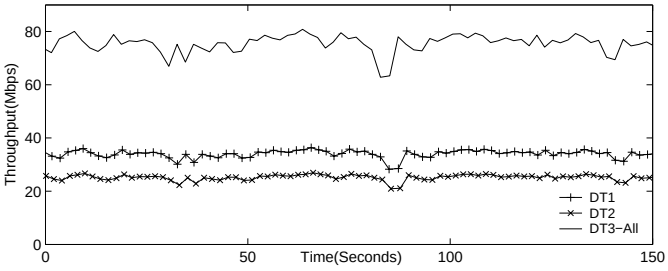
	Throughput(Mbps)				
	95 Perct.	99 Perct.	Mean	Std.	Target
Atom	0.846	0.672	1.5524	0.3863	3.249
Atom^F	2.789	2.744	3.2111	0.3273	3.249
Atom^P	3.236	3.216	3.2487	0.0150	3.249
Atom^O	3.240	3.239	3.2489	0.0058	3.249
Bond1	5.768	4.569	10.5843	2.6348	22.148
Bond1^F	19.248	18.946	22.1300	2.2321	22.148
Bond1^P	22.068	21.959	22.1476	0.0790	22.148
Bond1^O	22.138	22.139	22.1477	0.0273	22.148
Bond2	18.297	14.486	33.7021	8.3949	70.340
Bond2^F	52.446	51.76	59.0786	8.0397	70.340
Bond2^P	45.782	45.038	59.0588	9.5765	70.340
Bond2^O	45.757	45.019	59.0583	9.5587	70.340

throughput under PGOS as under MSFQ.

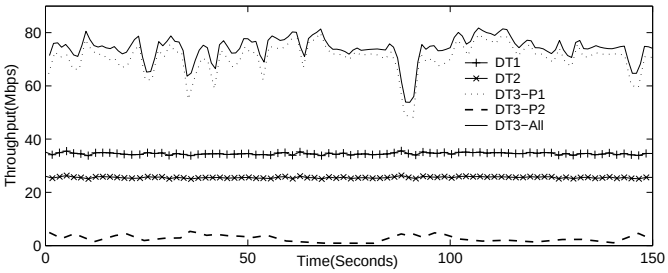
4.5.2 GridFTP Experiments

GridFTP [2] is widely accepted as one of the common data transfer services available for high performance applications, with extension to the FTP protocol including parallel data-transfer, SPAS(Striped Passive), and SPOR(Striped Data Port). In this subsection, we present our experiences with IQ^{PG}-GridFTP, which strengthens our previous work [15] by including support for PGOS-enabled parallel file transfers. IQ^{PG}-GridFTP generalizes the publicly available wu-ftpd [104] server to support the GridFTP protocol extensions for parallel transfers and implements the Partitioned and Blocked data layout options to distribute file contents across the connections in addition to the PGOS layout. A partitioned data layout is one where contiguous chunks of file are distributed evenly across all the connections for transfer, while a blocked data layout is one where data blocks (each of size block-size) are distributed in a round-robin fashion.

We use a climate database in our experiment as simulation of the Earth System Grid II [7]. Each record in this database has three data components: (1) the numeric data



(a) *GridFTP Throughput.*

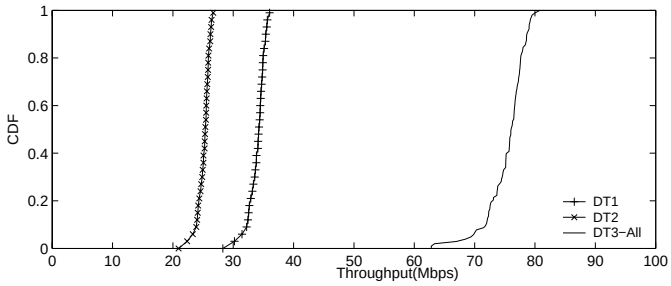


(b) *IQ^{PG}-GridFTP Throughput. Line DT3-All is the throughput achieved by stream DT3(sum of throughput on two paths: DT3-P1 and DT3-P2).*

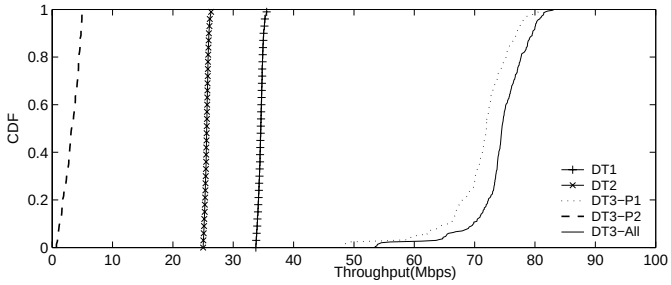
Figure 32: Throughput Achieved by GridFTP and IQ^{PG}-GridFTP

(approximately 172.8KB, denoted by 'DT1'), and (2) and (3) are low resolution images (128KB, denoted by 'DT2') and high resolution images (384KB, denoted by 'DT3'), respectively. GridFTP and IQ^{PG}-GridFTP are configured to concurrently transfer file records over two overlay paths. For such transfers, we want to ensure that the numeric data and low resolution images receive their required bandwidths of at least 25 records/second for real-time data streaming. In addition, we also want to fully utilize bandwidth to transfer high-resolution data.

Experimental results are depicted in Figures 32 and 33. From these measurements, it is apparent that IQ^{PG}-GridFTP can ensure that the streams DT1 and DT2 receive their required bandwidths consistently, while stream DT3 is transferred as fast as possible. In comparison, standard GridFTP splits the dataset into blocks allocated to the multiple connections for transfer, but when the available bandwidth of any path is low, all types of data have to compete with each other. This causes the important data streams to not receive their required bandwidths during these periods. Quantitatively, stream DT1 achieves



(a) *GridFTP Throughput CDF.*



(b) *IQ^{PG}-GridFTP Throughput CDF.*

Figure 33: GridFTP and IQ^{PG}-GridFTP Throughput CDF Comparison

33.94Mbps average throughput using GridFTP with a large standard deviation (1.4297), while using IQ^{PG}-GridFTP, it achieves 34.55Mbps average throughput with a small standard deviation (0.4040). Similar results are observed for stream DT2. Note that here the network can provide almost the total throughput required by the application. If the network cannot provide such throughput, then the two streams DT1 and DT2 obtain even less bandwidth using GridFTP, as they have to compete with stream DT3 for the same limited bandwidth. In summary, with PGOS, IQ^{PG}-GridFTP can protect more important streams from competing with other less important streams, while also scheduling less important streams to be delivered when there exists sufficient bandwidth. Applications have full control over deciding how much bandwidth will be allocated for a particular stream and what kind of guarantee is for each stream.

4.5.3 Multimedia Streaming Experiments

Video and audio streaming over the Internet are known to be important applications. Because the dynamic behavior of the Internet makes it difficult to provide consistently good

Table 12: Comparing GridFTP and IQ^{PG}-GridFTP: Target, 95 Percentile, Mean and Standard Deviation .

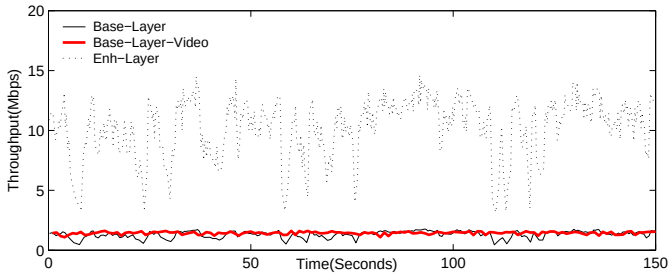
Stream Name	Throughput(Mbps)			
	Target	95 Perct.	Mean	Std.
DT1^{O}	34.56	31.431	33.9411	1.4297
DT1^{P}	34.56	33.869	34.5505	0.4040
DT2^{O}	25.60	23.282	25.1415	1.0590
DT2^{P}	25.60	25.094	25.5990	0.2993
DT3^{O}	76.80	69.393	75.4246	3.1770
DT3^{P}	76.80	65.287	74.3577	4.7668

quality of streaming video/audio, layered coding and multiple description (MD) provide layers of encoded video. Both layered and MD coding can leverage the QoS enhancements offered by PGOS, and in this section, we evaluate the performance of PGOS used with MPEG-4 Fine-Grained Scalable (MPEG-4 FGS) layered video coding [42].

