IMPROVING DATA CONSISTENCY MANAGEMENT AND OVERLAY MULTICAST IN INTERNET-SCALE DISTRIBUTED SYSTEMS

by

Yijun Lu

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Doctor of Philosophy

Major: Computer Science

Under the Supervision of Professor Hong Jiang and Professor Ying Lu

Lincoln, Nebraska

July, 2007

To my wife, Jianqin, for her love.

IMPROVING DATA CONSISTENCY MANAGEMENT AND OVERLAY MULTICAST IN INTERNET-SCALE DISTRIBUTED SYSTEMS

Yijun Lu, Ph.D.

University of Nebraska, 2007

Advisors: Hong Jiang and Ying Lu

In the past decade, the Internet has become the dominant platform for doing research and business. Compared to other platforms, the Internet promises to manage shared data efficiently and deliver the data to the consumer/user in real-time and in a transparent way. A wide variety of applications, including online collaboration and e-business, have already been implemented on the Internet.

However, providing services on the Internet poses two challenges. First, an Internet-based collaboration often has to replicate the shared data in many geographically distant locations to improve efficiency and availability. This replication makes the shared data difficult to manage. Second, the Internet is usually loosely connected due to its autonomous and decentralized nature and thus the transmission is inherently unreliable. Even when a message gets delivered ultimately, it is not uncommon for the message to be delayed for a significant amount of time. These challenges significantly hinder effective data management and efficient data delivery. In this dissertation research, we focus on improving data consistency management, from the data management perspective, and overlay multicast, from the data delivery perspective, in Internet-scale distributed systems.

The current data consistency management mechanisms often enforce a predefined consistency level, which cannot cater to the needs of multiple applications with different consistency requirements and incur high communication cost when there are

a large number of participants. To improve data consistency management, we first propose IDF (Inconsistency Detection Framework) that provides a unified framework to maintain adaptive consistency control for online distributed collaborations; second, we propose CVRetrieval (Consistency View Retrieval), a new system that greatly improves the scalability of consistency control by reducing the communication cost. These two systems tackle the consistency control problem from two angles: IDF provides adaptive consistency control and CVRetrieval improves the scalability of consistency control. Together, they help scale a variety of distributed online collaboration applications to the Internet level.

To encourage participation of online collaboration and e-business, a main focus of current research on overlay multicast is to improve the fairness among all participants' contributions. However, current mechanisms to enforce fairness only considers one multicast session. In this dissertation, we propose FairOM (Fair Overlay Multicast) that provides a new definition to fairness in overlay multicast to support multiple simultaneous multicast sessions. FairOM hence enables scalable, fair, and high performance overlay multicast of shared data in an Internet-scale distributed system.

IDF, CVRetrieval, and FairOM are fully evaluated through analysis, simulation, and deployment of prototypes on Planet-Lab, a widely used test-bed for distributed Internet services that is constructed on top of the Internet. The results show that (1) IDF achieves an adaptive and efficient consistency maintenance that can benefit a wide variety of applications; (2) CVRetrieval greatly improves consistency control by removing passive participants from the expensive consistency maintenance protocol and instead satisfies their needs on demand; and (3) FairOM can support more simultaneous multicast sessions than other protocols without sacrificing significant multicast performance and incurring unacceptable communication overhead.

ACKNOWLEDGEMENTS

First, I want to give my heartfelt thanks to Dr. Hong Jiang, my advisor. Since I came to Lincoln five year ago, Dr. Jiang has been giving me valuable guidance to conducting research. With his guidance and patience, I have greatly improved my research in many ways, including the writing skills and the right scientific method to do research. Dr. Jiang is very accessible. No matter he is at UNL or traveling abroad, he is always available to answer my research questions and give me valuable comments. Without his active participation and guidance, this dissertation research would not have been possible. Dr. Jiang is also a very warm person. Several times each year, he invites his students, including me, to his house for holidays or special events, which makes me feel like a member of a big family.

I would like to thank Dr. Ying Lu, my co-advisor, for her great efforts of improving my research and communication skills. In the past two years, she has put significant efforts into helping me improve the presentation of the IDEA research, the forth chapter of this dissertation. As well, her knowledge and insight of publish-subscribe schemes has helped shape the CVRetrieval project, the fifth chapter of this dissertation. She has stressed to me the importance of presenting my work clearly and correctly to others to further my research and helped make that happen.

I am grateful to Dr. Byrav Ramamurthy for his valuable comments on deepening my research through concrete evaluations and on improving my communication skills during both the dissertation proposal and final oral defense. Dr. Ramamurthy has also helped me with my experimentation on the Planet-Lab. As the coordinator of the Planet-Lab at UNL, he generously provided me with the resources needed to conduct experiments on the Planet-Lab. Without his help, the evaluation results provided in this dissertation would have been difficult, if not impossible, to obtain.

I would like to thank Dr. Leen-Kiat Soh for carefully reviewing my dissertation draft and offering his insightful comments. During the review process, he has exchanged with me a dozen insightful email messages to offer his valuable input on how to improve the dissertation. For example, he suggested that I do a complexity comparison for CVRetrieval, Deno, and IDEA to show the advantages of CVRetrieval more convincingly. Dr. Soh has also edited my dissertation draft carefully to improve the presentation.

I would like to thank Dr. Xun-Hong Chen for reviewing my dissertation draft and commenting on my dissertation research. From my comprehensive exam to dissertation proposal to final oral defense, Dr. Chen has offered his valuable insight without reservation. He is also very flexible about the time and location arrangement of my defense, which makes it easier for me to get things done in time.

I would like to thank Dr. Xueming Li from Chongqing University, China, who has stayed at Lincoln for one year from November 2005 to November 2006 and conducted collaborated research with me during his stay. Xueming is both a wonderful researcher with clear theoretical thinking and a nice person to work with. He has significantly helped me improve the analytical work of both the inconsistency detection effectiveness of IDF and the comparison between FairOM and non-FairOM approaches.

I would like to thank fellow members and alumnus of our research group, including Dr. Xiao Qin, Dr. Yifeng Zhu, Dr. Xuli Liu, and Mr. Feng Xian, for their valuable input about my research. I also like to thank my fellow students who came to UNL at the same time with me, including Dong Li, Hui Cheng, Jun Gao, and Yaling Zheng. We have had a wonderful experience together at Lincoln and have helped one another in many ways. I also want to express my thanks to all the Lincoln friends who have helped my wife and me in the past five years.

I am grateful to Yong Wang and Anwar Mamat who have provided valuable help during my stay in Lincoln when I did my dissertation proposal and the final oral defense.

I am indebted to Prof. Zongfen Han and Prof. Hai Jin from Huazhong University of Science and Technology, China, who have helped me build a solid foundation of my research while I was pursuing a master degree there.

I am grateful to Ed and Wanda Kelley. During the past five years, they have helped my wife and me in many ways. As a host family for my wife and me right after we arrived at Lincoln, they have helped us improve our spoken English, purchase our first car, and understand the American culture. Their kindness and friendship have made us feel at home even when we were tens of thousands of miles away from home.

I would like to thank my parents. They have always believed in me and loved me.

Last but certainly not the least, I would like to thank my wife, Jianqin to whom this dissertation is dedicated, for her love, her confidence in me and her continuous support of my PhD study, and for giving birth to our beautiful son, Wyatt. Without her firm confidence in me and continuous support, I would not have overcome the numerous difficulties I have faced during my PhD study to get the dissertation done.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the past decade, the Internet has become the dominant platform for doing research and business [24, 80, 97, 98, 100] because, as a unified global network, it connects geographically dispersed users together and promises to manage shared data efficiently and deliver the data to the users in a real-time and transparent way. With the Internet and appropriate enabling software, a research group can conduct an effective collaboration no matter where the group members are located; a business can connect its employees, clients, and business partners to collaborate or to do business. This ability, as put by Ian Foster, is the "fuel for innovation engine" because the workers do not "have to wait for resources or have to adapt their work processes to the peculiarities of available resources" [29]. Hence, a wide variety of applications–such as the Grid [30], online collaboration [17, 31], and e-business [79]–have already been moved or are being moved to run on the Internet.

However, providing these services on the Internet poses two challenges. First, an Internet-based collaboration often has to replicate the shared data in many geographically distant locations to improve efficiency and availability. This replication makes the shared data difficult to manage [64], particularly in terms of data con-

sistency management. Second, the Internet is usually loosely connected due to its autonomous and decentralized nature and thus the transmission is inherently unreliable. Even when a message gets delivered eventually, it is not uncommon for the message to be delayed for seconds or more [40]. These challenges make effective data management and efficient data delivery more difficult.

The objective of this dissertation is to improve the data consistency management aspect of data management and the overlay multicast aspect of data delivery in an Internet-scale distributed system.

In the rest of this chapter, we first clarify the concept of our targeted environment– an Internet-scale distributed system. Then we discuss the open research problems in this new environment, the scope of this dissertation research, and a scenario in which our proposed solutions work together to facilitate Internet-scale collaborations. Then, we summarize the contributions of this dissertation. The organization of this dissertation is outlined in the end of this chapter.

1.1 Problem Statement

In this section, we first define what an Internet-scale distributed system is. Then we elaborate on some relevant open research problems. Limitations of existing approaches are discussed thereafter.

1.1.1 Internet-Scale Distributed Systems

This dissertation targets Internet-scale distributed systems. Figure 1.1 illustrates such a system. The keywords here are "Internet-scale" and "distributed". By "Internet-scale", we mean that the system has a large number of participants and those participants are geographically dispersed across the Internet. According to this definition,

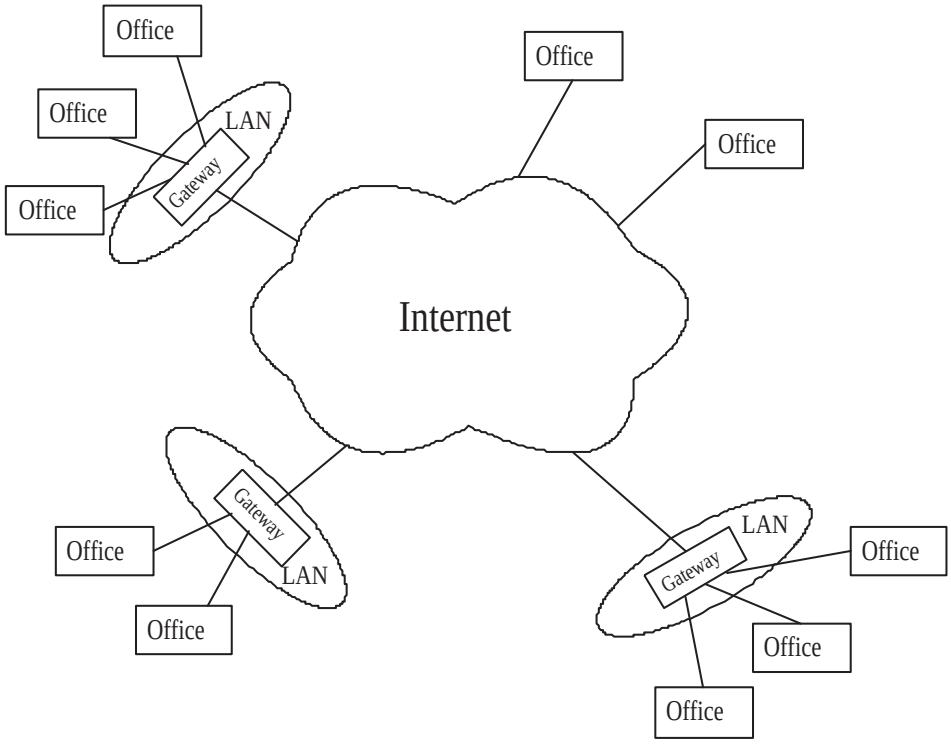


Figure 1.1: An example of Internet-scale distributed systems

it is necessary that the participants be geographically dispersed. If a large number of participants (such as thousands) work together but all of them reside in the same building, we do not consider it as Internet-scale.

By "distributed", we mean that the participants of the system are distributed across different sites and there is no central point of control. In this scenario, each site or participant manages its own data. When data need to be shared, a distributed protocol is used to maintain their integrity. This avoids single point of failure and prevents any central control mechanism from slowing down the whole system.

1.1.2 Open Research Problems

Internet-scale distributed systems have attracted a great deal of attention in the research and development community and there have been a number of research

projects that tackle various issues from different aspects [7, 19, 28, 32, 76, 92, 94, 96]. In this dissertation, we address the issues of data management and data delivery for shared data, particularly focusing on the data consistency management aspect and the overlay multicast aspect, respectively. Effective solutions to data management and data delivery are essential to improve collaboration among participants and help businesses readily and efficiently reach and serve their clients, thus enabling a wide variety of applications [29].

Data consistency directly determines data integrity: if the data is not sufficiently consistent among all the sites to make the collaboration meaningful, all the work based on such data will be useless [107]. In multiple-player online gaming, for example, inconsistencies among players could "lead to undesirable and sometimes paradoxical outcomes" [11]. The current consistency control mechanisms often enforce predefined consistency levels and cannot effectively support multiple applications with different consistency requirements. In addition, current mechanisms incur high communication overhead, which makes it it difficult for applications to scale to the Internet level.

Equally important to the data management problem, especially the consistency management, is the data delivery issue, *i.e.*, to deliver the well managed data to the places where they are needed. A promising strategy to achieve this goal is overlay multicast. Here, overlay multicast means that the multicast is done at the application level (often using TCP or UDP), not at the IP level as is the case for the IP-level multicast. Overlay multicast has several advantages, of which the most important one is that it does not need any router-level configuration and thus can be easily deployed. To encourage participation, a main focus of current research on overlay multicast is to improve the fairness among all nodes' contributions. However, current mechanisms to enforce fairness are designed from the point of view of one multicast session and cannot effectively support multiple simultaneous multicast sessions.

This dissertation addresses these research problems by proposing IDF (Inconsistency Detection Framework) and CVRetrieval (Consistent View Retrieval) to improve consistency control, and FairOM (Fair Overlay Multicast) to improve overlay multicast.

1.1.3 Limitations of Existing Approaches

Originated from a tightly coupled environment, most of the existing data consistency management mechanisms often enforce a predefined consistency level to a system, whereby all applications are subject to the same level of consistency control at all times. This strategy, while relatively simple, has several disadvantages.

First, it cannot cater to the needs of multiple applications running on the same system that have different consistency requirements. Even with a single application, this strategy can be inadequate when this application's consistency requirement changes from time to time. We propose IDF to provide an adaptive consistency control protocol to solve this problem.

Second, current consistency control protocols are rooted at those used in small-scale computer systems and tend to incur high communication overhead as the number of participants increases. Since participants in Internet-scale distributed systems are distributed world-wide and they often have limited network bandwidth to communicate among them, the high communication cost is not acceptable. Inspired by the observation that not all participants are equally engaged in a collaboration, we propose CVRetrieval to separate passive participants from active participants and use different consistency control strategies to satisfy the needs of passive participants, thus saving significant communication cost.

In the context of overlay multicast, an important research topic is to enforce fairness. Because participants in an Internet-scale distributed system are not necessarily

affiliated to the same organization, they may not be willing to contribute significantly more resources than others when everyone gets the same service (the same multicast content). Thus, to encourage participation, it is important to enforce fairness. However, current mechanisms to enforce fairness only consider one multicast session and cannot effectively support multiple sessions (*i.e.*, multiple sessions cannot be enabled). FairOM, our proposed scheme, enforces proportional contributions and is capable of enabling multiple simultaneous multicast sessions.

1.2 Dissertation Scope

This dissertation has three main parts: IDF, CVRetrieval, and FairOM. The first two parts of this dissertation, IDF and CVRetrieval, improve consistency control in Internet-scale distributed systems and their basic ideas are discussed as follows.

IDF improves the adaptability of consistency control in Internet-scale distributed systems. Instead of applying one or several predefined consistency protocols before a system is started, IDF detects inconsistencies as they occur. Then, through predefined criteria or real-time interactions with users, IDF efficiently resolves the detected inconsistencies when the consistency level falls below the desired level. IDF is suitable for a variety of online distributed collaboration applications.

While IDF achieves adaptability of consistency control, CVRetrieval further improves the scalability of consistency control by separating passive participants from active participants. In CVRetrieval, the consistency among active participants is maintained by a consistency maintenance protocol, such as IDEA–the one supported by IDF, and the passive participants that are interested in the shared object/service will be kept informed in an on-demand fashion via a publish-subscribe infrastructure.

The third part of this dissertation, FairOM, redefines the fairness in the context

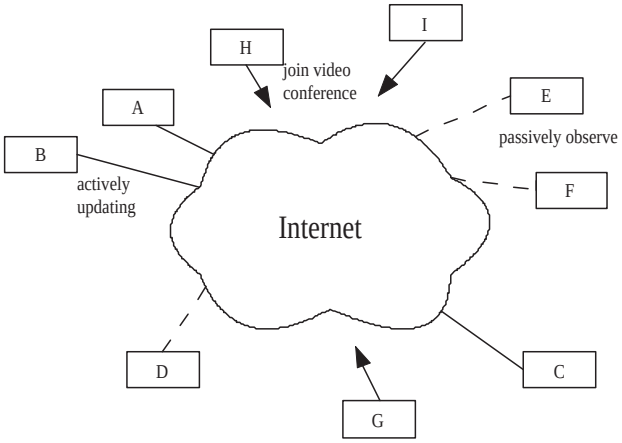


Figure 1.2: A scenario in which IDEA, CVRetrieval, and FairOM work together to facilitate collaboration. Solid line: active updating; dashed line: passive observe; line with arrow: only join for video conference.

of overlay multicast: each participant contributes the same proportion of his or her available outgoing bandwidth for each multicast session. With this definition, the enforced fairness translates to high performance because it is possible to support multiple simultaneous multicast sessions. FairOM is such a protocol to support this new definition and can support a large number of simultaneous multicast sessions.

1.3 A Scenario

We now illustrate a scenario, in which the three parts of this dissertation–IDF, CVRetrieval, and FairOM–work together seamlessly to support data sharing. In this scenario, we assume a global company named *Smart Inc.* that has employees all over the world and these employees need to collaborate with one another from time to time.

As shown in Figure 1.2, globally distributed users *A*, *B*, ..., and *F* from *Smart Inc.* are collaborating on one project file. In this case, a major overhead of this collaboration is the consistency control because, unlike the computational cost that is insignificant for modern computers, consistency control must take place across the

globe in a timely fashion and thus consumes precious network bandwidth. If not managed well, it can quickly consume the bandwidth among participants. In light of this, CVRetrieval greatly improves the scalability of consistency control (by reducing the associated bandwidth overhead) with the separation of passive observers from active collaborators (writers). For example, at a given time, only A, B, and C are actively working on this file (denoted by solid line connection), while others (D, E, and F) are just reviewing it, as denoted by the dash line connection. With this separation, CVRetrieval uses IDF to control consistency of the three active writers and to keep others informed about the most recent version of the shared file via its publish-subscribe infrastructure to save precious network bandwidth.

For active writers A, B, and C, IDF hides the details of consistency control from them so that they can focus on their main task: collaborating on the project. By deploying a detection-based consistency control protocol IDEA, the users can interact with IDEA to set their acceptable consistency level and IDEA will: (1) further save bandwidth of consistency control by delaying consistency resolution until it is needed; and (2) learn not to bother users by automatically resolving inconsistencies and to maintain the consistency above a desirable level. This adaptive interface lets users focus on their core task and thus further improves their productivity.

Finally, let's assume that the active collaborators (writers) have successfully finalized the file before the deadline and it is time for a company-wide presentation. More likely, this presentation will contain multiple channels, such as a channel for video conference and another channel for distributing any related documents to the participants as the conference progresses, for example, as in online course/presentation packages such as ConferenceXP [21] and I-MINDS [38]. More importantly, this network has to be shared with other usage (*e.g.*, another group of people may need to start a conference at the same time). In this case, FairOM is an excellent candidate

as it strives to support multiple simultaneous multicast sessions, no matter whether they are long lasting ones (such as video conference) or spontaneous ones (such as distributing certain files from time to time). In the figure, all the participants, including *G*, *H*, and *I* who did not contribute to creating the shared data, have joined the video conference.

As illustrated, put together, IDF, CVRetrieval, and FairOM provide a seamless framework for employees at *Smart Inc.* to support their collaboration.

1.4 Contributions

The current data consistency management schemes deploy predefined consistency levels before systems start to run and cannot change the consistency levels on the fly. These schemes cannot effectively support multiple applications to run simultaneously with different consistency requirements, nor do they support applications whose consistency requirements change from time to time. To improve data consistency management, this dissertation presents IDF and CVRetrieval.

First, this dissertation presents an inconsistency detection framework (IDF), which has two major parts: an efficient detection mechanism and an adaptive consistency control mechanism on top of it (*i.e.* IDEA, which will be formally presented in chapter 4). IDF provides a unified framework to adaptively control consistency levels on the fly. IDF is also able to support a wide range of applications by capturing their different semantics. These two features sets IDF apart from current consistency control protocols.

Second, to further improve the scalability of consistency control in general, this dissertation presents CVRetrieval, which separates passive observers from writers. CVRetrieval maintains consistency among writers via a standalone consistency main-

tenance protocol, such as IDEA, and keeps interested observers informed of the shared data via a publish-subscribe infrastructure. CVRetrieval improves data consistency management in Internet-scale distributed systems by providing a way to achieve highly scalable consistency control.

In the context of overlay multicast, enforcing fairness among participants is an important research issue because, in Internet-scale distributed systems, participants have equal status and usually do not want to contribute much more than others. While there are research that has proposed different fairness schemes, they only consider one multicast session and cannot effectively support multiple sessions. To improve overlay multicast, this dissertation presents FairOM, which supports multiple simultaneous multicast sessions while enforcing fair contributions. The novelty of FairOM is that, instead of treating fairness simply as letting each participant contribute once, it redefines fairness so that each participant, for one particular multicast session, contributes the same proportion of its available outgoing bandwidth. In this way, FairOM advances the state of the art of overlay multicast research by achieving both improved fairness and performance.

1.5 Dissertation Organization

This dissertation is organized as follows.

Chapter 1, this chapter, introduced the background of this dissertation, its targeted environment, and its main contributions. Also, it used a scenario to elaborate how the three major parts of the dissertation–IDF, CVRetrieval, and FairOM–can work seamlessly together to facilitate data management and data delivery.

Chapter 2 discusses related work to IDF, CVRetrieval, and FairOM.

Chapter 3 and chapter 4 discusses IDF, the Inconsistency Detection Framework.

In chapter 3 we will discuss the design and evaluation of its two-layer inconsistency detection mechanism, and an analytical study of the detection mechanism. Chapter 4 discusses IDEA, our adaptive consistency control protocol that is built on top of the detection mechanism.

Chapter 5 presents CVRetrieval, a consistency control mechanism that treats passive participants and active participants differently so that the consistency control scalability is further improved.

Chapter 6 presents FairOM, an overlay multicast protocol that is able to support multiple simultaneous multicast sessions fairly and achieve high performance. Within this context, we also briefly discuss the motivation and design principle of FairOM+, the next phase of research and development of FairOM.

Chapter 7 concludes this dissertation and discusses future work.

Chapter 2

Related Work

We discuss related work in this chapter. This chapter is presented in three parts, namely, work related to IDF, CVRetrieval, and FairOM, respectively.

2.1 Work related to IDF

For related work of IDF, we first briefly review consistency control approaches in distributed systems to put IDF in a broader context, which is then followed by a discussion on the work related to our adaptive and efficient consistency control protocol, IDEA.

2.1.1 Consistency Control in Distributed Systems

The problem of accessing shared data in a consistent way has been studied in different contexts and there are a variety of consistency protocols that suit different applications.

Traditionally, consistency control is discussed in a multi-processor environment. Originating from that environment, strong consistency control tries to achieve con-

sistency similar to that in a single processor environment. Lamport proposed a consistency model, called sequential consistency, which states that the operations of multiple processors should follow a sequential order [47]. However, this kind of consistency incurs huge synchronization overhead. Hence, efforts have been made to define other models with less overhead. As a result, several weaker consistency models, such as weak consistency [25] and release consistency [33], have been proposed in the literature.

Recently, it becomes both popular and necessary to replicate data and services across a cluster or even across the Internet. Unlike a multi-processor environment, an Internet-scale system spans a long distance and the communication is not very reliable. Things become even more complicated as mobile devices are added to the system which makes the connection more fragile. In this new environment, the traditional consistency control mechanisms do not work very well because they incur long delays and too much overhead [26].

To scale the applications to the scope of Internet, many newly proposed consistency control protocols take an optimistic approach and only maintain consistency by best effort [43, 99]. In Bayou [99], for example, updates are passively serialized on a primary-copy. During the update dissemination, no consistency is guaranteed.

However, truly optimistic consistency control cannot support applications that do need certain consistency guarantee, if not a perfect consistency. For example, e-business applications prefer to trade only a certain level of consistency guarantee for greater data availability: on one hand, they are willing to sacrifice a certain level of consistency guarantee for greater data availability, so as to reach more customers; on the other hand, they do need the consistency level to be above a threshold, if just to put a limit on monetary losses due to decisions made based on inconsistent information.

To support the tradeoff, it is useful to quantitatively specify the consistency level. In TACT [106], a set of parameters have been proposed to quantify the consistency level of an application and algorithms have been developed to bound the system-wide inconsistency within a certain level.

While IDF adopts TACT's definition of the consistency level, it is significantly different because instead of achieving a system-wide predefined consistency level, IDF recognizes applications' need for consistency adaptation and provides mechanisms to adapt the consistency level according to applications' semantics.

An important feature of IDF is its use of a universal and efficient two-layer inconsistency detection mechanism. Here, IDF's universality means that it supports all kinds of application types as long as their consistency requirements are translated into the format that can be understood by IDF. Lpbcast [27] is a gossip based broadcast protocol and is similar to the detection protocol we deployed in the bottom layer of our two-layer inconsistency detection framework. The difference between lpbcast and our work is that lpbcast is a pure gossip based broadcast protocol that does not differentiate between active and passive participants, and hence it cannot achieve the required efficiency and detection speed; while IDF is designed to minimize the delay of inconsistency detection by capturing the majority of inconsistency among the top layer that is much smaller in size and thus faster.

2.1.2 Adaptive Consistency Control in Distributed Systems

IDEA, the protocol built on top of the detection module of IDF, provides the adaptability that caters to applications' different needs for consistency guarantee.

For example, Yang and Li [105] have proposed a framework that puts a set of existing consistency protocols in a central module and, based on the current application's characteristics, attaches the right consistency protocol dynamically and adaptively.

While their work has the benefit of accommodating existing protocols, IDF is more advantageous in the sense that there is only one protocol needed to be deployed, which greatly simplifies the system design.

Content Addressable Parallel File System [102] discusses the adaptive consistency control in the context of a cluster file system. This system provides adaptability by allowing different applications to define different granularity of consistency control. This work is different from IDF in two aspects: (1) it lets applications choose which consistency level to start with, but do not allow applications to make such a choice on the fly; and (2) it deals with a tightly coupled cluster environment, where the communication is of high bandwidth and reliable, which is quite different from an open Internet-scale system that IDF deals with.

There are also several studies that try to obtain the best consistency guarantee when possible. For example, Om [108] maintains consistency among replicas by automatically generating a consistent replica when needed. Essentially, Om tries to achieve the best consistency whenever possible. Unlike Om, IDEA caters to applications' requirement by resolving inconsistencies only when the consistency level drops below a desirable level. Hence, IDEA avoids unnecessary overhead associated with resolving inconsistencies whenever they occur.

The quorum system [61, 62, 63, 69] is a widely deployed scheme to maintain consistency in a distributed system. Using quorums, it can tolerate system faults such as network partitions. However, in a quorum system, the consistent data will not be available if the node failures prevent a quorum from being formed. Unlike quorum systems, our two-layer framework is robust because if nodes fail, the consistency control mechanism can still work among the remaining working nodes.

IDF can benefit a wide range of applications such as autonomic computing [36, 41, 74] or Peer-to-Peer systems [22, 46, 68, 86]. Peer-to-Peer file systems use replication-

based schemes to prevent data loss. However, they either assume that the files are read only, or are based on optimistic consistency control. Designed as an infrastructure to enforce consistency according to applications' ongoing needs, IDF could benefit these Peer-to-Peer file systems by providing an adaptive and thus better consistency control mechanism.

2.2 Work related to CVRetrieval

Consistency control has attracted many researchers. A major challenge of designing an appropriate consistency control protocol is that the protocol has to enforce a required consistency level with only modest and acceptable communication overhead. Essentially, this is a scalability issue: if the communication overhead incurred is too high, the protocol cannot support a large number of participants. To improve scalability, CVRetrieval, cognizant of the fact that in a variety of applications a large number of participants are not actively collaborating for most of the time, uses different strategies to satisfy the consistency needs of active participants and passive participants. For active participants, CVRetrieval uses a protocol that periodically communicates with all participants to maintain a certain consistency level; for passive ones that are only interested in consistency occasionally and not actively participating in the collaboration, CVRetrieval lets them explicitly request consistent views from the system when the need arises in stead of joining the expensive consistency maintenance protocol all the time. The rationale is that it is much more cost-effective to satisfy a passive participant's need on-demand.

CVRetrieval is significantly different from any scheme that statically separates passive participants from active participants and maintain consistency only for the latter in two aspects. First, CVRetrieval is not merely differentiating active and

passive participants once and staying with a fixed differentiation all the time. Instead, the differentiation by CVRetrieval is a dynamic one, meaning that the active and passive participants are relative concepts and can change from time to time. The ability to capture this dynamism is a salient feature that sets CVRetrieval apart from any static approaches. Second, CVRetrieval assumes that passive participants do occasionally care about consistency, instead of assuming that they are not interested in the shared objects at all. CVRetrieval then actively satisfies passive participants' consistency needs with a publish-subscribe mechanism, which is a new feature in consistency control.

A wide range of consistency control approaches do not differentiate active participants from passive participants, and hence are very limited in their support of an Internet-scale distributed system with a large number of participants. For example, in MS NetMeeting, only one participant can operate on the shared object; all other participants will be blocked [4]. Other work, such as Local-lag and Timewarp [50, 103], are only able to support consistency control in a small cluster environment due to their frequent and expensive communication among participants to resolve inconsistencies. Overall, these previous studies still use consistency maintenance for all participants, which causes high overhead for a system with a large number of participants.

To the best of our knowledge, CVRetrieval is the first scheme to explicitly consider the retrieval aspect of consistency control in support of distributed online collaboration applications.

CVRetrieval is also a message delivery system because the main infrastructure of CVRetrieval is a publish-subscribe system. However, unlike traditional publish-subscribe system [78], the messages CVRetrieval publishes, the consistent view of the shared object, are time sensitive and CVRetrieval has to quantify the view's

consistency degree before the publishing operation, which has not been done before by traditional publish-subscribe systems.