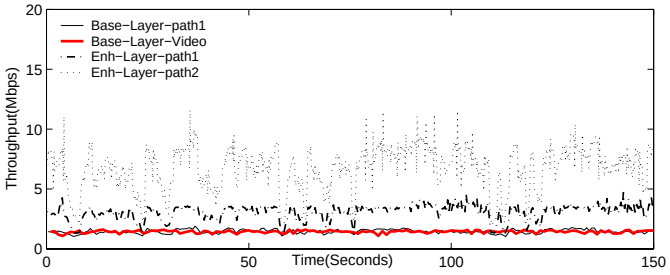
The MPEG-4 FGS framework consists of a base layer and one or two enhancement layer. The base layer is generated by motion estimation/motion compensation and entropy coding with fixed quantization step size. The SNR FGS enhancement layer adds DCT coefficients with reduced quantization step size and leads to more accurate DCT coefficients and higher video quality. The Temporal FGS enhancement layer improves temporal resolution by providing a higher frame rate and smooth motion. The base layer is the most important set of data, and its bandwidth requirement should be consistently provided for smooth playback. Receivers can subscribe to as many enhancement layers as possible to maximum video quality, but these layers are less important and may be dropped at when there in insufficient available bandwidth.

The base layer and the enhancement layer require 1.4820Mbps and 11.2901Mbps average bandwidths, respectively. Since encoded video exhibits variable throughput, the input parameter for PGOS is a 95% prediction guarantee of 1.22Mbps for the base layer, which corresponds roughly the 95 percentile of the actual bit rate of the base layer. There is no

requirement for the enhancement layer, i.e., we would like to transmit the enhancement layer using the remaining available bandwidth.



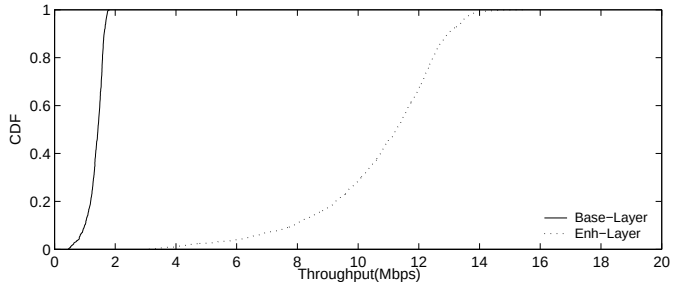
(a) *Multi-Servers Fair Queuing(MSFQ)*



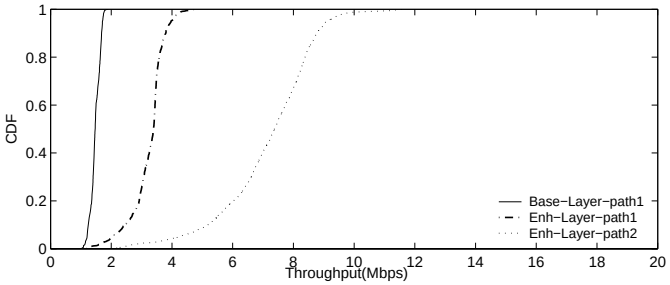
(b) *PGOS*

Figure 34: Throughput Time Series of MSFQ and PGOS Algorithms.

Experimental results appear in Figures 34 and Figures 35. In Figure 34, the thick red line is the bandwidth of the encoded base layer, and the solid black line is the delivered bandwidth of the base layer. Comparing these two graphs, PGOS can deliver about the bandwidth required by the base layer. In comparison, since mean bandwidth cannot be predicted well, with MSFQ, for some time, the achieved bandwidth of the base layer is significantly less than the required video bit rate. More precisely, PGOS provides 1.20Mbps for 95 percent of the time while MSFQ provides only 0.81Mbps for 95 percent of the time.Further, PGOS provides at least 1.22Mbps for 93 percent of time which is very close to our bandwidth guarantee requirement (i.e., at least 1.22Mbps for 95 percent of the time). In comparison, MSFQ provides at least the required bandwidth for only 78 percent of the time.



(a) *Multi-Servers Fair Queuing(MSFQ)*



(b) *PGOS*

Figure 35: Throughput CDF Comparison of Three Algorithms.

Table 13: Comparing Throughput of MSFQ and PGOS Algorithms: 95 Percentile and Mean

	Throughput(Mbps)	
	95 Perct.	Mean
Base Layer^F	0.81722	1.3693
Base Layer^P	1.2026	1.4761
Enh. Layer^F	6.5250	10.7690
Enh. Layer^P	6.2931	10.3910

4.5.4 Risk-Based Resource Mapping

In this section, we use simulation based results to illustrate how risk attitudes affect resource mapping. Consider an enterprise able to use two alternative paths between two data centers. Normally, the cost and reliability of these multiple paths are different and at the same time, multiple streams have different probabilities for service guarantees, for example, one high end optical link which has reserved (guaranteed) bandwidth and one common Internet connection with best effort service guarantees. The cost of these links are different,

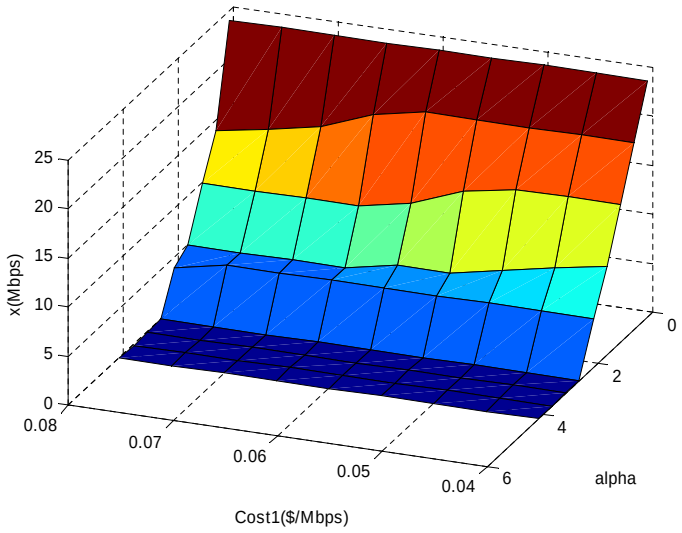


Figure 36: Optimal Bandwidth Allocation under Different Values of α and $Cost1$. x is the bandwidth allocated to the high end link.

and we use the data from [41] as a guideline for cost estimation: $\$120.000$ per Month for a OC12(622Mbps) link and $\$4.000$ per Month for a T3(43.232Mbps) link (a range of costs is considered in the experiments). Utility is defined as $U(S^j) = benefit - cost$, where $benefit = 0.1 * S^j$. The bandwidth required by stream S is 40Mbps. Figure 36 shows the optimal bandwidth allocation under different $Cost1$, the cost of the high end link and different values of alpha, which defines whether a customer is risk averse, risk neutral or risk seaking. The z-axis is the bandwidth we need to allocate to the high end link (so, $40Mbps - x$ is the bandwidth we need to allocate to the common Internet connection). The cost of the common Internet connection, $Cost2$, is fixed: $Cost2 = \$0.015Mbps$. Clearly, when the value of α decreases (in the range of 0.15 to 4.15), the decision will be more risk averse and more bandwidth will be allocated to the high end link, despite its high cost. When the cost of the high end link $Cost1$ increases, less bandwidth will be allocated to this link, given same value of α. Interestingly, when α is extremely small or high (means very risk averse or very risk seaking), the cost of the high end link does not affect the bandwidth allocation decision: when α is small, all bandwidth will be allocated to the high end link; and when α is high, zero bandwidth is allocated to the high end link.