Chang *et al.* use a multicast scheme to disseminate consistent information in a video conference context [18]. Multicast works well in the video conference context because the participants are relatively stable during the conference so that a well built multicast tree can be used throughout the conference. CVRetrieval, however, targets a more dynamic application background in which participants join and leave the system at any time. This dynamism makes it undesirable to use a multicast tree because for this to work a multicast tree must be rebuilt whenever a participant joins or leaves the system, which may incur very high overhead. On the other hand, a publish-subscribe approach is suitable for CVRetrieval because the infrastructure remains stable as long as the publishers and subscribers, instead of all participants, remain in the system.

2.3 Work related to FairOM

There are three ways to disseminate data across the network: unicast, broadcast and multicast. Unicast is suited for one-to-one communication in which a receiver maintains a link with the sender. Broadcast is for one-to-all communication and the sender sends message to all connected receivers. For the scenario where a sender wants to send messages to a subset instead of all of the participants, it uses multicast, which is a one-to-many communication.

Initially, multicast was implemented at the IP level [23] and this kind of multicast is called IP multicast. However, IP multicast is not widely available on Internet today because it needs special configurations on network routers, which are often not enabled by system administrators due to security concerns [48].

Recently, the notion of overlay multicast–*i.e.*, implementing multicast in the application layer–has been explored to provide an option for multicast. Overlay multicast [14, 20, 110] is a better choice than the IP level multicast for several reasons. First, overlay multicast does not need the special configuration of network routers that are often not enabled by system administrators due to security reasons. Second, it can be configured on top of the application level, thus providing opportunities to capture the semantics of the applications. And finally, it is easy to use and configure in practice. In an open Internet, fairness–*i.e.*, all participants share the multicast load–is important because it prevents hot spots and encourages participation.

There are different strategies to build an overlay multicast. Seminal work such as Overcast [39] and End System Multicast (ESM) [20] builds a single multicast tree for a multicast source. While these systems proved the feasibility and validity of overlay multicast, the asymmetric nature of the tree structure implies that a single-tree approach does not enforce fairness as the leaves in a tree have no contribution to the multicast transmissions at all.

To enforce fairness, multicast forest, or multiple multicast trees, is used to share forwarding load among all participants. In a typical scenario, the data are divided into n (a configurable parameter) stripes and each stripe is multicasted by a different multicast tree. In this way, each node will get a chance to contribute in some trees by serving as interior nodes. Two representative systems of this scheme are CoopNet [72, 73] and SplitStream [14]. The main design issue here is how to define fairness and how to effectively enforce fair contribution.

CoopNet defines fairness as the requirement that each node contributes something in a multicast. CoopNet uses a centralized mechanism to build multiple trees and, to enforce fairness, uses randomization in the tree construction process so that each node will have a chance to serve as an interior node. There are two main differences

between CoopNet and FairOM. First, CoopNet defines fairness as a requirement that each node contributes something and FairOM defines it as one in which each participant makes proportional contributions so that multiple multicast sessions can be supported. Second, FairOM utilizes both decentralized initial forest construction and centralized forest improvement while CoopNet is based on a purely centralized algorithm.

SplitStream builds a multicast forest in which each participant only serves as an interior node once, and serves as leaves in all other trees, and so it is a fair system in the sense that each participant contributes once and only once. The first difference between FairOM and SplitStream lies in the way fairness is defined. In SplitStream, a system is fair if each participant shares certain forwarding load. In FairOM, however, a system is fair if participants' contributions are proportional to their total available bandwidths so that multiple multicast sessions can be supported. Second, as an important data structure to facilitate the multicast forest building process, spare capacity group is used in SplitStream as a backup mechanism to build a complete forest; while FairOM has a similar concept, the spare capacity group in FairOM is staged, meaning that we have certain levels of contribution in the spare capacity group, and it plays a central role in enforcing the proportional contribution requirement.

Bullet [44] is a representative of mesh based multicast protocol, which builds an overlay mesh to disseminate data. Compared to a single-tree based multicast, Bullet has the benefit of removing forwarding bottleneck, which helps achieve high throughput. The design philosophy behind Bullet is to exploit excessive bandwidth while the primary design goal of FairOM is to enforce fair contribution among participants.

Recently, Bharambe *et al.* [5] presented the impact of heterogeneous bandwidth on DHT-based (Distributed Hash Table) multicast protocols, such as Scribe [15], the

origin of SplitStream. However, their work is based on DHT-based multicast while ours on unstructured multicast.

FairOM assumes that the multiple multicast sessions cover the same group of participants. However, certain applications, such as multimedia and distributed computing, can potentially have multiple multicast groups that, although overlap, are not identical. FairOM+, the next stage of research and development of FairOM and not fully reported in this dissertation, supports proportional contributions in such an environment.

In the context of supporting multiple overlapping but non-identical multicast groups, Chu *et al.* proposed a scheme to divide the multimedia packets into several stripes and multicast each with a different multicast tree [37]. Then, based on its bandwidth, a participant chooses to join one or more multicast groups. However, they treat joining multiple multicast groups as an optimization problem: the clients receive high QoS by joining more sessions. In contrast, FairOM+ treats joining multiple multicast groups as a feasibility problem: the clients' requirements are only satisfied when they can join the number of sessions they request. From this perspective, FairOM+ can potentially complement the work by Chu *et al.* as they target different aspects of the same problem.

2.4 Summary

The objective of this dissertation is to improve data consistency management and overlay multicast in Internet-scale distributed systems. Therefore, this chapter discussed the work related to data consistency management and overlay multicast respectively. The related work surveyed and discussed in this chapter show that our proposed schemes–IDF, CVRetrieval, and FairOM–advance the state of the art in

their respective areas by successfully addressing some pressing open issues or otherwise complementing some existing approaches.

Chapter 3

IDF: Inconsistency Detection Framework – The Framework and Its Two-Layer Based Timely Inconsistency Detection Module

Replicating data and services is an attractive strategy to increase availability and performance in distributed systems [13, 87, 90, 91, 106]. It increases availability because, given multiple replicas in existence, the data are still available even when one or more replicas crash as long as one replica survives; it increases performance because users can fetch the data from a near-by replica and, by doing so, the system's response time can be dramatically reduced. For these reasons, replicating data and services has been widely adopted in distributed systems, especially in Internet-scale ones in which availability and performance are of critical importance [13, 106].

Conventionally, consistency control uses either well-designed protocols to avoid inconsistency up-front, such as strong consistency protocols [8, 95] to avoid any in-

consistency, or optimistic consistency protocols to increase the availability and tolerate relaxed inconsistency among nodes [43, 99]. In this dissertation, we refer to such schemes as *inconsistency avoidance*.

While inconsistency avoidance can be effective in a small-scale networked system, such as a small cluster, it has some drawbacks in an Internet-scale environment. On the one hand, a strong consistency protocol can be very costly to maintain due to the membership maintenance and strict protocol enforcement cost. And because of the relatively unreliable network transmission in large-scale networks, it is impossible in most cases to maintain strong consistency [26].

On the other hand, while an optimistic consistency control protocol relieves the costly maintenance and strict enforcement burden associated with strong consistency control protocols, it also does not suit a large-scale distributed system because it is predefined. It is not preferable because, in a modern computer system, multiple applications are deployed at the same time [49] and, even for one application only, the application's requirement for consistency can change from time to time. These two points are further elaborated as follows.

- **A system may run multiple applications with different requirements of consistency.** In this scenario, a predefined consistency protocol can be either overkill when an application does not need that strong consistency, or insufficient when an application needs a stronger one. While several consistency protocols can be deployed to cater to different applications simultaneously, it would inevitably increase the complexity of the system design and degrade system's performance due to the cost associated with operating each consistency protocol.

- **For an application, the requirement of consistency may change from**

time to time. Take a virtual white board as an example, in which participants draw on a virtual white board to communicate and collaborate. In this scenario, a participant may have less consistency requirement in the first several minutes when the discussion is about the background, but can have stronger consistency requirement when an important topic arises. Under such circumstance, a predefined consistency level does not reflect participants' changing requirements of consistency over time.

To this end, this dissertation proposes a framework to detect inconsistency in a timely manner as it occurs instead of avoiding it in the first place or depending on any predefined consistency control protocol. Upon detection, the framework can adjust the system's consistency level adaptively based on applications' semantics. We refer to this approach as *inconsistency detection* and the framework is called as IDF (Inconsistency Detection Framework).

3.1 Advantages and Overview of IDF

Compared to inconsistency avoidance, several advantages can be obtained from an inconsistency detection framework implemented as middleware. First, it removes the costly membership management requirement that is used to enforce consistency in the first place. Instead, it detects the inconsistency when it happens, which makes the system more scalable.

Second, after the inconsistency is detected, the middleware can respond based on the application semantics. That is, it resolves the inconsistency when it is needed, while letting the detected inconsistency continue to exist when it is tolerable or even preferred. For the latter case, consider an air ticket booking system, an example of e-business, which requires a consistency control protocol allowing inconsistency–

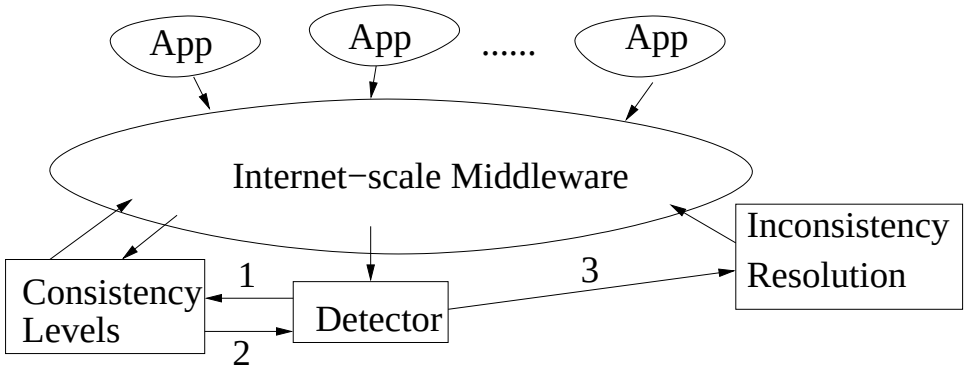


Figure 3.1: Architecture of the Inconsistency Detection Framework

overselling–to exist within a certain threshold to cover the returned tickets. The benefit of this semantic-based consistency control is materialized by IDEA, an infrastructure for detection-based adaptive consistency control that is an important component and will be formally discussed in the following chapter.

As an alternative to inconsistency avoidance, IDF detects inconsistency among nodes in a timely manner. An overview of IDF is shown in Figure 3.1.

In this framework, multiple applications share data and services through the support of the Internet-scale middleware and the inconsistencies among them are detected by the detector. Upon detection, the detector consults with the inconsistency level monitor (step 1 and step 2) before reaction is initiated. Based on the applications' semantics, if the inconsistency is tolerable, the detector does not react; otherwise, the detector informs the inconsistency resolution model to resolve this inconsistency (step 3).

The arrows from the middleware to the detector module means that the detector gets information from the middleware, and the arrow from the inconsistency resolution module to the middleware means that the module can influence the middleware. The two arrows between the consistency level module and the middleware indicates that it can get the consistency levels for applications from the middleware and it can

potentially help the applications adjust their consistency levels.

As we can see, there are two core modules in this framework: the timely inconsistency detection mechanism (detector) and the adaptive semantic-based consistency resolution (consistency level and inconsistency resolution). The consistency level component is needed for both detection and resolution modules.

In this and the following chapters, we discuss IDF in three steps: (1) we present design and evaluation of the underlying inconsistency detection mechanism of IDF; (2) we discuss a comprehensive analytical study of the effectiveness of the inconsistency detection mechanism–a critical performance and correctness evaluation; and (3) we present IDEA (an Infrastructure for DEtection-based Adaptive consistency control), an adaptive consistency control protocol that is built on top of the inconsistency detection mechanism and directly interacts with both application developers and end users. This chapter covers the first two steps. The third step is covered in the following chapter.

3.2 Two-Layer Based Timely Inconsistency Detection

The timely inconsistency detection mechanism is the foundation of IDF and provides a versatile inconsistency detection function to modules above. Essentially, the detection module provides a powerful API (Application Programming Interface) to IDEA: *detect (update).* Given an update, this operation will return "success" when there is no inconsistency or "fail" when there is conflict (thus inconsistency) detected. Theoretically speaking, this detection mechanism, as a rather independent component in IDF, can be used by other consistency control mechanism (other than IDEA) as well.

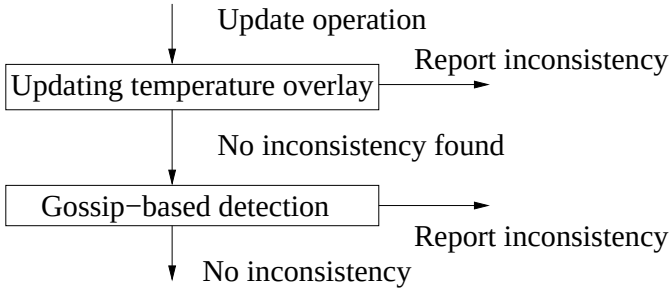


Figure 3.2: Two-layer infrastructure

3.2.1 The Basic Idea

The basic idea behind the two-layer detection module is to build a two-layer overlay on top of the underlying network based on nodes' updating history. The top layer is constructed based on nodes' updating history, or updating temperature, and is referred as "temperature overlay". The bottom layer is a gossip-based inconsistency detection module consisting of the rest of the nodes involved in the shared object/file and is used as a backup and only triggered when the top layer does not find any inconsistency. The architecture of the detection mechanism is illustrated in Figure 3.2.

In the temperature overlay, each node tracks its own updating history and exchanges this information periodically with others through the RanSub [45] protocol that will be explained shortly. When a node commits an update, this update is propagated in the temperature overlay in such a way that the nodes that update this file most frequently are visited first. The rationale behind this design is that a user usually works on a file for a certain period of time. For example, he/she may edit a report for 10 minutes, then debug, and thus update, a C++ file for 20 minutes. The hypothesis of this design is that, through this two-layer framework, most inconsistencies can be detected in the top layer quickly, probabilistically speaking. The two enabling technologies and their roles in timely inconsistency detection are discussed

below.

3.2.2 Enabling Technologies

Here, we discuss two enabling technologies on which the timely inconsistency detection mechanism depends.

RanSub

RanSub [45] is proposed to address the challenge of locating disjoint content within a system. RanSub distributes random subsets of nodes' information through a tree by executing the collect and distribute processes. The collect process starts from the leaves and goes all the way up to the root. In this process, each node informs its parent about the information it has about its sub-tree by constructing a representative subset of all the nodes in it and then delivers the information all the way up to the root. The distribute process then starts from the root and delivers the information from a subset of nodes to each of its children. The child then distributes a subset information determined based on its own information and the subset information received from its parent to pick a subset of nodes and distribute the information further down the tree. Using the *collect* and *distribute* processes, RanSub delivers information from a random subset of nodes to each node per *epoch*.

A key operation in RanSub is the *compact* operation. In the *collect* process, *compact* constructs a fixed size subset to randomly and uniformly represent its sub-tree members. In the *distribute* process, *compact* constructs a fixed size subset to randomly and uniformly represent the global information for each of the current node's children. There are several flavors of the *compact* operation, and we choose the *RanSub-all-non-identical* option to deliver the update information. This option distributes a random subset of nodes among the whole system, thus is suitable for

the inconsistency detection purpose.

While we use RanSub as an underlying protocol to exchange nodes' information among one another, we advance RanSub by proposing an *interest-group based* collect/distribute process.

Gossip-based data dissemination

To alleviate the scalability bottleneck of information dissemination, gossip based data dissemination has been proposed. Simply put, gossip-based schemes disseminate data by periodically exchanging information among neighboring nodes. The Lightweight Probabilistic Broadcast (lpbcast) [27] scheme advances the gossip-based scheme by eliminating the requirement of global view of the nodes. Instead, a node maintains a fixed size of a random subset of this system. Then a node disseminates non-duplicate packets to a randomly chosen subset of neighbors in its local view every T seconds. To minimize bandwidth cost, each message only travels a certain number of hops.