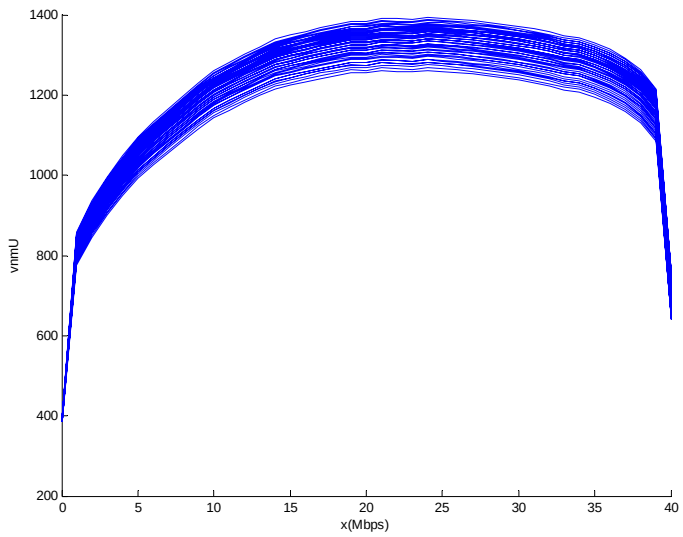


Figure 37: vNM Utility under Different x Value, When $alpha = 0.15$. Each combination of different $Cost1$ value and $Cost2$ value has one curve.

Figures 37 to 41 depict how the von Norman-Morgenstern utility changes when x changes. Each combination of different $Cost1$ value and $Cost2$ value has one curve in each figure. The highest point of each curve, where we obtain highest vNM utility, represents the optimal bandwidth allocation at certain values of $Cost1$, $Cost2$, and α. This series of graphs shows the clear trend that when α increases, the highest points of the curves shift to the left, meaning less bandwidth should be allocated to the high end link, and vice versa. These graphs also show that different allocations of bandwidth affect vNM utility dramatically, which could range from 10 to 900 (e.g., see Figure 40). This implies that if we do not consider the risk attitude when optimizing bandwidth allocation (the bandwidth mapping step), the outcome could much diverge from the actual customer risk preference. As also illustrated in Chapter 2, it is important to consider risk attitude when risk is unavoidable, when considering failures and fluctuations of either computational or network resources.

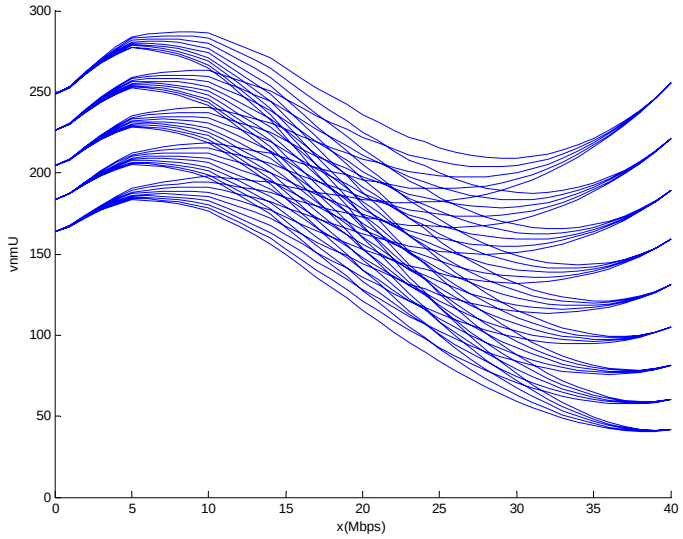


Figure 38: vNM Utility under Different x value, When $alpha = 1.65$. Each combination of different $Cost1$ value and $Cost2$ value has one curve.

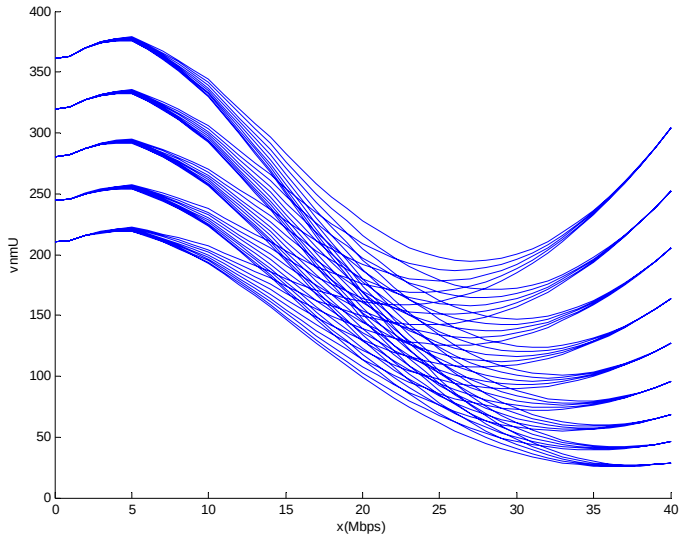


Figure 39: vNM Utility under Different x Value, When $alpha = 2.15$. Each combination of different $Cost1$ value and $Cost2$ value has one curve.

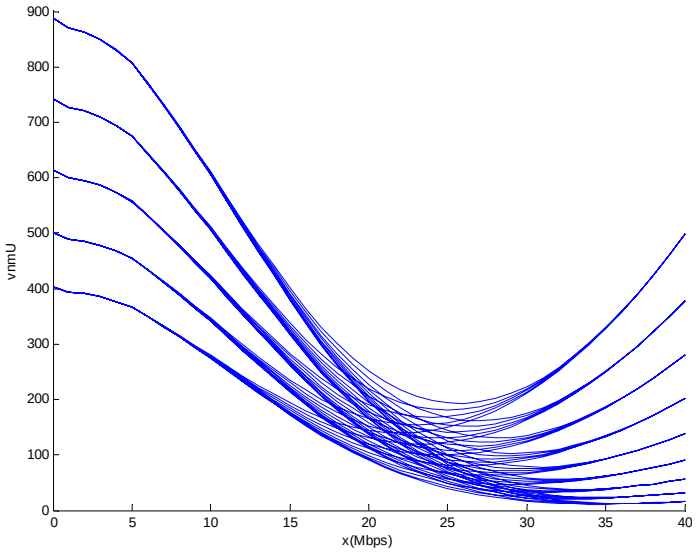


Figure 40: vNM Utility under Different x Value, When $alpha = 3.15$. Each combination of different $Cost1$ value and $Cost2$ value has one curve.

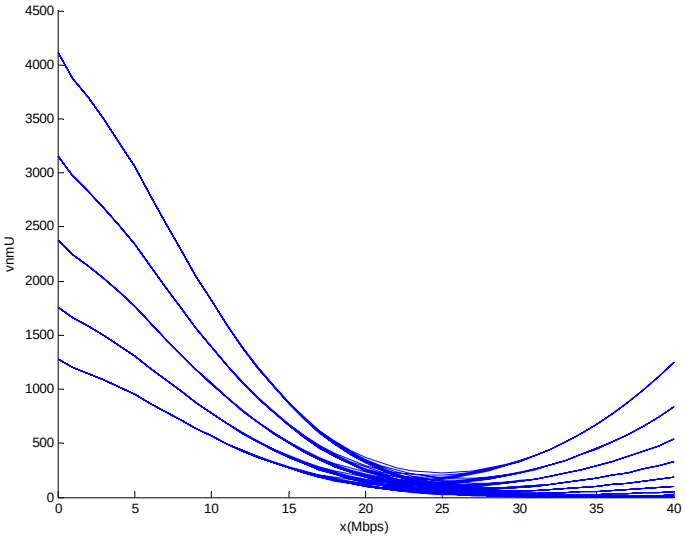


Figure 41: vNM Utility under Different x Value, When $alpha = 6.15$. Each combination of different $Cost1$ value and $Cost2$ value has one curve.

CHAPTER V

RELATED WORK

5.1 Availability and Reliability

Traditional Fault-Tolerance. Redundancy is probably the earliest form of fault-tolerance; the approach popularly known as the active replication approach is well-studied, and a thorough description appears in [68]. Log-based recovery is well-known in the database domain. Here, a failure is handled with an undo-redo log [37]. Fault-tolerance has also been studied in the context of transactions [103] and distributed systems [82]. Dynamically trading consistency for availability is proposed in [105] using a continuous consistency model. A number of factors distinguish our approach from these traditional mechanisms, the first and the foremost being its utility-awareness. Another distinction is our ability to use failure prediction to reduce the overhead of ensuring high-availability.