In the context of the two-layer inconsistency detection module presented in the next section, the bottom layer uses this gossip-based dissemination to distribute updates received by a bottom-layer node receives to other bottom-layer nodes periodically.

3.2.3 Design

Now we discuss the detailed design of the two-layer timely detection mechanism.

Measuring the updating patterns

An important operation in the detection module is to measure the updating temperatures of nodes. Basically, we let each node track the number of its updates operations with regard to a particular file in a certain period of time (in the current study, we use

30 seconds as the default). Intuitively, the higher the number of updates on a file is, the higher the updating temperature of this node is. Because there could potentially be many files in a node's machine, there is actually a temperature vector in each machine, with each file having an entry in this vector.

The scheme, while correct, is obviously not scalable or network-bandwidth efficient. For example, a node may have 10,000 files in its machine but may only modify less than 10 files in a certain period of time. In this case, there is really no point of keeping a vector in which 99.9% entries are 0 (no updates in the past). To solve this problem, we introduce the notion of interest-group based temperature collection and distribution. However, before we proceed to describing that optimization, we first need to discuss the mechanism by which the nodes learn updating temperatures from each other, thus laying the background for such description.

Learning the updating patterns

We assume that each node is tracking its own updating temperatures and has prepared its temperature vector. Then the temperature information is propagated via RanSub. Recall that RanSub assumes the existence of a multicast tree that covers all the nodes and use that to collect and distribute the nodes' information.

The only concern we have about RanSub is that it is based on a single-tree structure and thus can not tolerate even a single interior node failure. As identified in its original paper, possible ways to work around this include the use of multiple trees to substitute the single-tree structure. For example, we can deploy a multi-tree based multicast, such as SplitStream [14] or our own forest-based multicast protocol FairOM (FairOM will be presented later in this dissertation), as the underlying communication mechanism of RanSub. We expect such mechanisms to dramatically increase the resilience to node failures of the detection framework.

Interest-group based temperature collection/distribution

One critical question about utilizing RanSub to propagate temperature information is how to minimize the network bandwidth cost. Without optimization, a huge amount of data (updating temperature information in this case) could be sent across the network, potentially putting significant strain on the network. In this section, we propose an interest-group based temperature collection and distribution scheme to minimize the network bandwidth cost.

More specifically, we let the nodes report only the updating temperature of the files that they are interested in the *collect* process and every interior node tracks the interested files of its sub-tree. In the *distribute* process, an interior node only accepts and forwards the updating temperatures of files which are of interest to the nodes within its sub-tree.

To discuss this mechanism and analyze its bandwidth cost more formally, we define the parameters as follows.

Assume that the total number of nodes in the system is n and each node has k_i number of files in total. Each node is interested in p_i files within a certain period of time. Suppose that there are q exchanges involved in propagating the temperature information. Then we assume that each updating temperature entry has a size of s bytes.

Because RanSub collects information through a tree structure, for the purpose of analysis, we assume that an interior node maintains m neighbors on average. If we assume a balanced tree, then the height of tree, h, is the smallest integer larger than $log_m(n)$.

Thus, the total number of messages exchanged among the tree nodes in the *collect* process is:

$$N = m + m_2 + \cdots + m_h \tag{3.1}$$

Each node only submits $k_i \times s$ bytes of data in the *collect* process. In the *distribute* process, a fixed number of nodes' information is distributed. Suppose the fixed number is b, then the message is of size $b \times k_i \times s$. Let k_a denote the average number of interested files. If the length of an epoch is L seconds, then the total bandwidth cost is:

$$\begin{aligned} TotalBW &= (N \times k_a \times s + N \times b \times k_a \times s)/L \\ &= (b+1) \times N \times k_a \times s/L \end{aligned} \tag{3.2}$$

There are N links in this RanSub tree, so on average, the bandwidth cost is:

$$\begin{aligned} AvgBW &= TotalBW/N \\ &= (b+1) \times k_a \times s/L \end{aligned} \tag{3.3}$$

Given a network with parameters b of 100, k_a of 5, s of 10 and L of 30, the bandwidth cost is 183 bytes per second, which is small enough that can be supported by a dial-up connection.

To further reduce the bandwidth cost, we use a threshold to control the reporting of updating temperatures. If a node has a temperature less than a threshold t for a file and an interior node has already had at least k entries for this file, then the lowest temperature information will be dropped.

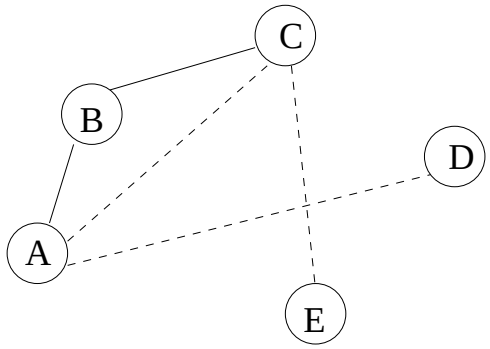


Figure 3.3: Inconsistency detection

Two-layer inconsistency detection

When a node is updating a file, it consults the two-layer inconsistency detection module to detect any possible conflicts as follows. First, the node checks its local cache to carry out the top-layer detection, where it chooses the node with the highest updating temperature on the file being updated and forwards the update to that node. If the receiving node has an update that conflicts with the one it receives, it notifies the sender directly. Otherwise, the receiving node chooses another node in its local cache that has the highest updating temperature on this file and relays the update to it. In addition, the traveling path is attached to the update to prevent the same update from traveling back to a previously visited hop.

If there is no conflict, then the update will stop eventually at a node that has no nodes in its local cache and has not been visited before, implying that the top-layer has been completely traversed by the update without finding any conflict. Having failed to detect any inconsistency at the temperature-based top-layer, the gossip-based bottom-layer inconsistency detection is triggered. The update is then sent to the last hop's friend list and its friend sends it out to the friend's friends, etc. To control flooding, each update only travels up to a predefined number of hops. This process is illustrated in Figure 3.3.

In Figure 3.3, the solid line represents the top layer (updating-temperature based) and the dotted line represents the bottom layer (gossip based). In the figure, when node A commits an update, it first traverses the top layer to check with B and C to detect any inconsistency. If neither of them conflicts with A's update, C (the last hop in the top layer) starts the detection from the bottom layer. In this case, if E happens to conflict with A, then this inconsistency will be detected in the bottom layer.

Version vectors [75] are used to detect conflicts among updates. A version vector tracks the number of times a file is updated by a certain user to detect inconsistency. For example, version vector $<A$:3 B:5$>$ is earlier in time than version vector $<A$:4 B:7$>$. Two version vectors u and v are comparable if and only if $u < v$, $u = v$ or $u > v$. If not, they conflict with each other. For example, $<A$:5 B:3$>$ conflicts with $<A$:3 B:6$>$.

In an ideal case, if all the nodes never change their interested files, then all the inconsistency can be resolved in the top layer. However, this is not the case in practice. The analysis about the case where the nodes change their interested files with certain rate, based on different applications' semantics, is conducted in Section 3.3.

Caching and garbage collection

Two forms of caching are considered here to improve performance. First, a node caches the temperature information. When the temperature information about other nodes arrives at a node, it is stored in the node's local cache. When new information arrives at a node, the local cache is updated if the information is already in the cache. Otherwise, the new information is added to the cache.

The second caching scheme is to help minimize the update routing cost. Here, a node caches the propagation path along which the update traverses within the top

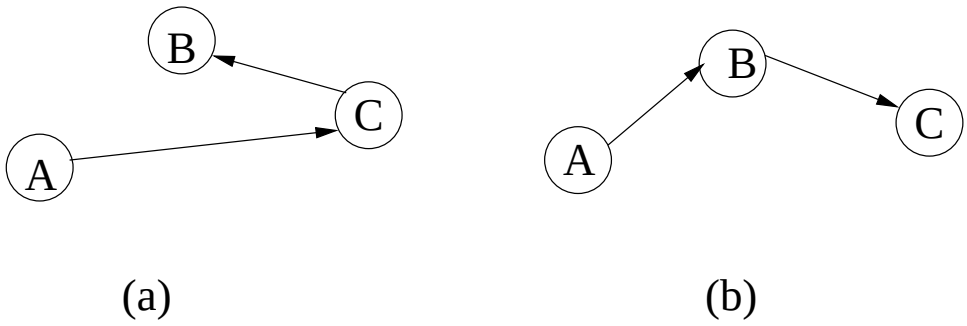


Figure 3.4: Issues with update propagation path

layer to detect any inconsistency.

Figure 3.4 illustrates the process through an example. In (a), the update from A is forwarded to C, then B. The rationale is that the temperature of the updated file in C is higher than that of B, thus there is a better chance of resolving inconsistency by visiting C first than by visiting B first. However, there is a tradeoff between high probability of resolving inconsistency and low routing cost. Consider (b), A visits B first, then C without sacrificing much routing delay. In general, (b) is a better update propagation scheme than (a).

We use a simple heuristic scheme to deal with the tradeoff. First, each node picks three nodes with the highest temperatures on the file being updated and compares their routing delays, based on information cached locally. It then chooses the closest node and forwards the update to it. There are certainly other options available. For example, more complicated schemes could be developed to derive a formula and assign different weights to the two parameters. However, to accurately choose the right weights, extensive empirical studies need to be conducted to investigate the issue and we leave this to future work.

Garbage collection is used to keep the temperature history fresh. To check the freshness, each entry in the local cache is assigned a time stamp, which basically tracks when that entry was last updated. If an entry is updated again, then the

update time is reset to the new time. All the entries in the local cache are sorted according to its freshness. Periodically (we currently use a period of 3 minutes) the garbage collection scans the list and removes all the entries that are older than that period.

Discussions

First, we apply the following optimization to improve the likelihood that an inconsistency from the bottom layer can be captured by the top layer. If a node becomes interested in a file for the first time, we let the node report its interest in new files one epoch before its updating. This mechanism increases a first-time writer's visibility in the system and in turn can make its updates more likely to be captured by other writers. In practice, this can be done when a user first opens a file. This assumption is valid as long as the open operation is at least an epoch ahead of the real updating operation and we believe that this assumption is reasonable.

Then, up to this point, our discussion about updating operation is based on a rather abstract concept. In practice, however, there are several forms of update operations, such as creating, modifying, and deleting operations, of which the delete operations needs some extra attention because it will be semantically wrong if one user still modifies a file while the other user has already deleted it. One possible solution to this problem is to immediately cancel all ongoing modification processes when a file is deleted. Nonetheless, the proposed two-layer inconsistency detection module can benefit all the cases by improving the efficiency of inconsistency detection.

3.2.4 Detection Delay and Maintenance Cost

As the first part of evaluating the two-layer inconsistency detection mechanism, we experimentally evaluate two aspects of its performance: detection delay through pro-

totyping and maintenance cost through simulation. The probability of detecting inconsistency in the top layer, another critical measurement of the correctness and effectiveness (of this mechanism is conducted through a comprehensive analytical modeling in Section 3.3 due to its complexity. The importance of this measure lies in the fact that the usefulness of the top-layer depends on whether or not the majority of inconsistencies can be captured there.

Detection delay

Here we are interested in the difference between the absolute detection delay of the top layer and the bottom layer because, only when the detection delay of the top layer is significantly smaller than that of the bottom layer, can the high detection probability in the top layer lead to efficient detection.

To quantitatively measure the detection latency in a real environment, we have implemented a prototype of the two-layer detection module and deployed it on the Planet-Lab [77]. In the setup, we deploy the module on 12 Planet-Lab nodes that span US and Canada. In the beginning of the experiment, we designate three of the 12 nodes to serve as the initial top layer overlay and the bottom overlay contains the remaining nine nodes. We choose three, not other numbers, as the size of top layer because we do not expect the conclusions we will draw from other configurations to be any different.

In each experiment, we choose a number of simultaneous writers and every writer represents an entity of updating that updates a given file to warm up the system. The writers are randomly chosen among the 12 nodes but we assume that there is at least one writer in the initial top layer. As stated in the protocol, version vectors are deployed to detect inconsistency. After the warm-up, we randomly choose two writers to issue updates for the given file.

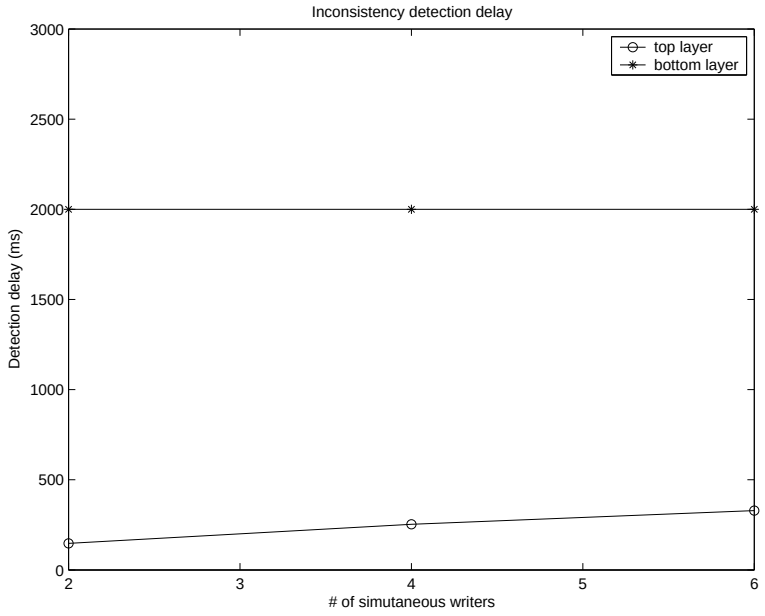


Figure 3.5: Inconsistency detection delay for two layers

Figure 3.5 plots the average detection delay of a chosen node as a function of the number of concurrent writers in the warm-up process. The number of simultaneous writers does not reach 12 because, if we do so, the bottom layer will have 0 nodes and we won't have a two-layer structure anymore. Because the gossip-based protocol sends update information periodically, its detection delay should be no smaller than the predefined period. Thus, we use two seconds (the period) as the detection delay for the gossip-based protocol, a choice that favors the gossip protocol because we essentially suppose gossip protocol can find an inconsistency in exactly one hop, while in a large system, it could take many hops, thus longer delay, to find an inconsistency.

As the figure shows, the detection delay of the top layer increases when the number of simultaneous writers in the warm-up process increases. This is due to the fact that the size of the final top layer grows with the number of concurrent writers in the warm-up process, which in turn implies a longer delay.

Also, the detection delay by the top layer is much smaller than by the gossip-

based protocol, which validates our hypothesis that the top layer can detect the inconsistency much faster than the gossip-based protocol (the bottom layer). Because the vast majority of inconsistency occurrences can be detected by the top layer at a much higher speed than the gossip-based protocol, we conclude that the proposed two-layer detection module can indeed detect inconsistency in a timely manner, thus achieving our design goal.

Maintenance cost

Now that we have established the efficiency of the top-layer detection, another critical question to ask is whether this ability is achieved in a cost-effective way. To quantitatively determine the cost-effectiveness, we use the number of messages received by each node during the maintenance process to evaluate the maintenance cost of the temperature overlay. In order to best evaluate the system performance in a large/Internet-scale distributed system that is usually not attainable in an experimental setting, we rely on simulation and choose the Transit-Stub model [109] to simulate a large/Internet-scale physical network for the purpose of assessing maintenance cost. The Transit-Stub model is widely used for simulation purpose because it reflects the current Internet topology. In the following simulation, the Transit-Stub model generates 1452 routers that are arranged hierarchically. Then we generate 1,000 end nodes and attach them to routers randomly with a uniform distribution. Each end node was directly attached by an LAN link to its assigned router. Thus, on average, each node is in a different LAN. We believe that this reflects the nature of an Internet-scale distributed system because it has a large number of nodes and all nodes are distributed on different LANs. We run the simulation five times and use the mean value as the final result.