Failure Detection & Prediction. [85] focuses on the implementation of fault detection, and proposed a scalable fault detection/collection framework. More recently, researchers in the autonomic domain have used statistical monitoring techniques to detect failures in component-based Internet services [30, 106]. MSET or multi-variate state estimation techniques [106] constitute an early warning system that enables failure prediction with low false alarm probability and has been successfully applied to the thermal control domain, and more recently, to software aging problems, including predicting memory leaks, data corruption, shared memory pool latching, etc. In [25], instrumentation data is correlated to system states using statistical induction techniques to identify system-level metrics that correlate with high-level performance states. In addition, these techniques are used to forecast service level objective violations, with prediction accuracy reported to be around 90%. Our system provides a framework in which several such failure detection

and prediction techniques can be implemented to provide high-availability while imposing a low-overhead. Often, failures occur at client sites are hard to be reproduced offsite for diagnosis. Triage [94] provides a light-weighted method which automatically detects and diagnoses software failure diagnosis at the very moment of failure, at client sites. Such methods would be potentially useful to shed light on failure prediction and avoidance.

Fault-Tolerant Distributed Information Systems. Stars [82] presents a fault-tolerance manager for distributed application, using a distributed file manager which performs actions like message backups and checkpoints storage for user files. Its reliance on causal and atomic group multi-cast, however, demands additional solutions in the context of today's widely geographically distributed enterprise systems [21].

MTTR may be improved with solutions like Microreboot [20], which proposes a fast recovery technique for large systems. It is based on the observation that a significant fraction of software failures in large-scale Internet systems can be cured by rebooting. While rebooting can be expensive and cause nontrivial service disruption, microrebooting is a fine-grain technique for surgically recovering faulty application components, without disturbing the rest of the components of the application. Our work could benefit from such techniques.

GSpace [74] and replica management in Grids [78] studied dynamic data replication policy and modeling in distributed component-based systems when multiple replicas of data are desired, e.g., for global configuration data, or in a highly dynamic environment, to improve availability. For this kind of data replication management, efficient read-one write-all protocol [67] can be used when updates of the replicated data occur frequently.

IFLOW's techniques may be directly compared to the fault-tolerance offered in systems like Fault-Tolerant CORBA [58, 63], Arjuna [64] and REL [33], which replicate selected application/service objects. Multiple replicas allow an object to continue to provide service even when one of its replicas fails. Passive replication is also provided. Here, the system records both the state of the currently executing member (primary member) and the entire

sequence of method invocations. While CORBA focuses on the client-server model of communication, recent systems like Borealis [5] and SMILE [86] have focused on fault-tolerance for applications that process data streams. The former uses replication-based fault-recovery, and the authors propose to trade consistency for recovery time. The latter proposes the soft-checkpointing mechanism that can be used to implement a low-overhead passive replication scheme for fault tolerance. We differ from such earlier work because of our explicit consideration of system utility for managing system availability, and because our system also provides a framework for incorporating failure prediction techniques.

Utility-Functions. The specific notions of utility used in this thesis mirror the work presented in [99], which uses utility functions for autonomic data-centers. Autonomic self-optimization according to business objectives is studied in [1], and self-management of information flow applications in accordance with utility functions is studied in [49]. A preliminary discussion about availability-aware self-configuration in autonomic systems appears in [22]. Our middleware carefully integrates the ideas from the above systems and other domains to build a comprehensive framework for fault-tolerant information flows.

5.2 Service Management and Risk Management

IT service management driven by business policies is a relatively new area. Buco et al. present SLA management system that is based on business-objectives [12]. Salle et al. propose a solution to minimize the exposed business impact of service level violation [76]. They further present the Management by Business Objective (MBO) technology for IT management that can take into account strategic business objectives [77] and they specifically apply this approach to incident management domain [6]. Yuan et al. use performance modeling with performance profiling to translate Service Level Objectives to lower-level resource requirements. To manage complex enterprise systems, state-space approach can be used to divide a large system state-space into partitions that are more manageable [51].

In the context of design, Sahai et al. propose a policy-based model for automated configuration management [75]. It automatically creates a suitable configuration and a workflow to deploy the configuration based on user requirements, operator constraints, and technical constraints of the system. Their business-objectives-driven performance management uses utility function to optimize resource allocation and maximize the total utility. Compared to these efforts, our work focuses on the optimization of availability management to meet business objectives. Our solution involves the aspect of utility function, performance modeling, and availability modeling. In addition, availability management always involves uncertainty. In this work, we provide a method to deal with users' risk attitudes and handle different tradeoff between performance and reliability.

To improve availability, fault tolerance techniques are widely used in systems such as Fault-Tolerant CORBA [58, 63], and Arjuna [64]. These systems replicate selected application/service objects and provide specifications to allocate standbys for fast recovery. Multiple replicas allow an object to continue to provide service even when one of its replicas fails. Based on these widely used techniques, automated system design for availability is proposed in [43]. For availability management during changes, rolling upgrades are often used to minimize the down time during upgrades. A more systematic approach is proposed in Mirage [26]. Here, staged deployment (after clustering based upgrade behaviors), user-machine testing, and problem reporting cooperate in a structured manner to reduce overall failures.

5.3 Routing and Scheduling in Overlay Network

While the PGOS packet scheduling algorithm is inspired by the DWCS packet scheduling algorithm [100], its use for efficient multimedia data streaming across the Internet leverages substantial prior work on improving the quality of network video streaming [46, 72]. Here, early work established the utility of adding and dropping different encoding layers of video streams for longer term coarse-grain stream adaptation [98]. Improvements like those

in [72] use TCP-friendly control mechanisms to react to congestion on shorter timescales, with mismatches between the two timescales absorbed by buffering at the receiver. The control mechanisms used for multimedia data streaming are based on an adaptive layered video streaming algorithm for MPEG-4 with limited buffer size described in [46], where priorities are used in the VOP (video object plane) to select or discard each VOP element based on average bandwidth prediction, to control the fashion in which fine-grain scalable coding allocates bandwidth to different encoding layers. The contributions of IQ-Paths in this context are its use of statistical bandwidth measurement and prediction to capture network link qualities, and its PGOS self-regulating data routing and scheduling algorithm can utilize both multiple or alternate overlay paths to satisfy different video layers' utility requirements. The outcome is improved smoothness of video playback, despite the variable-bit-rate nature of layered video. The additional techniques described in [29] can be used to further smooth such variable-bit-rates, thereby attaining a constant transfer rate for each time interval in the transmission process.

OverQoS [87] describes the general idea of using overlays and admission control to deliver video across the Internet. In OverQoS, performance gains are achieved by FEC (Forward Error Correction) and conditional packet retransmission in the form of ARQs (Automatic Repeat reQuests). Bandwidth less than the total available bandwidth can be achieved for a subset of the OverQoS flows, with high probability, but potentially at the expense of other flows. In contrast, the PGOS algorithm controls path usage with a more general abstraction that is able to provide statistical guarantees for both single and multiple streams across both single and multiple paths across the overlay.

Both IQ-Paths and OverQoS assume that overlay routing nodes can be placed such that the paths between different pairs of routing nodes do not share common bottlenecks. In practice, such placements require knowledge of the network, by using methods of detecting shared congestion across flows [73], or by using more direct ways of detecting network topologies [83]. A general way to implement information exchanges between middleware

and networks is described in [59], with a design of a network underlay that extracts and aggregates topology information from the underlying Internet. Overlay networks query the underlay when making application-specific routing decisions. Further, the implementation of IQ-Paths could take advantage of underlays [59], which is a general framework that extracts and aggregates topology and other network information from the underlying Internet, and the results of recent work on a 'map of the Internet' described in [84], which annotates it with properties that include connectivity, geography, routing policy, patterns of loss, congestion, failure and growth, etc.