We run the two-layer inconsistency detection module for 800 seconds. Because the

Max	Mean	Median
4680	51.9	26

Table 3.1: Maintenance cost in terms of number of messages exchanged

RanSub process starts at the end of every 30 seconds, there are 26 epochs involved in the entire 800-second simulation. At the end of the simulation, we collect the number of messages received by each node and the result is illustrated in Table 3.1.

Although the Max (which occurs at the root of the information-collection tree that RanSub uses) is much higher than the mean value, it must be pointed out that it is accumulated over 26 epochs. Thus within each epoch, it receives 180 messages which equals to 6 messages per second (one epoch runs every 30 seconds). Even if the size of a message is 1KB, the network bandwidth cost is only 6KB/s for the root, indicating that the maintenance cost will not overwhelm the root.

The maintenance cost can be further reduced by utilizing multiple-tree based RanSub. For example, if there are multiple trees that are configured with different roots, each epoch can then use a different root to run the RanSub procedure, in which all the roots equally share the maintenance cost.

3.3 Success Rate of Top Layer Detection through Analytical Modeling

The two-layer design of the detection mechanism is based on the hypothesis that the majority of inconsistencies can be detected among the top layer. While this hypothesis is intuitively correct due to the definition of the top layer–writers that write most frequently form the top layer–we feel that there is still a need to analyze the precise success rate of the top-layer detection for two reasons.

First, while we have shown that IDF achieves low detection delay in the top layer

and incurs minimal bandwidth cost–it can be supported by dial-up connections, it is still not clear exactly what the percentage of total inconsistencies can be captured among the top layer. If the percentage is not very high and IDF has to go through the bottom layer regularly, the significance of the timely detection in the top layer is diminished.

Second, different applications may exhibit different updating patterns as well as user distributions, thus it is highly likely that, for different applications, the success rate of the top-layer detection in IDF can be different. Hence, we need to put the success rate of the top-layer detection in IDF in the context of a wide range of applications. More importantly, if we can gain some useful insight through this analysis, we can hopefully identify certain parameters that can be tuned to improve the accuracy of the top-layer detection for different applications, which can then be used by a practitioner to adjust or optimize IDF for specific applications when adopting IDF.

Thus, this analysis tries to answer two questions. First, what is the success rate at which an inconsistency can be detected by the top layer, which in turn results in much faster detection? Second, how can practitioners relate the successful rate to a particular application, so that they can determine how to adapt or even optimize their IDF designs for specific applications?

We approach the two questions by first developing an analytical model to characterize the main principles of IDF, and then accurately deriving the successful rate of inconsistency detection by the top layer in all cases. Finally, a unified formula is derived to relate these success rates to specific applications.

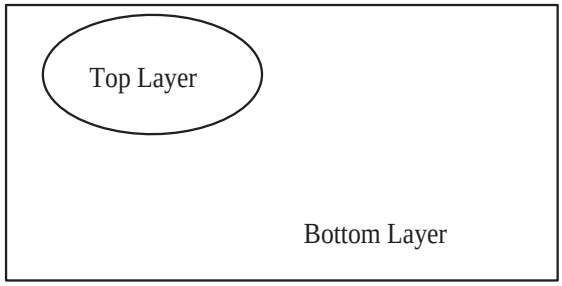


Figure 3.6: An abstract view of the two-layer system for a given file

3.3.1 The Analytical Model

Assumptions and definitions

The goal of this analysis is to derive the success rate of the top-layer inconsistency detection. In this analysis, we assume that there are n nodes in this system and, after a warm-up period, the top layer associated with a given file f has been formed. This can be visually represented by Figure 3.6.

A copy of file f exists in every writer of it. For the purpose of analysis and simplicity, we assume that, without loss of generality, there are two concurrent writers A and B for file f, and each writer can be from either the top layer or the bottom layer. While there can potentially be more than two concurrent writers, the analysis can be simplified to the case of two writers without loss of generality because, as far as inconsistency detection is concerned, it is a matter of two writers–the writer that triggers the detection in the first place and whether its inconsistency (in the form conflicting updates) is detected when its update conflicts with that from the first concurrent writer, thus all other concurrent (but later) writers do not matter in this case.

Now let us first define some terms and parameters that we will use throughout the rest of the paper, starting with the following seven events.

1. E: IDF fails to detect the inconsistency between A and B in the top layer.
2. e_1: writers A and B are both from the top layer.
3. e_2: one writer comes from the top layer, and the other comes from the bottom layer.
4. e_3: A and B are both from the bottom layer.
5. d_1: IDF fails to detect inconsistency between A and B in the top layer given event e_1.
6. d_2: IDF fails to detect inconsistency between A and B in the top layer given event e_2.
7. d_3: IDF fails to detect inconsistency between A and B in the top layer given event e_3.

It must be noted that, when we say that IDF fails to detect inconsistency in the top layer, we do not mean that IDF cannot detect the inconsistency. Actually, IDF can detect all the inconsistency eventually through the bottom layer, although it will be much slower than the detection in top layer. Thus, if IDF fails to detect an inconsistency in the top layer, the bottom layer has to be triggered, only at the expense of performance, not the correctness of IDF. As a measure of the performance of IDF, we denote the probabilities of several events as follows.

1. The overall success rate of the top layer detecting an inconsistency is denoted as P_{succ}.
2. The probability of event E is denoted as P.
3. The probability of event e_i ($i = 1, 2, 3$) occurring is denoted by h_i. Obviously, the sum of h_i ($i = 1, 2, 3$) is 1.

4. The probability of event d_i (i = 1, 2, 3) occurring is denoted by p_i.

P_{succ} is the overall success rate of the top layer detecting an inconsistency. Clearly, we have

$$P_{succ} = 1 - P = 1 - \sum_{i=1}^{3} h_i \times p_i \tag{3.4}$$

The analysis

In this section, we analyze the performance of IDF by deriving the formulas for p_i (i = 1, 2, 3)–the failure rates–and use these values as an indicator of the performance of IDF (success rate = 1 - failure rate). We need to note that the value of p_i is an abstract value. That is, these values do not directly relate to particular applications. In order to translate p_i to the performances with regard to particular applications, we need to derive P_{succ} as indicated in formula 3.4 in which h_i (i = 1, 2, 3) is determined by the applications' characteristics. The derivation of P_{succ} will be presented in Section 3.3.2.

Derivation of p_1 Recall that p_1 is the probability of event d_1, which in turn implies that event e_1 happens. When event e_1 happens, concurrent writers A and B are both from the top layer. According to the system design, the top layer is able to detect inconsistency whenever the two writers can find each other. In this case, they can find each other through the top layer, which is visible to both of them. Thus the probability that the system fails to detect an inconsistency in event e_1 is 0. So we have

$$p_1 = p(d_1) = 0 \tag{3.5}$$

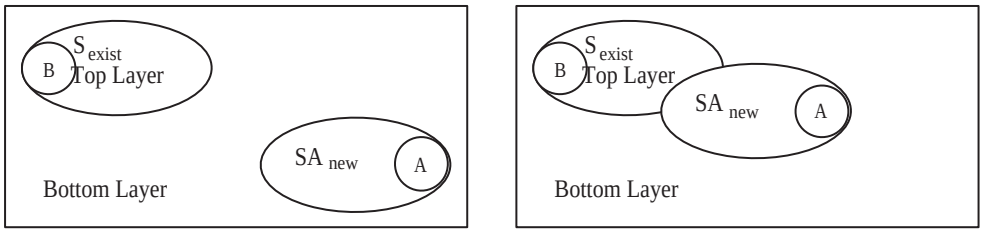


Figure 3.7: Two cases given event e_2

Derivation of p_2 Note that p_2 is the probability of event d_2, which in turn implies that event e_2 happens. In this case, one of the two writers comes from the top layer and the other is from the bottom layer. Without loss of generality, let us suppose that B is from the top layer while A is from the bottom layer.

Let us assume that there is a set S_{exist} of n_1 nodes in the current top layer, then, from the assumption, we know that B belongs to the set S_{exist} and A is form the bottom layer. After A starts the update operation, its update will be distributed to certain nodes in the network overtime. Without loss of generality, we assume that there is a set SA_{new} of n_2 nodes that have learned that A has become an active writer of the file f by the time both A and B become concurrent writers.

Because there is no particular requirement of these SA_{new} nodes, it is possible that, probabilistically speaking, some nodes in SA_{new} are from S_{exist}. If sets S_{exist} and SA_{new} do not intersect, the top layer will fail to detect this inconsistency because A and B cannot meet each other and the bottom layer has to be triggered (Figure 3.7(a)). On the other hand, if sets S_{exist} and SA_{new} intersect, this inconsistency can be detected in the top layer because, according to the detection protocol, as long as A and B can meet each other, the inconsistency can be detected (Figure 3.7(b)).

Assuming that there are a total of n nodes in the system, we have

$$p_2 = p(d_2) = \frac{C_n^{n_1} C_{n-n_1}^{n_2}}{C_n^{n_1} C_n^{n_2}} = \frac{C_{n-n_1}^{n_2}}{C_n^{n_2}} \tag{3.6}$$

To calculate p_2, we use the Stirling's approximation [84], which states that

$$n! \approx \sqrt{2\pi n} e^{-n} n^n$$

Thus we have

$$p_2 = p(d_2) \approx \sqrt{\frac{(n-n_1)(n-n_2)}{n(n-n_1-n_2)}} \frac{(n-n_1)^{n-n_1}(n-n_2)^{n-n_2}}{n^n (n-n_1-n_2)^{n-n_1-n_2}}$$

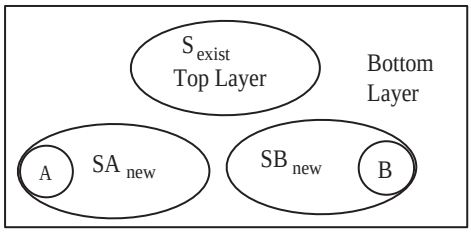
For the purpose of easy calculation, we can then use *log* operation, such as

$$\begin{aligned}\log(p_2) = \frac{1}{2}(&\log(n-n_1) + \log(n-n_2) - \log(n) - \log(n-n_1-n_2)) \\ &+ ((n-n_1)\log(n-n_1) + (n-n_2)\log(n-n_2)) \\ &- (n\log(n) + (n-n_1-n_2)\log(n-n_1-n_2))\end{aligned}$$

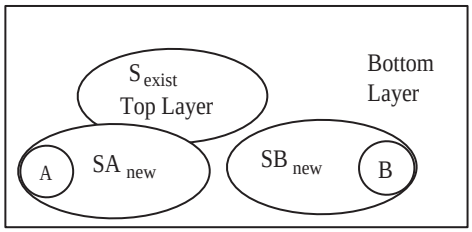
At this point, $\log(p_2)$ is very easy to calculate and p_2 can then be calculated accordingly.

Derivation of p_3 The value of p_3 is the probability of event d_3, which in turn implies the occurrence of event e_3, meaning that both A and B are from the bottom layer. Following the same analysis in the derivation of p_2, we can assume the existence of three sets S_{exist}, SA_{new} and SB_{new}, because both A and B are from the bottom layer.

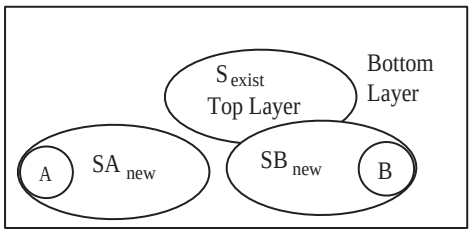
Depending on whether the three sets intersect with each other, there are six cases that need to be considered. The six cases are illustrated in Figure 3.8 (from 3.8(a)



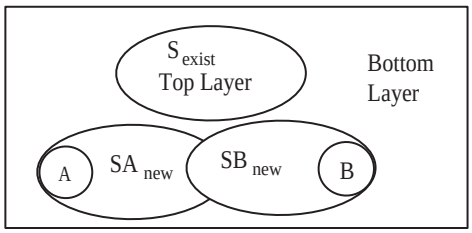
(a) None of these sets intersect with one another



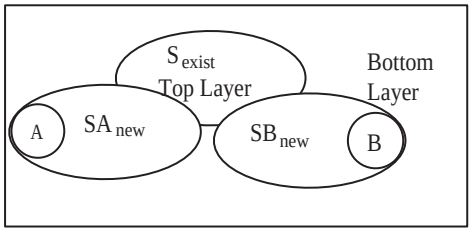
(b) S_{exist} intersects with SA_{new}



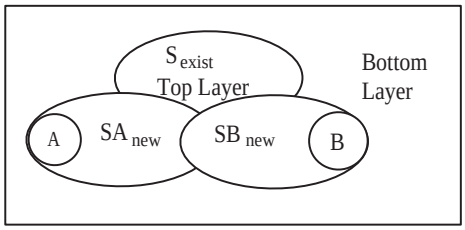
(c) S_{exist} interests with SB_{new}



(d) SA_{new} intersects with SB_{new}



(e) Both SA_{new} and SB_{new} intersect with S_{exist}, but SA_{new} and SB_{new} do not intersect with each other



(f) Three sets intersect with one another

Figure 3.8: Six cases given event e_3

to 3.8(f)).

For simplicity of analysis, we use n_1, n_2 and n_3 to denote the sizes of S_{exist}, SA_{new}, and SB_{new}, respectively. According to the protocol of the two-layer detection mechanism, the top layer will fail to detect any inconsistency in cases (a), (b) and (c); and will be able to detect inconsistency in the remaining cases (d, e, and f) where A and B can meet each other eventually.

Let g_1, g_2 and g_3 denote the probabilities of cases (a), (b) and (c), respectively. Then we have

$$g_1 = \frac{C_n^{n_1} C_{n-n_1}^{n_2} C_{n-n_1-n_2}^{n_3}}{C_n^{n_1} C_n^{n_2} C_n^{n_3}}$$

$$g_2 = \frac{C_n^{n_3} [C_{n-n_3}^{n_1} C_{n-n_3}^{n_2} - C_{n-n_3}^{n_1} C_{n-n_3-n_1}^{n_2}]}{C_n^{n_1} C_n^{n_2} C_n^{n_3}}$$

$$g_3 = \frac{C_n^{n_2} [C_{n-n_2}^{n_1} C_{n-n_2}^{n_1} - C_{n-n_2}^{n_1} C_{n-n_2-n_1}^{n_3}]}{C_n^{n_1} C_n^{n_2} C_n^{n_3}}$$

To simplify the formulas, we can assume that n_2 is equal to n_3, which is reasonable for analysis purpose because A and B have equal status and behave similarly. Then we have

$$g_1 = \frac{C_n^{n_1} C_{n-n_1}^{n_2} C_{n-n_1-n_2}^{n_3}}{C_n^{n_1} C_n^{n_2} C_n^{n_3}} = \frac{C_{n-n_1}^{n_2} C_{n-n_1-n_2}^{n_2}}{(C_n^{n_2})^2}$$

$$g_2 = g_3 = \frac{C_{n-n_2}^{n_1} [C_{n-n_2}^{n_2} - C_{n-n_2-n_1}^{n_2}]}{C_n^{n_1} C_n^{n_2}}$$

Finally, we have

$$p_3 = p(d_3) = g_1 + g_2 + g_3 \tag{3.7}$$

To calculate the final value of p_3, we can approximate the values of g_1, g_2, and g_3 by using the Sterling formula and *log* computation as suggested in the derivation of p_2.

3.3.2 Application Characteristics and the Unified Formula

In this section, we explore the performance and the implications of IDF to specific applications. In particular, we relate the failure rate of the top-layer detection (the

values of p_1, p_2, and p_3) to real applications by deriving a unified formula. We then show how to adjust two parameters in the unified formula to reflect a particular application's user distribution, as well as usage pattern. In this way, a practitioner can adjust IDF protocol parameters to optimize the IDF performance for his or her particular applications.

As discussed in Section 3.3.1, to derive a unified formula for IDF as a whole, P_{succ}, thus P, has to be derived (see formula 3.4 in Section 3.3.1). In turn, we have to derive h_i ($i = 1, 2, 3$) first (h_i is the probability that event d_i occurs).

This analysis is conducted in three steps. In the first step, we assume that all nodes have the same probability of issuing the next update request for a particular file *f*, which implies that the updating operation is truly random and uniform. In the second step, we relax this assumption and consider different updating patterns. In the third and last step, we derive a unified formula to reflect the characteristics of particular applications.

Finally, we discuss the performance and the implications of IDF to specific applications in hope of helping practitioners choose the best IDF parameters to suit their particular needs.

Random and uniform updates

In step 1, we assume that all nodes have an equal probability of updating,thus the probability of a writer coming from the top layer is decided by the relative size of the top layer. For the same reason, the probability of a writer coming from the bottom layer is decided by the relative size of the bottom layer.

Now we can evaluate h_i ($i = 1, 2, 3$) as follows:

1. Event d_1 means that the two concurrent writers A and B are both from the top layer. In this case, we have

$$h_1 = \frac{C_{n_1}^2}{C_n^2}$$

2. Event d_2 means that one of the two concurrent writers comes from the top layer, while the other is from bottom layer. In this case, we have

$$h_2 = \frac{C_{n_1}^1 C_{n-n_1}^1}{C_n^2}$$

3. Event d_3 means that the two concurrent writers are both from the bottom layer. In this case, we have

$$h_3 = \frac{C_{n-n_1}^2}{C_n^2}$$

Non-uniform updates between two layers

In this step, we relax the random and uniform assumption made in Step 1 by assuming that the updating frequency of the nodes in the top layer is F_1 and that of the nodes in the bottom layer is F_2. However, it is implied that within each layer, all nodes are equally likely to issue update requests for a given file.

In practice, if F_1 is larger than F_2, it means that nodes in the top layer have more update requests than the nodes in the bottom layer, which is the case for applications like online gaming in which active players keep playing the game for a long period of time (thus they will form a top layer and keep issuing updating requests). If F_2 is larger than F_1 instead, it means that new updating requests are more likely from bottom layer. This is possible because, while the current top layer reflects the history, it is still possible that the future pattern is different from the history. For example, if there is a forum in which its user base keeps changing and number of requests from

new members is larger than that from current members, F_2 would be larger than F_1.

After incorporating the frequencies of updating operations of the top layer and bottom layer nodes, the frequency of event d_1 can be expressed as $F_1^2 C_{n_1}^2$, the frequency of event d_2 as $F_1 F_2 C_{n_1}^1 C_{n-n_1}^1$, and the frequency of the event d_3 as $F_2^2 C_{n-n_1}^2$. Hence, we have

1. The value of h_1 to be calculated as

$$h_1 = \frac{F_1^2 C_{n_1}^2}{F_1^2 C_{n_1}^2 + F_1 F_2 C_{n_1}^1 C_{n-n_1}^1 + F_2^2 C_{n-n_1}^2}$$

2. The value of h_2 to be calculated as

$$h_2 = \frac{F_1 F_2 C_{n_1}^1 C_{n-n_1}^1}{F_1^2 C_{n_1}^2 + F_1 F_2 C_{n_1}^1 C_{n-n_1}^1 + F_2^2 C_{n-n_1}^2}$$

3. The value of h_3 to be calculated as

$$h_3 = \frac{F_2^2 C_{n-n_1}^2}{F_1^2 C_{n_1}^2 + F_1 F_2 C_{n_1}^1 C_{n-n_1}^1 + F_2^2 C_{n-n_1}^2}$$

To normalize the values of F_1 and F_2, We introduce f_1 and f_2, which are defined as

$$f_1 = \frac{F_1}{F_1 + F_2}$$

$$f_2 = \frac{F_2}{F_1 + F_2} = 1 - f_1$$

After the normalization, both f_1 and f_2 are in the range of (0, 1). Then h_1, h_2, and h_3 can be expressed as follows

$$h_1 = \frac{f_1^2 C_{n_1}^2}{f_1^2 C_{n_1}^2 + f_1 f_2 C_{n_1}^1 C_{n-n_1}^1 + f_2^2 C_{n-n_1}^2}$$

$$h_2 = \frac{f_1 f_2 C_{n_1}^1 C_{n-n_1}^1}{f_1^2 C_{n_1}^2 + f_1 f_2 C_{n_1}^1 C_{n-n_1}^1 + f_2^2 C_{n-n_1}^2}$$

$$h_3 = \frac{f_2^2 C_{n-n_1}^2}{f_1^2 C_{n_1}^2 + f_1 f_2 C_{n_1}^1 C_{n-n_1}^1 + f_2^2 C_{n-n_1}^2}$$

The unified formula

We can see that, if f_1 equals to f_2, then the values of h_1, h_2, and h_3 are exactly the same as those derived in Step 1. In short, the formulas of step 1 are special cases of the formulas we derived in Step 2.

Now we simplify these expressions and take two important characteristics of real applications, namely, user distribution and usage pattern, into consideration. The two aspects are based on the observation from inside IDF and outside IDF, respectively. The meanings of the two characteristics, the justification of their inclusion, and their relationships with the parameter of the formula for h_i (i = 1, 2, 3) are summarized as follows.

- **User distribution:** This application characteristic reflects the distribution of the active writers, *i.e.*, how many users are active writers in the system. The inclusion of this application characteristic is due to the inherent two-layer architecture of IDF. As the single important characteristic of IDF, it defines the performance of IDF from inside. User distribution directly affects the size of the top layer, namely, the n_1 value.

- **Usage pattern:** This application characteristic reflects where the new updates are most likely coming from. For example, if the updating operations are highly

concentrated on a small group of dedicated users, then most new updates would come from the top layer; otherwise, more updates would come from the bottom layer. This application characteristic is from outside IDF and its inclusion is due to the importance of usage patterns in data sharing systems. In e-commerce, for example, the usage pattern is the main focus based on which the system performance can be optimized [101]. In the case of IDF, usage pattern determines the value of f_1, as well as f_2 because $f_2 = 1 - f_1$.

To formally represent the two application characteristics in the formulas, we do the following substitutions. First, we use α to substitute $\frac{n_1}{n}$, resulting in n_1 in the formulas being represented by $n\alpha$. Second, we use β to substitute f_1 and (1 - β) to substitute f_2. Please note that now both α and β are in the range of (0, 1). Given the total number of nodes in the system–the value of n–the performance of IDF for different applications can be obtained by adjusting the α and the β values.

Now we can finally represent h_1, h_2, and h_3 as

$$h1 = \frac{\beta^2 C_{n\alpha}^2}{\beta^2 C_{n\alpha}^2 + \beta(1-\beta) C_{n\alpha}^1 C_{n-n\alpha}^1 + (1-\beta)^2 C_{n-n\alpha}^2} \tag{3.8}$$

$$h2 = \frac{\beta(1-\beta) C_{n\alpha}^1 C_{n-n\alpha}^1}{\beta^2 C_{n\alpha}^2 + \beta(1-\beta) C_{n\alpha}^1 C_{n-n\alpha}^1 + (1-\beta)^2 C_{n-n\alpha}^2} \tag{3.9}$$

$$h3 = \frac{(1-\beta)^2 C_{n-n\alpha}^2}{\beta^2 C_{n\alpha}^2 + \beta(1-\beta) C_{n\alpha}^1 C_{n-n\alpha}^1 + (1-\beta)^2 C_{n-n\alpha}^2} \tag{3.10}$$

With the expressions for h_1, h_2, and h_3, we can obtain a unified formula to evaluate the performance of IDF based on formula 3.4 in Section 3.3.1. This unified formula can reflect the different characteristics of particular applications by adjusting the values of α and β, which are in the range of (0, 1). Based on this unified formula,

the performance of IDF in the context of a wide range of applications can be derived. The results and how they can be used by practitioners to tune IDF are discussed below.

3.3.3 Results and Their Implications

In this section, we show results from the analytical model and offer insights into how much benefit particular applications can gain from adopting IDF and, more importantly, how a practitioner can tune the IDF's parameters to tailor it for his or her particular needs.

First of all, a set of results based on the formula 3.4, where p_i (i = 1, 2, 3) are defined by formulas 3.5 to 3.7 in Section 3.3.1; h_i (i = 1, 2, 3) are defined by formulas 3.8 to 3.10 in Section 3.3.2, are summarized in Table 3.2 that lists several important variables including the detection success rate. From this table, we can see that the lowest success rate is 91.4% while in the vast majority cases the rate is more than 98%, which is very encouraging because it states that the top layer is indeed able to detect most inconsistencies.

Now we take a closer look at the impact of the parameters on the performance. First, intuitively, the larger the values of α and n_2 are, the higher success rate of detection should be achieved. The results in Table 3.2 are consistent with this intuition. Second, the impact of the value of β is more complex because it increases the values of h_1 and h_2, but decreases that of h_3. From Table 3.2, we can see that the success rate decreases when β increases. However, at some point, especially when β is close to 1, success rate of detection should start to increase because, when β is close to 1, most writers are from the top layer and the success rate of those detections would be 1. To validate this hypothesis, we collect another set of results as shown in Table 3.3 that clearly shows that the success rate start to increase when β reaches 0.91.

Set	α	n_1	n_2	β	p_2	p_3	h_1	h_2	h_3	P	P_{succ}
0	0.02	20	50	0.2	35.487	4.252	0.002	1.011	98.986	4.568	95.432
1				0.5	35.487	4.252	0.038	3.924	96.038	5.476	94.524
2				0.8	35.487	4.252	0.542	13.971	85.487	8.593	91.417
3			80	0.2	18.556	0.032	0.002	1.011	98.986	0.220	99.780
4				0.5	18.556	0.032	0.038	3.924	96.038	0.759	99.241
5				0.8	18.556	0.032	0.542	13.971	85.487	2.620	97.380
6	0.05	50	50	0.2	7.198	1.004	0.017	2.566	97.417	1.163	98.837
7				0.5	7.198	1.004	0.245	9.510	90.245	1.590	98.410
8				0.8	7.198	1.004	2.968	28.772	68.260	2.756	97.244
9			80	0.2	1.385	0.003	0.017	2.566	97.417	0.038	99.962
10				0.5	1.385	0.003	0.245	9.510	90.245	0.134	99.864
11				0.8	1.385	0.003	2.968	28.772	68.260	0.400	99.600
12	0.1	100	50	0.2	0.448	0.064	0.072	5.265	94.663	0.084	99.916
13				0.5	0.448	0.064	0.991	18.018	80.991	0.133	99.867
14				0.8	0.448	0.064	9.387	42.667	47.947	0.222	99.778
15			80	0.2	0.015	0.000	0.072	5.265	94.663	0.001	99.999
16				0.5	0.015	0.000	0.991	18.018	80.991	0.003	99.997
17				0.8	0.015	0.000	9.387	42.667	47.947	0.006	99.994

Table 3.2: Success rate (in percentage) when n is 1000 with 18 sets of parameter values

Set	α	$n_1 = n \times \alpha$	n_2	β	P_{succ}
1	0.1	100	50	0.8	99.778
2				0.9	99.759
3				0.91	99.762
4				0.95	99.871

Table 3.3: Investigate the impact of β

Now, we investigate how the parameters of IDF can affect the system performance for different applications. Among the four parameters we discussed–n_1, n_2, α, and β–n_1 and α correlate ($n_1 = n \times a$). Then, β is the relative activity of top layer nodes when compared to bottom layer nodes and is entirely application-dependent, so it is not adjustable. However, we do not expect this inability to affect the overall performance of IDF seriously because, as shown in Table 3.2, comparing with α and n_2, β does not impact the performance of IDF significantly.

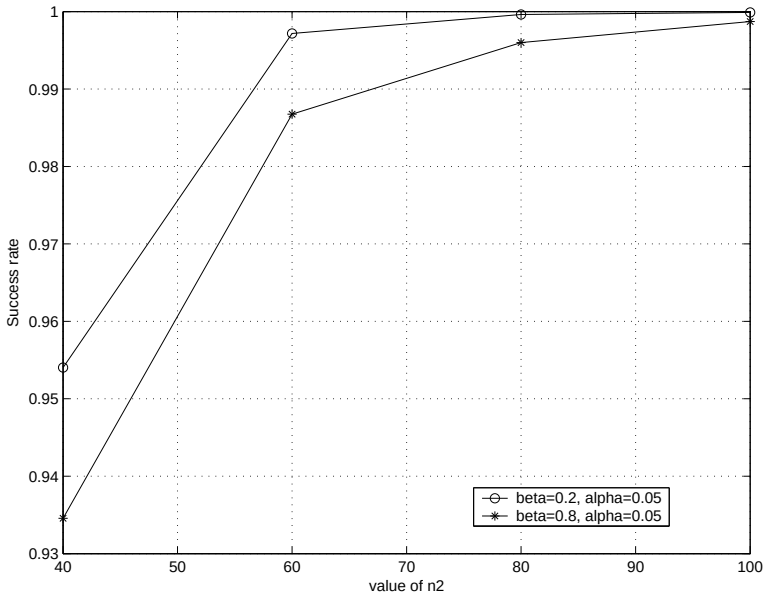
The remaining two parameters are adjustable. α is adjustable because the protocol can lower the threshold of the required temperature for the top layer participants, which in turn can increase the size of the top layer; n_2 is also adjustable because it is a parameter in IDF. The larger the values of α and n_2 are, the higher the success rate is.

For all the discussions, we choose two values of β, 0.8 and 0.2, and investigate the impact of the two parameters, α and n_2, as follows. We first fix α and β, then calculate P_{succ} based on different values of n_2. The result is shown in Figure 3.9(a). Two (α, β) pairs and four different n_2 values are used in this calculation.

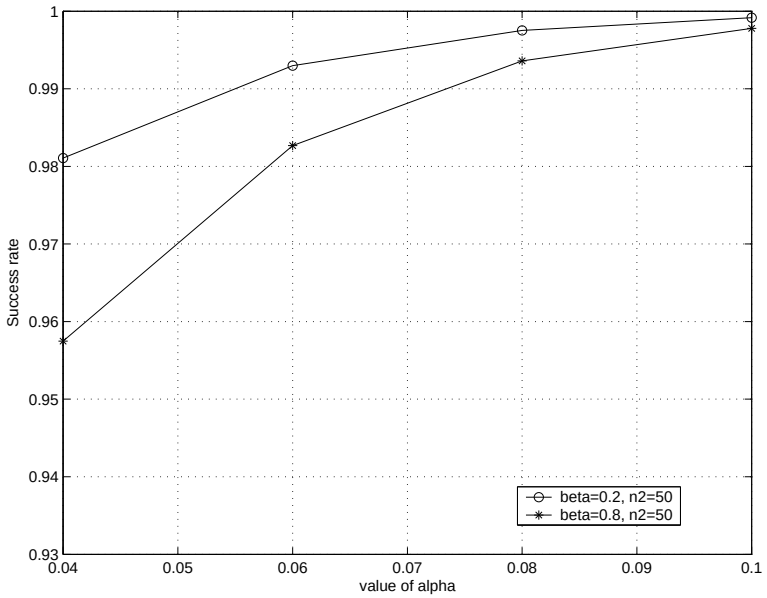
Then we fix n_2 and β and calculate P_{succ} based on different values of n_1. The result is shown in Figure 3.9(b). Two (n_2, β) pairs and four different α values are used in this calculation.