Our general approach of using overlay networks to adapt to network dynamics is shown feasible in [23], which compares the performance of an End System Multicast architecture to that of IP Multicast. We also note that noisy link measurements coupled with aggressive adaptation can cause overlay instability, while conservative adaptations may experience low performance. The proposed solution is to use exponential smoothing to capture the long term performance of a link, thereby distinguishing persistent from temporary changes. Our approach differs in that it exploits knowledge about noise rather than suppressing it, for example, by mapping critical data flows to less noisy links.

While our work complements research on process/job scheduling for Grid services [92]), Data Grid or similarly data-intensive applications in particular must harness both computational and network resources for their distributed operation [69]. Algorithms like PGOS can provide valuable assistance in these contexts, by integrating it, for instance, into the Dataset Scheduler framework described in [69]. To demonstrate the importance our methods, we have extended the popular GridFTP package [2] with the PGOS routing and scheduling algorithm, to better control how parallel data streams with different service-level objectives are scheduled across multiple network links.

The ability of overlay networks to provide differentiated data delivery services requires certain levels of independence in underlying network links' packet losses, changes in bandwidth, etc. For the Internet, [4] shows that there is a reasonable degree of loss and failure

independence across different links. Measurements on Planetlab [24] show that there are reasonable degrees of bandwidth independence for different Internet links. A basic contribution of the PGOS algorithm is its ability to predict future network behavior. [109] points out the difficulty of predicting bandwidth in wide area networks, studying the likelihood of observed bandwidth remaining in a region for which the ratio between the maximum and minimum observed values is less than a factor of ρ. We adopt a similar approach, assuming that it is difficult to predict the exact value of throughput in the next time interval (e.g., in the next second) and instead, providing statistical guarantees for predicting the distribution of throughput in the near future. Interestingly, as shown in [70], it is easier to make guarantees about RTT. Finally, we also leverage the substantial research on measuring available bandwidth described in [39, 56]. Of specific relevance to this work is recent research presenting more accurate metrics and algorithms to measure the variation of end-to-end available bandwidth [57].

CHAPTER VI

CONCLUSION AND FUTURE WORK

In this thesis, we first proposed a risk-based availability management approach to automating the availability of management in IT systems driven by business policy. We implement a policy engine that dynamically optimizes expected utility according to high level availability and performance objectives. We further study how to apply utility theory, such as von Neumann-Morgenstern utility functions, to deal with users' risk attitudes and preferences and thereby, enable users to customize their availability to the risks tolerable by business objectives. The evaluation of the proposed solution demonstrates that our approach is applicable for availability management of complex IT environments.

Future work includes the evaluation of our approach on large number of applications and how to deal with heterogeneous risk attitude. Many emerging applications have different performance/availability requirement and are typically co-located in one or multiple shared data centers. For example, web 2.0 provides an architecture of participation where users can contribute website content creates network effects, which often requires quick response time and high availability of the content. At the same time, many video sharing website have different sub-systems with some of them require good performance and high availability (e.g., streaming cluster must scale well and has high availability as a whole), some of them require mediocre performance and availability (e.g., video conversion servers) and some of them require high availability (e.g., content server). It's interesting to invest how the risk-based availability management approach would benefit data centers hosting these applications. The experiments done in this work don't consider changing risk policies, e.g., as dynamic functions of time t. It's expected that the same techniques can be applied to this type of risk policies with some enhancement, and we

hope to see what kind of benefits our approach can bring to this more complicated yet more interesting situation. How to elicit risk attitudes easily, especially for large number of risk policies would also be an interesting and useful research.

For the critical and complex enterprise applications, since a key contributor to application utility is the time taken to recover from failures, we develop a novel proactive fault tolerance approach. Techniques are presented to manage the tradeoff between availability and cost in information flow middleware. First, a net-utility-based formulation of the benefits an enterprise derives from its information flows combines both performance and reliability attributes of such flows. The goal is not simply to attain high utility, but to reliably provide high utility to large-scale information flow applications. Second, since reliability techniques incur costs, thereby reducing utility, proactive methods for availability-management regulate resources used to guarantee availability and take into account the fact that system and application behaviors change over time. A specific example is a higher likelihood of failure in high load vs. low load conditions. Reliability costs, therefore, are reduced by exploiting knowledge about the current 'perceived' system stability. Additional cost savings result from the use of failure prediction methods. Third, the implementation presented in this thesis can deal with both transient and non-transient failures, the latter relying on application-specific techniques for fault avoidance. Finally, utility-driven proactive availability management techniques have been integrated into our infrastructure for large-scale information flows, where it is shown to impose low additional communication and processing overheads on information flows. Experimental results with IFLOW attained on Emulab [93] demonstrate the effectiveness of proactive fault tolerance in recovering from failures.

Future work will experiment with richer failure prediction techniques and investigate specific enterprise environments. For instance, it's interesting to model the redundant datacenters mandated by government rules, and to consider the attainment of high availability and net-utility in information flows that cross multiple organizational boundaries.

IQ-Paths is a data streaming middleware that uses overlay networks to better serve the needs of distributed applications running on wide area networks. IQ-Paths employs statistical methods to provide to applications predictive guarantees for the bandwidths available to them from the underlying network, for all paths in the overlay connecting data sources to sinks. In addition, its PGOS scheduling algorithm both suitably routes packets across overlay paths and schedules them across single and multiple (concurrent) paths, coupling parallelism in data transfer with statistical bandwidth guarantees.

The statistical prediction technique used in IQ-Paths not only measures average available bandwidth, but also captures the dynamic or noisy nature of the bandwidth available on overlay paths. As a result, IQ-Paths can provide to applications both probabilistic and 'bounded violation' delivery guarantees. The former state that with some large probability P, stream S_i will receive the required bandwidth on the selected path. The latter state that the average number of messages that violate some constraint (e.g., miss their deadlines) during each scheduling window is bounded.

We have used IQ-Paths to meet the needs of several representative distributed applications including the SmartPointer real-time collaboration system, and GridFTP. The integration of IQ-Paths into these applications is facilitated by its design as a 'model-neutral' data streaming layer underlying the application-specific communication models offered by higher middleware layers, including the publish/subscribe model used by SmartPointer, the simple data transfer model used by GridFTP, and the data streaming model used by the multimedia application.

Risk-based traffic delivery guarantees across overlay network is utilized to link the operational guarantees expressed by utility and enforced by the PGOS algorithm with the higher level business objectives sought by end users.

Several extensions of the proposed IQ-Paths work are of future interest. The path characteristics collected by IQ-Paths can benefit a wide range of high performance, multimedia, and enterprise applications. For enterprise applications, our research is further enhancing

this work by developing runtime methods for fault tolerance. Here, we want to differentiate data traffic required for replication from other traffic, by dynamically varying reliability/performance tradeoffs with selective replication techniques. Another interesting topic is to use IQ-Paths to isolate the effects of fault tolerance or recovery traffic from regular data traffic, perhaps to avoid the additional disturbances arising during recovery. This is explored in the risk-based traffic delivery guarantees, however, it is interesting to see how this will affect service availability and performance in a real cross-country enterprise computing infrastructure. Also, the statistical bandwidth prediction could be further enhanced using typical seasonal behaviors of available bandwidth.

CHAPTER VII

APPENDIX

7.0.1 Proof of Lemma 1

Proof Let the service rate over path j at time t be $r^j(t)$. Then the service rate cumulative distribution $G^j(r^j)$ is:

$$\begin{aligned} G^j(r^j) &= P\{r \le r^j)\} = P\{rs \le r^j s\} \\ &= P\{b \le r^j s\} = F^j(r^j s). \end{aligned} \tag{16}$$

The probability that x_i packets will be served during the scheduling window t_w is

$$\begin{aligned} P &= P\{x_i \le r t_w\} = P\{x_i/t_w \le r\} \\ &= 1 - P\{r \le x_i/t_w\} \\ &= 1 - G^j(x_i/t_w) = 1 - F^j(x_i s/t_w). \end{aligned}$$

Note that this is essentially bounding the probability of throughput violations.