From the two figures, we can see that the success rate increases with the n_1 and n_2 values. This makes sense because, with the increased top layer size (n_1) and the limited broadcasting size of a new writer (n_2), there are more chances for concurrent writers to meet. But this comes at a cost as well: slower detection delay due to the larger top layer size (in the case of increased value of α) and increased network bandwidth overhead (in the case of increased value of n_2).

Based on this information, practitioners can fine tune the parameters as follows.



(a) Fix β and α



(b) Fix β and n_2

Figure 3.9: Tuning IDF parameters with $n = 1000$

If the system has enough bandwidth available, they can increase the value of n_2; if the response time of IDF is better than the required value, it is also possible for them to increase the value of α. While seemingly simple, this suggestion can at least help practitioners tune IDF in the best way to achieve the best performance of their particular applications.

3.4 Summary

In this chapter, we have proposed the overview of IDF (Inconsistency Detection Framework) and IDF's inconsistency detection module. As a framework, IDF provides adaptive consistency control by detecting inconsistencies among replicas in a timely manner and resolving the detected inconsistencies only when needed. IDF has two main components: an efficient universal inconsistency detection mechanism and IDEA, the adaptive consistency control protocol built on top of the detection mechanism.

The universal inconsistency detection mechanism uses a two-layer infrastructure in which the top layer includes all the active writers and the bottom layer includes the rest of the participants. Due to the relatively small size of the top layer, IDF is able to detect inconsistencies efficiently in the top layer, where most inconsistencies are originated. The detection mechanism is evaluated from three aspects: speed of top layer detection, maintenance cost, and success rate of top layer detection. The evaluation results show that: (1) the detection in the top layer is much faster than that in the bottom layer; (2) the detection mechanism incurs minimal communication overhead and, since communication overhead is the main aspect of membership management cost, the minimal communication overhead removes costly membership management cost associated with conventional consistency control approaches; and (3) the mech-

anism has a very high probability (over 95%) to successfully detect inconsistencies among the top layer nodes in a variety of scenarios.

Chapter 4

IDEA: An Infrastructure for Detection-based Adaptive Consistency Control for IDF

In the previous chapter, we have presented the IDF and its inconsistency detection module. In this chapter, we present the adaptive consistency control module, IDEA (an Infrastructure for DEtection-based Adaptive consistency control), which is built on top of the detection mechanism and interacts with different applications. IDEA achieves both the adaptability and high-performance goals to complete the design of IDF.

4.1 Motivation and The Role of IDEA in IDF

So far, we have explained the underlying detection infrastructure of IDF. However, the detection mechanism itself does not interact with applications directly because it does not understand applications' semantics. For example, given a white board

application, there is no apparent way by which the detection mechanism can figure out what the application's expectation for consistency level is. Also, as mentioned before, IDF promises great adaptivity to a set of applications and there is no mechanism in IDF thus far that provides this kind of interface.

More formally, the term *adaptability* has two meanings here. First, the system should be able to adjust its consistency level on the fly, as opposed to a predefined and fixed consistency level. Second, the end users should have the control on how to adjust the consistency level (or requirement) on the fly. That is, the users first give a hint about what kind of consistency level (or requirement) they prefer and then adjust that preference when the need arises. The rationale behind this is that, although the users themselves may know what they want, they may not be good at expressing it in concrete and/or quantitative terms. Instead, they know whether a given consistency level is sufficient or not only when they see it.

To achieve adaptability, IDEA adjusts the consistency level on the fly through its interaction with users. Upon the detection of inconsistencies, IDEA resolves them if the current consistency level does not satisfy applications' requirement; otherwise, IDEA will not resolve the inconsistencies except when the system is lightly loaded. The advantages of this approach are two folds: it can adjust the consistency level on the fly by resolving inconsistencies on demand; and more importantly, it gives the users the ability to control their perceived consistency level. It is worth mentioning that, because higher consistency level necessarily means longer response time for the user due to higher consistency maintenance cost, we do not expect users to abuse the system by overstating their consistency requirement because that will ultimately hurt them (with longer response time).

Beyond the adaptive interface, IDEA also has to achieve high performance. That is, to find inconsistencies and, when necessary, to resolve them in a timely manner.

This is crucial because slower resolution can lead to poor QoS. IDEA utilizes the two-layer infrastructure deployed in the efficient inconsistency detection mechanism described in Chapter 3 to achieve this goal. As shown in the evaluation section later, this ability to capture most inconsistencies in a small top layer is crucial to guarantee the efficiency of the resolution.

4.2 Overview of IDEA

IDEA is assumed to work with a general distributed file system that handles the ordinary read/write operations. The general distributed file system is assumed to ensure the correctness of read/write functionalities, while IDEA detects inconsistencies among nodes and resolves them based on applications' changing requirements. That is, IDEA provides consistency control to this general file system.

Figure 4.1 illustrates the vision of IDEA. IDEA is deployed in the middleware level and consulted by applications on different nodes when the latter access files for the purpose of consistency control. Upon a request for a file by a user of an application, IDEA retrieves a copy of the file from the underlying replication-based system and returns it to the application. At the same time, IDEA derives a consistency level for the returned replica. Then IDEA checks whether the consistency level is acceptable based on either users' predefined tolerance levels or the interaction with users in real time. If the consistency level is acceptable, IDEA does nothing; otherwise, IDEA will resolve this inconsistency.

Users can communicate with IDEA about why the current consistency level is not sufficient and IDEA will learn from this to prevent annoying users again. More specifically, upon initiating an application, users have the option to predefine or hint on their acceptable consistency levels. Or they could just respond to IDEA

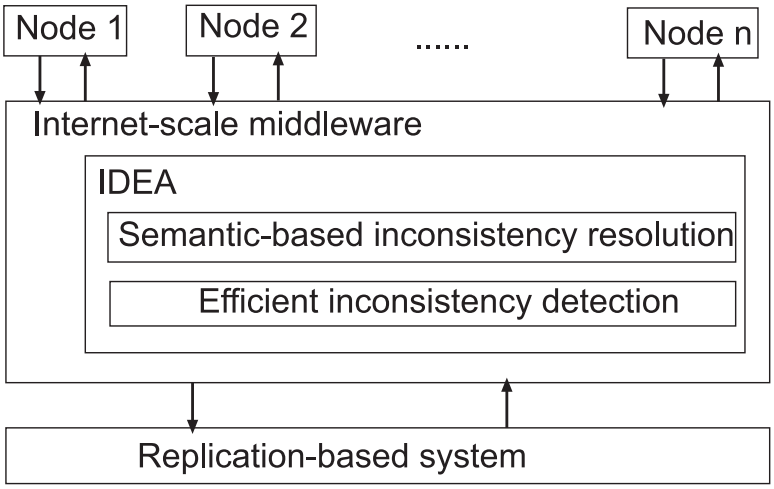


Figure 4.1: The vision of IDEA

interactively. If there is an initial hint level, we denote it as L_1. Upon receiving the response, IDEA will not invoke the inconsistency resolution module unless the consistency level is below L_1. When a user is not satisfied with the result, IDEA will increase the consistency level by $\triangle$. $L_1 + \triangle$ will then become the new desired consistency level for the user and IDEA will keep the application's consistency above this new level to avoid annoying the user again in the future. In a similar manner, a user can demand a lower consistency level for better responsiveness. This way, IDEA makes the consistency control adaptive and gives users great flexibility to adjust consistency level themselves.

4.3 Targeted Applications

IDEA is designed to support a wide range of distributed, replication-based applications that are willing to trade certain consistency requirement for the ability to scale to an Internet-scale distributed system. In particular, we consider distributed online collaboration applications in this dissertation.

Previous research has indicated that there are two types of distributed online

collaborations: synchronous collaboration in which the participants appear online at the same time and asynchronous collaboration in which the participants do not necessarily appear online at the same time [17].

In this section, we list two representative applications–one is synchronous collaboration and the other is asynchronous collaboration–and discuss their working flow. How consistency levels can be measured and how adaptability is achieved through IDEA for both applications will be discussed in Section 4.5 after the design of IDEA is presented in Section 4.4.

4.3.1 Distributed White Board System

A distributed white board system allows participants to draw or write on the same virtual white board so that they can interact and collaborate with one another while working on a single project or task. Because all participants usually appear online at the same time, this is a synchronous application. We assume that, in a distributed white board system, each user/participant has a white board system locally. Thus, due to network delays, the message a user reads may not be the most up-to-date information on other users' view (or vice versa), which results in inconsistency.

4.3.2 Airline Ticket Booking System

An airline ticket booking system is an example of e-business applications. Because not all clients are necessarily to appear online at the same time, this is an asynchronous application. In this system, we assume the existence of several booking servers, which are distributed in a wide area environment. To improve the efficiency of booking and avoid underselling, each server tracks its booking record independently. However, this may cause inconsistency–one server does not necessarily know the booking record of other servers in a timely manner–and hence overselling. Certainly, both underselling

and overselling will hurt the company economically. From the company's point of view, there is a clear trade-off between the efficiency of booking–to avoid underselling- and the chance of overselling. Essentially, overselling is fine as long as the amount is within a certain range, which can be treated as a cost of avoiding underselling.

4.4 The Design of IDEA

There are two important features of IDEA: first, its ability to adaptively resolve the detected inconsistency based on applications' changing requirements and users' preference; and second, the high performance of inconsistency resolution in terms of delay.

In this section, we first review and highlight the essence of the two-layer (top/bottom layer) infrastructure, described in Chapter 3, that is also used by the detection mechanism, and the workflow of the IDEA protocol is then presented. Because the detection mechanism described in Chapter 3 does not quantitatively measure how inconsistent a conflict is and thus cannot tell the system whether a detected inconsistency is acceptable or not, IDEA modifies the original detection messages by incorporating a single formula to quantify consistency level based on the information provided by the return value of the detection API. This formula is applicable to a variety of applications and will be presented.

In terms of inconsistency resolution, we discuss two mechanisms, namely, background and active resolution, that serve different purposes: the former improves consistency in the system from periodically behind the scene while the latter is triggered when a user explicitly requests a resolution operation. Different applications naturally may assume different meanings of adaptability, an issue that is discussed afterwards. Finally, we discuss the interface provided by IDEA for application devel-

opers to configure IDEA.

4.4.1 The Two-Layer Infrastructure

As in the detection mechanism, IDEA also utilizes the two-layer (top/bottom layer) infrastructure to resolve inconsistency for each shared file or object. Due to the top-layer's relatively small size, it is much faster to detect and resolve inconsistency among its members than it is for the whole network. In the background, however, IDEA always visits the bottom layer, which covers all the nodes in the network, to catch the possible, although somewhat unlikely, missed detections or resolutions by the top layer.

4.4.2 Overview of the IDEA Protocol

An overview of the IDEA protocol is depicted in Figure 4.2. From the figure, we can see that the IDEA protocol is triggered by two operations: write and certain read operations. The write operation, such as issuing an update in a white board, triggers the IDEA protocol because it is essentially an update operation that will surely cause inconsistency among replicas. For read operations, IDEA is triggered when a reader tries to retrieve a new file (such as a new snapshot of a white board) because, in this case, the user needs to make sure that the file retrieved is sufficiently consistent for the user's purpose. For other reads, IDEA is triggered according to the context: if the file is locally updated frequently, the read will not trigger IDEA; if the file has not been locally updated for a long time and the user is afraid that the file may be inconsistent, IDEA can be triggered.

After IDEA is triggered, it will use the detection mechanism to check the consistency level, represented by a single percentage number, such as 90%. Here, we assume that this number can be obtained appropriately either from interpreting users' view