7.0.2 Proof of Lemma 2

Proof

$$Z = \begin{cases} x_i - r^j t_w & \text{if } x_i > r^j t_w \\ 0 & \text{if } x_i \le r^j t_w \end{cases} \tag{17}$$

Let f_B be the pdf of available bandwidth b, then we have

$$E[Z] = \int_{-\infty}^{+\infty} Z \cdot f_B(b) d(b)$$

$$
\begin{aligned}
&= \int_0^{x_i s/t_w} (x_i - b t_w/s) \cdot f_B(b) d(b) \\
&= x_i \cdot \int_{-\infty}^{b_0} f_B(b) d(b) - \frac{t_w}{s} \cdot \int_{-\infty}^{b_0} b f_B(b) d(b) \\
&= x_i \cdot F^j(b_0) - \frac{t_w}{s} \cdot M[b_0] \qquad (18)
\end{aligned}
$$

7.0.3 Proof of Theorem 1

Proof Let the service rate stream S_i receives on path j be r_i^j, and stream S_i will obtain service rate r_i^j with probability P_i^j, $\sum_{j=1}^{L} P_i^j = P_i$. X_i is the actual number of packets delivered for stream S_i during the scheduling window t_w, and among these packets, X_i^j packets are delivered through path j.

The probability that x_i packets will be served during the scheduling window for stream S_i is:

$$
\begin{aligned}
P &= P\{x_i \le X_i\} = P\{x_i^1 \le X_i^1, ..., x_i^L \le X_i^L\} \\
&= \sum_{j=1}^{L} P\{x_i^j/t_w \le X_i^j/t_w\} \\
&= \sum_{j=1}^{L} P\{x_i^j/t_w \le r_i^j\} = \sum_{j=1}^{L} P_i^j = P_i. \qquad (19)
\end{aligned}
$$

REFERENCES

[1] AIBER, S., GILAT, D., LANDAU, A., RAZINKOV, N., SELA, A., and WASSERKRUG, S., "Autonomic self-optimization according to business objectives," in *Proc. of ICAC*, 2004.

[2] ALLCOCK, B., BRESNAHAN, J., KETTIMUTHU, K., LINK, M., DUMITRESCU, C., RAICU, I., and FOSTER, I., "Zebra: the globus stripped gridFTP framework and server," in *Proc. of Super Computing*, 2005.

[3] AMZA, C., CECCHET, E., CHANDA, A., ELNIKETY, S., COX, A., GIL, R., MARGUERITE, J., RAJAMANI, K., and ZWAENEPOEL, W., "Bottleneck characterization of dynamic web site benchmarks," in *Rice University Technical Report TR02-388*, 2002.

[4] ANDERSEN, D. G., SNOEREN, A. C., and BALAKRISHNAN, H., "Best-Path vs. Multi-Path Overlay Routin," in *Proc. of IMC*, 2003.

[5] BALAZINSKA, M., BALAKRISHNAN, H., MADDEN, S., and STONEBRAKER, M., "Fault-tolerance in the borealis distributed stream processing system," in *Proc. of the ACM SIGMOD international conference on Management of data*, 2005.

[6] BARTOLINI, C. and SALLE, M., "Business driven prioritization of service incidents," in *Proc. of IFIP/IEEE DSOM*, 2004.

[7] BERNHOLDT, D., BHARATHI, S., BROWN, D., CHANCHIO, K., CHEN, M., CHERVENAK, A., CINQUINI, L., DRACH, B., FOSTER, I., FOX, P., GARCIA, J., KESSELMAN, C., MARKEL, R., MIDDLETON, D., NEFEDOVA, V., POUCHARD, L., SHOSHANI, A., SIM, A., STRAND, G., and WILLIAMS, D., "The earth system

grid II: turning climate datasets into community resources.," in *Annual Meeting of the American Meteorological Society*, 2002.

[8] BIRMAN, K. P., *Relible distributed systems - technologies, web services, and applications*. Springer, 2005.

[9] BLANQUER, J. M. and OZDEN, B., "Fair queuing for aggregated multiple links," in *Proc. of ACM SIGCOMM*, 2001.

[10] BLOUGH, D. and LIU, P., "FIMD-MPI: A tool for injecting faults into mpi applications," in *Proc. of IPDPS*, 2000.

[11] BRIN, S. and PAGE, L., "The anatomy of a large-scale hypertextual Web search engine," *Computer Networks and ISDN Systems*, vol. 30, no. 1–7, pp. 107–117, 1998.

[12] BUCO, M., CHANG, R., LUAN, L., WARD, C., WOLF, J., and YU, P., "Managing of eBusiness on demand SLA contracts in business terms using the cross-SLA execution manager SAM," in *Proc. of IEEE ISADS*, 2003.

[13] CAI, Z., KUMAR, V., COOPER, B. F., EISENHAUER, G., SCHWAN, K., and STROM, R. E., "Utility-driven proactive management of availability in enterprise-scale information flows," in *Proc. of ACM Middleware*, 2006.

[14] CAI, Z., CHEN, Y., KUMAR, V., MILOJICIC, D., and SCHWAN., K., "Automated availability management driven by business policies," in *Proc. of IEEE IM*, 2007.

[15] CAI, Z., EISENHAUER, G., HE, Q., KUMAR, V., SCHWAN, K., and WOLF, M., "IQ-Services: Network-aware middleware for interactive large-data applications," *Concurrency & Computation. Practice and Exprience Journal*, 2005.

[16] CAI, Z., KUMAR, V., COOPER, B. F., EISENHAUER, G., SCHWAN, K., and STROM, R., "Utility-driven availability-management in enterprise-scale information flows." http://www.cercs.gatech.edu/tech-reports/, Dec. 2006. Technical report.

[17] CAI, Z., KUMAR, V., and SCHWAN, K., "IQ-Paths: self-regulating data streams across network overlays," in *Proc. of IEEE HPDC*, 2006.

[18] CAI, Z., KUMAR, V., and SCHWAN, K., "IQ-Paths: self-regulating data streams across network overlays," tech. rep., Georgia Tech, 2006.

[19] CAI, Z., KUMAR, V., and SCHWAN, K., "IQ-Paths: self-regulating data streams across network overlays," *Journal of Grid Computing*, vol. 5, no. 2, pp. 129–150, 2007.

[20] CANDEA, G., KAWAMOTO, S., FUJIKI, Y., FRIEDMAN, G., and FOX, A., "Microreboot - a technique for cheap recovery," in *Proc. of OSDI*, 2004.

[21] CHERITON, D. and SKEEN, D., "Understanding the limitations of causally and totally ordered communication," in *Proc. of SOSP*, 1993.

[22] CHESS, D. M., KUMAR, V., SEGAL, A., and WHALLEY, I., "Availability-aware self-configuration in autonomic systems," in *Proc. of IFIP/IEEE DSOM*, 2003.

[23] CHU, Y., RAO, S., SESHAN, S., and ZHANG, H., "Enabling conferencing applications on the internet using an overlay muilticast architecture," in *Proc. of SIGCOMM*, 2001.

[24] CHUN, B., CULLER, D., ROSCOE, T., BAVIER, A., PETERSON, L., WAWRZONIAK, M., and BOWMANT, M., "Planetlab: An overlay testbed for broad-coverage services," *ACM Computer Communication Review*, vol. 33, July 2003.

[25] COHEN, I., GOLDSZMIDT, M., KELLY, T., SYMONS, J., and CHASE, J., "Correlating instrumentation data to system states: A building block for automated diagnosis and control," in *Proc. of OSDI*, 2004.

[26] CRAMERI, O., KNEZEVIC, N., KOSTIC, D., BIANCHINI, R., and ZWAENEPOEL, W., "Staged deployment in mirage, an integrated software upgrade testing and distribution system," in *Proc. of SOSP*, Oct. 2007.

[27] DODGE, D., "Google data centers vs microsoft infrastructure - a battle of the titans." `http://dondodge.typepad.com/the_next_big_thing/2006/06/google_data_cen.html,Dec.2007`.

[28] DOE, "UltraScience Net." `http://www.csm.ornl.gov/ultranet/,Dec.2007`.

[29] FENG, W.-C. and REXFORD, J., "Performance evaluation of smoothing algorithms for transmitting prerecorded variable-bit-rate video," *IEEE Trans. on Multimedia*, Sept 1999.

[30] FOX, A., KICIMAN, E., and PATTERSON, D., "Combining statistical monitoring and predictable recovery for self-management," in *Proc. of SIGSOFT workshop on Self-managed systems*, 2004.

[31] FREY, J., TANNENBAUM, T., LIVNY, M., FOSTER, I., and TUECKE, S., "Condor-g: a computation management agent for multi-institutional grids," in *Proc. of IEEE HPDC*, 2001.

[32] FRIEDMAN, M., TRAN, P., and GODDARD, P., *Reliability of software intensive systems*. William Andrew Inc, 1995.

[33] FRIESE, T., MULLER, J., and FREISLEBEN, B., "Self-healing execution of business processes based on a peer-to-peer service architecture," in *Proc. of ICAC*, 2005.

[34] GAMS DEVELOPMENT CORPORATION, "General algebraic modeling system." http://www.gams.com/,Dec.2007.

[35] GAVRILOVSKA, A., SCHWAN, K., and OLESON, V., "A practical approach for zero downtime in an operational information system," in *Proc. of IEEE ICDCS*, 2002.

[36] GRAUPNER, S., PRUYNE, J., and SINGHAL, S., "Making the utility data center a power station for the enterprise grid," in *HP Labs. Technical Report HPL-2003-53*, 2003.

[37] GRAY, J., MCJONES, P. R., BLASGEN, M. W., LINDSAY, B. G., LORIE, R. A., PRICE, T. G., PUTZOLU, G. R., and TRAIGER, I. L., "The recovery manager of the system R database manager," *ACM Computing Surveys*, vol. 13, no. 2, 1981.

[38] GROSS, K. C. and LU, W., "Early detection of signal and process anomalies in enterprise computing systems," in *Proc. of IEEE International Conference on Machine Learning and Applications*, 2002.

[39] HU, N. and STEENKISTE, P., "Evaluation and characterization of available bandwidth probing techniques," *IEEE Journal on Selected Areas in Communications*, Aug. 2003.

[40] IBM, "IBM global services: Improving systems availability." http://www.cs.cmu.edu/~priya/hawht.pdf,Dec.2007.

[41] INFOBAHN INC., "T1, T3, OC3, OC12 and OC48 Internet service provider price quotes." http://www.infobahn.com/research-information.htm, Dec.2007.

[42] ISO/IEC 14496-2/FPDAM4, "Coding of audio-visual objects, part-2 visual, amedment 4: Streaming video profile," Jul. 2000.

[43] JANAKIRAMAN, G., SANTOS, J. R., and TURNER, Y., "Automated system design for availability," in *Proc. of IEEE/IFIP DSN*, 2004.

[44] KEPHART, J. and WHITE, S., "A research agenda for business-driven information technology," in *Proc. of the First Workshop on Hot Topics in Autonomic Computing*, 2006.

[45] KEPHART, J. O. and CHESS, D. M., "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[46] KIM, T. and AMMAR, M., "Optimal quality adaptation for MPEG-4 fine-grained scalable video," in *Proc. of IEEE INFOCOM*, Apr. 2003.

[47] KOPENA, J., SULTANIK, E., NAIK, G., HOWLEY, I., PEYSAKHOV, M., CICIRELLO, V. A., KAM, M., and REGLI, W., "Service-based computing on manets: Enabling dynamic interoperability of first responders," in *Proc. of IEEE Intelligent Systems*, 2005.

[48] KUMAR, V., COOPER, B., CAI, Z., EISENHAUER, G., and SCHWAN, K., "Middleware for enterprise scale data stream management using utility-driven self-adaptive information flows," *Cluster Computing Journal*, 2006.

[49] KUMAR, V., CAI, Z., COOPER, B. F., EISENHAUER, G., SCHWAN, K., MANSOUR, M., SESHASAYEE, B., and WIDENER, P., "Implementing diverse messaging models with self-managing properties using IFLOW," in *Proc. of ICAC*, 2006.

[50] KUMAR, V., COOPER, B. F., CAI, Z., EISENHAUER, G., and SCHWAN., K., "Resource-aware distributed stream management using dynamic overlays," in *Proc. of ICDCS*, 2005.

[51] KUMAR, V., COOPER, B. F., EISENHAUER, G., and SCHWAN, K., "iManage: Policy-driven self-management for enterprise-scale systems," in *Proc. of ACM Middleware*, 2007.

[52] LI, B. and NAHRSTEDT, K., "A control-based middleware framework for quality of service adaptations," *IEEE Journal on Selected Areas in Communications*, Sept. 1999.

[53] LUCAS, H. C., OH, W., SIMON, G., and WEBER, B. W., "Information technology and the new york stock exchange's strategic resources from 1982-1999," tech. rep., Computer Information Systems, Baruch Zicklin School of Business, The City University of New York, 2002.

[54] MAIR, G. M., "Telepresence - the technology and its economic and social implications," in *Proc. of IEEE International Symposium on Technology and Society*, 1997.

[55] MANSOUR, M. and SCHWAN, K., "I-RMI: Performance isolation in information flow applications," in *Proc. of ACM/IFIP/IEEE Middleware*, 2005.

[56] M.JAIN and DOVROLIS, C., "End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput," in *Proc. of SIGCOMM*, Aug. 2002.

[57] M.JAIN and DOVROLIS, C., "End-to-end estimation of the available bandwidth variation range," in *Proc. of SIGMETRICS*, June 2005.

[58] MOORSEL, A. and YAJNIK, S., "Design of a resource manager for fault-tolerant corba," in *Proc. of the Workshop on Reliable Middleware*, 1999.

[59] NAKAO, A., PETERSON, L., and BAVIERR, A., "A routing underlay for overlay networks," in *Proc. of ACM SIGCOMM*, 2003.

[60] NASA, "Using XML and java for telescope and instrumentation control," in *Proc. of SPIE Advanced Telescope and Instrumentation Control Software*, 2000.

[61] NATIONAL LABORATORY FOR APPLIED NETWORK RESEARCH, "Internet measurement and Internet analysis." `http://moat.nlanr.net/,Dec.2007`.

[62] NATIONAL LAMBDARAIL. `http://www.nlr.net/,Dec.2007`.

[63] OBJECT MANAGEMENT GROUP, "Final adopted specification for fault tolerant corba," in *OMG Technical Committee Document ptc/00-04-04*, 2000.

[64] PARRINGTON, G., SHRIVASTAVA, S., WHEATER, S., and LITTLE, M., "The design and implementation of Arjuna," in *Proc. of USENIX Computing Systems*, 1995.

[65] PERTET, S. M. and NARASIMHAN, P., "Causes of failure in web applications," in *PDL Technical Report PDL-CMU-05-109, Carnegie Mellon University*, 2005.

[66] PINHEIRO, E., WEBER, W.-D., and BARROSO, L. A., "Failure trends in a large disk drive population," in *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST07)*, 2007.

[67] RABINOVICH, M. and LAZOWSKA, E., "An efficient and highly available read-one write-all protocol for replicated data management," in *Proc. of the Int'l Conf. on Parallel and Distributed Information Systems*, 1993.

[68] RANDELL, B., LEE, P., and TRELEAVEN, P. C., "Reliability issues in computing system design," *ACM Computing Surveys*, vol. 10, no. 2, 1978.

[69] RANGANATHAN, K. and FOSTER, I., "Decoupling computation and data scheduling in distributed data-intensive applications," in *Proc. of Super Computing*, 2002.

[70] RAO, N. S. V., "Overlay networks of in-situ instruments for probabilistic guarantees on message delays in wide-Area networks," *IEEE Journal on Selected Areas of Communications*, vol. 22, no. 1, 2004.

[71] RAZINKOV, SELA, A., and WASSERKRUG, S., "Autonomic self-optimization according to business objectives," in *Proc. of IEEE ICAC*, 2004.

[72] REJAIE, R., HANDLEY, M., and ESTRIN, D., "Quality adaptation for congestion controlled playback video over the Internet," in *Proc. of ACM SIGCOMM*, 1999.

[73] RUBENSTEIN, D., KUROSE, J., and TOWSLEY, D., "Detecting shared congestion of flows via end-to-end measurement," *IEEE/ACM Transactions on Networking*, vol. 10, June 2002.

[74] RUSSELLO, G., CHAUDRON, M., and VAN STEEN., M., "Dynamically adapting tuple replication for high availability in a shared data space," in *Proc. Int'l Conf. on Coordination Models and Languages*, 2005.

[75] SAHAI, A., SINGHAL, S., MACHIRAJU, V., and JOSHI, R., "Automated generation of resource configurations through policies," in *Proc. of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04)*, 2004.

[76] SALLE, M. and BARTOLINI, C., "Management by contract," in *Proc. of IEEE/IFIP NOMS*, 2004.

[77] SALLE, M., SAHAI, A., BARTOLINI, C., and SINGHAL, S., "A business-driven approach to closed-loop management," in *HP Labs Technical Reports, HPL-2004-205*, 2004.

[78] SCHINTKE, F. and REINEFELD., A., "Modeling replica availability in large data grids," *Journal of Grid Computing*, 2003.

[79] SCHROEDER, B. and GIBSON, G. A., "A large-scale study of failures in high-performance computing systems," in *Proc. of DSN*, 2006.

[80] SCHROEDER, B., WIERMAN, A., and HARCHOL-BALTER, M., "Open versus closed: A cautionary tale," in *Proc. of USENIX NSDI*, 2006.

[81] SCHWAN, K., COOPER, B. F., EISENHAUER, G., GAVRILOVSKA, A., WOLF, M., ABBASI, H., AGARWALA, S., CAI, Z., KUMAR, V., LOFSTEAD, J., MANSOUR, M., SESHASAYEE, B., and WIDENER, P., *AutoFlow: autonomic information flows for critical information systems. Autonomic computing: concepts, infrastructure, and applications*. CRC Press, 2006.

[82] SENS, P. and FOLLIOT, B., "STAR: a fault-tolerant system for distributed applications," *Software - Practice and Experience*, vol. 28, no. 10, 1998.

[83] SPRING, N., MAHAJAN, R., and WETHERALL, D., "Measuring ISP topologies with rocketfuel," in *Proc. of ACM SIGCOMM*, 2002.

[84] SPRING, N., WETHERALL, D., and ANDERSON, T., "Reverse-engineering the Internet," in *Proc. of HotNets-II*, 2003.

[85] STELLING, P., FOSTER, I., KESSELMAN, C., LEE, C., and LASZEWSKI, G. V., "A fault detection service for wide area distributed computations," in *Proc. of HPDC*, 1998.

[86] STROM, R. E., "Fault-tolerance in the smile stateful publish-subscribe system," in *Proc. of the Int'l Workshop on Distributed Event-Based Systems*, 2004.

[87] SUBRAMANIAN, L., STOICA, I., BALAKRISHNAN, H., and KATZ, R., "OverQoS: an overlay based architecture for enhancing internet QoS," in *Proc. of 1st Symposium on Networked Systems Design and Implementation(NSDI)*, Mar. 2004.

[88] TAESOMBUT, N., WU, X., and CHIEN, A. A., "Collaborative data visualization for earth sciences with the OptIPuter," 2005.

[89] TAI, A. T., MEYER, J. F., and AVIZIENIS, A., *Software performability: from concepts to applications*. Boston: Kluwer Academic Publishers, 2005.

[90] TALWAR, V., MILOJICIC, D., WU, Q., PU, C., YAN, W., and JUNG, G., "Approaches to service deployment," *IEEE Internet Computing*, vol. 9, no. 2, 2005.

[91] TALWAR, V., WU, Q., PU, C., YAN, W., JUNG, G., and MILOJICIC, D., "Comparison of approaches to service deployment," in *Proc. of IEEE ICDCS*, 2005.

[92] TANNENBAUM, T., WRIGHT, D., MILLER, K., and LIVNY, M., "Condor – a distributed job scheduler," in *Beowulf Cluster Computing with Linux*, MIT Press, 2001.

[93] THE FLUX RESEARCH GROUP, "The Utah network testbed." `http://www.emulab.net/,Dec.2007`.

[94] TUCEK, J., LU, S., HUANG, C., XANTHOS, S., and ZHOU, Y., "Triage: Diagnosing production run failures at the users site," in *Proc. of SOSP*, Oct. 2007.

[95] VENGEROV, D. and IAKOVLEV, N., "A reinforcement learning framework for dynamic resource allocation: First results," in *Proc. of the IEEE ICAC*, 2005.

[96] VINT PROJECT, "The network simulator - ns-2." `http://www.isi.edu/nsnam/ns/,Dec.2007`.

[97] VON NEUMANN, J. and MORGENSTERN, O., *Theory of games and economic behavior*. Princeton University Press, 1944.

[98] WALPOLE, J., KOSTER, R., CEN, S., COWAN, C., MAIER, D., MCNAMEE, D., PU, C., STEERE, D., and YU, L., "A player for adaptive MPEG video streaming over the Internet," in *Proc. of SPIE Applied Imagery Pattern Recognition Workshop*, (Washington, DC), Oct. 1997.

[99] WALSH, W. E., TESAURO, G., KEPHART, J. O., and DAS, R., "Utility functions in autonomic systems," in *Proc. of ICAC*, 2004.

[100] WEST, R. and POELLABAUER, C., "Analysis of a window-constrained scheduler for real-time and best-effort packet streams," in *Proc. of IEEE Real-Time Systems Symposium*, 2000.

[101] WISEMAN, Y. and SCHWAN, K., "Efficient end to end data exchange using configurable compression," in *Proc. of ICDCS*, Mar. 2004.

[102] WOLF, M., CAI, Z., HUANG, W., and SCHWAN, K., "Smart pointers: personalized scientific data portals in your hand," in *Proc. of IEEE/ACM Supercomputing Conference*, Nov. 2002.

[103] WU, H. and KEMME., B., "Fault-tolerance for stateful application servers in the presence of advanced transactions patterns," in *Proc. of SRDS*, 2005.

[104] WUFTPD DEVELOPMENT GROUP, "WU-FTP." `http://www.wu-ftpd.org,Dec.2007`.

[105] YU, H. and VAHDAT, A., "The costs and limits of availability for replicated services," in *Proc. of SOSP*, 2001.

[106] ZAVALJEVSKI, N. and GROSS, K. C., "Uncertainty analysis for multivariate state estimation in mission-critical and safety-critical applications," in *Proc. MARCON*, 2000.

[107] ZEGURA, E. W., CALVERT, K., and BHATTACHARJEE, S., "How to model an internetwork," in *Proc. of IEEE INFOCOM*, March 1996.

[108] ZHANG, W. and ZHANG, W., "Linux virtual server clusters: build highly-scalable and highly-available network services at low cost," in *LinuxMagazine*, Nov. 2003.

[109] ZHANG, Y., DUFFIELD, N., PAXSON, V., and SHENKER, S., "On the constancy of internet path properties," in *Proc. of the ACM SIGCOMM Internet Measurement Workshop*, Nov. 2001.

VITA

Zhongtang Cai was born in Shanghai, China on April 6th, 1979. He received a B.S. degree in Computer Science under the supervision of Professor Yunhe Pan from Zhejiang University China in 2001. After his graduation from Zhejiang University, he began his graduate studies under the supervision of Prof. Karsten Schwan, in the College of Computing at the Georgia Institute of Technology. As a member of the Systems Research Group of the College of Computing and a member of the Georgia Tech Center for Experimental Research in Computer Systems (CERCS), he conducted research on various aspects of large-scale distributed systems, including fault-tolerant enterprise distributed systems, autonomic computing (e.g., automated system management, and online performance / availability monitoring and tuning etc), information dissemination middlewares, event delivery with Quality of Service and overlay message scheduling and routing. His research in these areas has resulted in a number of publications that have appeared in various journals and international conferences. In December 2007, under the supervision of Professor Karsten Schwan, Zhongtang Cai completed his Ph.D. dissertation entitled "Risk-Based Proactive Availability Management - Attaining High Performance and Resilience with Dynamic Self-Management in Enterprise Distributed Systems".