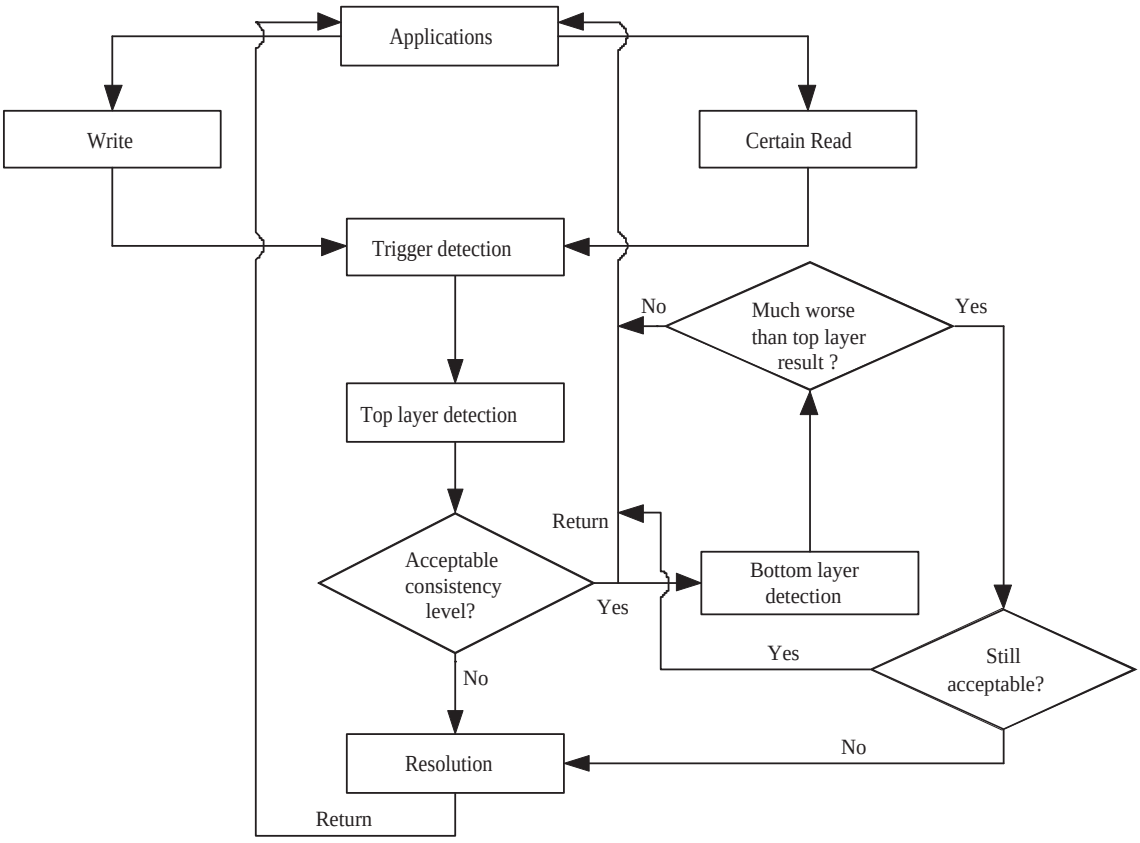


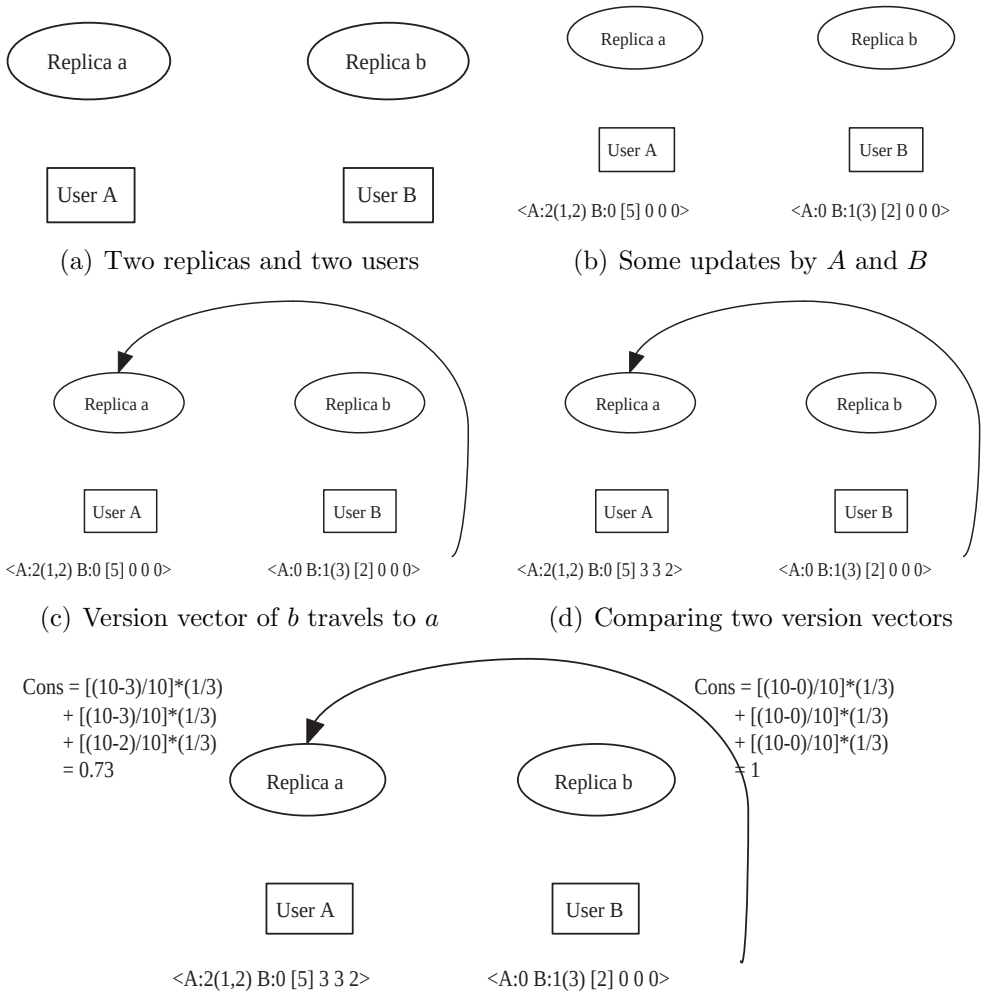
Figure 4.2: The IDEA Protocol

of QoS or by (analysis of) the nature of an application; the mechanism to properly quantify this parameter will be discussed shortly after. After the consistency level is returned, IDEA checks whether it is acceptable based on either users' predefined tolerance levels or through the interaction with users in real time. If the consistency level is acceptable, IDEA does nothing; otherwise, IDEA will resolve this inconsistency upon the request from the user. As discussed in Section 4.3.1, users can communicate with IDEA about why the current consistency level is not sufficient and IDEA will learn from this to prevent annoying users again.

For efficient inconsistency detection, the inconsistency is initially detected only among the top-layer nodes to improve the response time. Hence, the detected inconsistency level may not be accurate because the nodes in the bottom layer can cause inconsistencies too, albeit rather infrequently. To cope with this issue, we deploy a rollback mechanism. More specifically, IDEA lets users continue their work when they indicate that the initially returned consistency level (from top layer nodes) is acceptable. In the background, however, IDEA continues to detect inconsistency in the bottom layer and returns a new value. If the new value is sufficiently close to the one obtained from the top layer, IDEA keeps silent; otherwise, IDEA alerts the user about the discrepancy and resolves the inconsistency if the users so demand. In this figure, background and active inconsistency resolution schemes are collectively represented by the resolution module that is at the bottom of the figure.

4.4.3 Quantifications of Consistency Level: A Scenario

Basically, the inconsistencies among nodes is quantitatively measured by a metric adopted from the TACT measurement [106] where a <*numerical error, order error, staleness*> triple is used to indicate how inconsistent a replica is. Now we use an example to illustrate how we use an extended version vector developed by us to



(a) Two replicas and two users

(b) Some updates by A and B

(c) Version vector of b travels to a

(d) Comparing two version vectors

(e) Calculate consistency level for replica a and b

Figure 4.3: An example of consistency level quantification

derive the consistency level and how it can be applied to a variety of applications.

First of all, we assume two replicas (a and b) and two active users/writers (A and B), as depicted in Figure 4.3(a). User A resides in replica a and user B in replica b.

Figure 4.3(b) shows the extended version vectors of the above users, having participated some updating activities. While the original version vector only tells us the number of updates from each writers (in the form of $<A$:2 B:0$>$, for example), our extended version vectors is capable of convey more information as explained below

using replica a's version vector as an example.

First, the extended version vector has time stamps associated with each update, such as $< A : 2(1, 2) >$, implying that the two updates from A happen at time points of 1 and 2, respectively. To make the timestamp comparable among different sites, we assume that the gap among time clocks of participating nodes in the system is within seconds, which is small enough to neglect in a globally distributed system. Practically, there are two mechanisms to achieve this precision. First, the system can run a globally synchronizing clock algorithm, such as that proposed in [51]. If it is too troublesome to run such a clock synchronizing algorithm, another choice is to let each node to keep their time accurate by synchronizing with a time sever using NTP (Network Time Protocol) [66], which can be easily achieved in both Windows or Linux operating systems by enabling the corresponding modules [66].

Second, as we can see, there is a numerical value in the square brackets (the <[5]> column in the extended version vector). We use this value to represent some critical metadata of applications to characterize the difference among different versions, as explained below. In the case of distributed white board, for example, the metadata can be the sum of the ASCII value of the last several updates; in an airline ticket booking, it can be the total sale revenue. These metadata can give a quick sense of what the effect of the conflict would be, which is easier to understand in the airline booking example–the data tells the total sales revenue that has significant business value.

Third and finally, the <*numerical error, order error, staleness*> triple is attached at the end to conclude the extended version vector. The numerical error is deduced by comparing the value of critical data; the order error counts for the difference between numbers of updates (an example of calculation will be given shortly); and staleness error is calculated from the time stamps (an example will be given later). In Figure

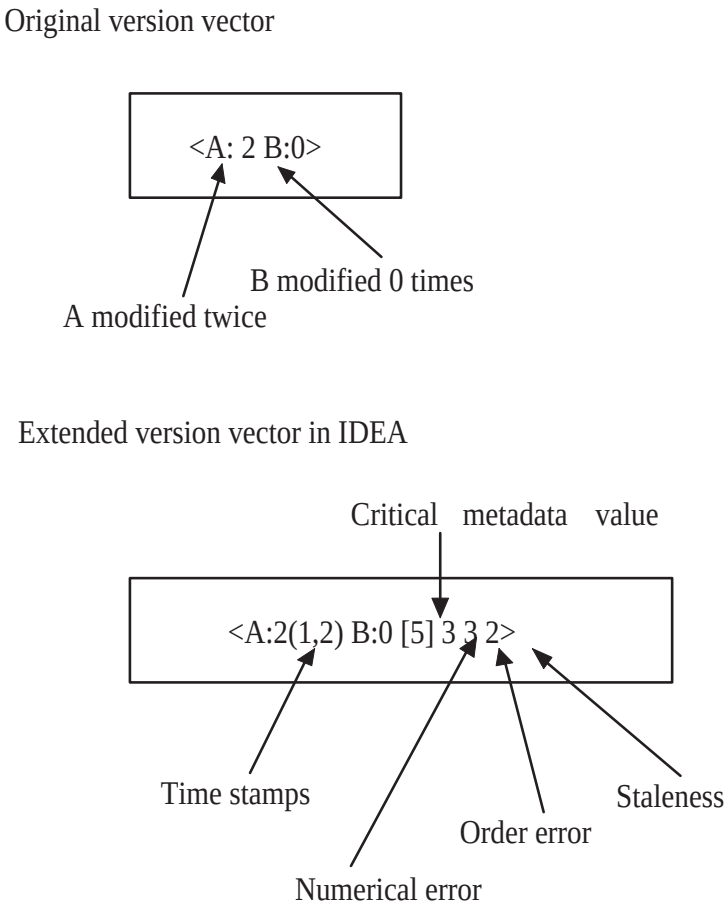


Figure 4.4: Comparison between original version vector and the extended version vector in IDEA

4.3(b), because replica a is not aware of any conflict in the system, all the errors are set to zero.

Figure 4.4 depicts the difference between the original version vector and the extended one used in IDEA. Because IDEA uses this extended version vector instead of the original one, we will simply use the term "version vector" to denote the extended version vector for brevity in the rest of this dissertation.

Suppose now that the detection process is started and the version vector of a replica is traversed from replica b to replica a, as shown in Figure 4.3(c).

Then, after comparing with that of replica a, the version vector of replica a and

b will be changed and the new ones are shown in Figure 4.3(d). The calculation is carried out as follows.

First, IDEA derives a *reference consistent state* which is the state chosen by IDEA that is regarded as the basis for consistency level calculation. As we will show later, there are several ways to derive the reference consistent state. For now, let's assume that the replica with higher ID value becomes the reference consistent state, which means that, if the version vector from a and that from b conflict with each other, IDEA will choose b ($b > a$) as the reference consistent state and then use it to calculate a and b's consistency levels.

But first, we need to use b (the reference consistent state) to calculate the $<$*numerical error, order error, staleness*$>$ triple that will be used to derive a numerical consistency level as follows: replica a's final value of its metadata has a gap of 3 with that of b (the reference one), so the numerical error is 3; replica a misses one update and has two extra ones, so the order error is 3 too; and finally, the last time point when a is consistent is time 1, which has a gap of 2 with the most recent update at b (time 3), so the staleness is 2. Generally, staleness of one replica is defined as the time difference between the most recent update in the *reference consistent state* and the last time point when it is consistent.

Then, IDEA calculates the consistency level as follows. First, IDEA predefines a maximum value for each member of the triple. For example, if in practice the order error is very unlikely to be larger than 10, then the maximum value for order error can be set as 10. Then IDEA gets input from users and sets weight for the three members respectively. For example, if users treat the three members equally, their weight will be equal and 33.3%. Then the consistency level can be quantified as in Formula 4.1:

$$
\begin{aligned}
Consistency\_level &= \frac{Max\_num - num\_error}{Max\_num} \times num\_weight \\
&+ \frac{Max\_order - order\_error}{Max\_order} \times order\_weight \\
&+ \frac{Max\_staleness - staleness\_error}{Max\_staleness} \times stale\_weight
\end{aligned}
\tag{4.1}
$$

The calculation of consistency level of version vector of replica a and b according to Formula 4.1 are presented in Figure 4.3(e) by assuming that the maximum error for all three metrics are 10.

One may wonder that, if the consistency state is easy to be figured out, why don't we resolve it immediately? There are two important reasons: communication overhead for the system and its potential to block updating operations for users. First, if we resolve every conflict, the huge communication cost (*e.g.*, copying remote updates to local sites) will be huge. For this reason, we prefer to defer this resolution whenever possible. Second, once we decide to resolve the inconsistency, all future updates will be blocked until the resolution is finished (to prevent invalid updates that are based on an inconsistent copy). Thus, to improve system's responsiveness, it is preferable not to run the resolution unless the inconsistency is unacceptable.

4.4.4 Quantifications of Consistency Level: Accuracy of the Calculation

It must be noted that the aforementioned calculation of consistency level may not be 100% accurate because it does not include the replicas in the bottom layer. Nonetheless, as explained in the IDEA protocol, inconsistency detection will be carried out in

the bottom layer after the top-layer detection is done. After a certain period of time, the result of the bottom layer will be returned. Then, if the new result is sufficiently close to the one returned from the top layer (e.g., 78% vs. 80%), the top-layer result remains intact; if the results from the two layers are not close enough, the top-layer result needs to be modified and the operations during this period should be rolled back if the new consistency level is not acceptable according to the user's preference that IDEA has learnt so far. To insure correctness, the top-layer and the bottom-layer results should always be compared.

There are two things that we need to point out about this potential rollback operation. First, to avoid annoying users, IDEA will handle the rollback in the background and return the result to the users afterwards. Second, the impact of rollback should not be overstated. According to our analysis in Section 3.3, it is very rare (less than 5% in the majority of a variety of scenarios) that the top layer will leave an inconsistency undetected. Thus, we treat the rollback mechanism as a back-up mechanism and do not expect it to slow down the performance of IDEA.

Due to the potentially large number of nodes in the bottom layer, which covers all nodes in the system, a critical question is how long the detection in the bottom layer would take. Intuitively, the longer the delay, the larger number of states will potentially need to be rolled back, which would cause more overhead and frustrates users more. Currently, we use TTL (Time to Live) to control the traversal of the bottom-layer detection messages, thus bound the delay. Clearly, there is a trade-off between accuracy and responsiveness. We believe that, in an Internet-scale system like Gird, this trade-off is reasonable and necessary. Other mechanisms to tackle this problem certainly exist and we plan to investigate this issue further in the future.

4.4.5 Inconsistency Resolution Mechanisms

Inconsistency resolution is needed when replicas are inconsistent with one another. Given two version vectors u and v from two replicas, the replicas are inconsistent if their version vectors are different and a *reference consistent state* needs to be derived. We denote the method of this derivation as a resolution policy. As defined in [75], two vectors are comparable if and only if $u < v$, $u = v$ or $u > v$. In this case, the resolution is relatively easy: just let the one with the smaller value learn from the one with the larger value by choosing the latter as the *reference consistent state*. Otherwise, they are not comparable with each other. For example, $<A{:}5\ B{:}3>$ is not comparable with $<A{:}3\ B{:}6>$. In this case, the best way to resolve the inconsistency (*i.e.*, choosing the reference consistent state) is not obvious. Here we list three possible resolution policies for cases where updates are not comparable. These policies are briefly described next for illustration purposes only since in practice other policies are also possible.

- **Invalidate both.** In this case, the two conflicting versions are both invalidated and they will roll back to a previous consistent version. In a distributed white board, for example, two simultaneous updates at the same spot can be both cleared to prevent ambiguity and ensure fairness (so that no one is more important than the other).

- **User ID based.** To ensure fairness, each node can be assigned a randomly chosen ID, such as the hash value of their IP address via MD5, which is a commonly used practice in Peer-to-Peer systems [46]. When a conflict arises, the user with the larger ID wins. This approach can be used in both a distributed white board and an airline ticket booking system where certain progress is preferred (if both updates are to be invalidated, no progress can be made in a white-board-based discussion and no ticket will be sold in an airline ticket

booking system). In this case, it is desirable to treat its members equally (ensured by using randomized user IDs).

- **Priority based.** In this policy, different levels of priorities are assigned to users. For example, the supervisor of a company will have a higher priority than ordinary employees. When a conflict arises, the version created by a higher-priority user wins. In a distributed white board, a supervisor can have a higher priority than other employees; in an airline ticket booking system, giving preferred customers, such as those who have traveled the most with this airline, higher priority is a sensible choice.

4.4.6 Background and Active Resolution

Here we discuss two inconsistency resolution mechanisms–background and active resolution–that serve different purposes: while the former improves consistency in the system from time to time, the latter is triggered when a user explicitly requests a resolution operation. Active resolution is needed because we expect that end users will explicitly request an inconsistency to be resolved when it becomes unacceptable. However, if we only resolve inconsistencies when they become unacceptable, it will unavoidably annoy users from time to time whenever the system's consistency deteriorates to a level discernable by the user. Even though IDEA can avoid annoying users by resolving the inconsistency right before it becomes unacceptable, it does not prevent the consistency from continuous deterioration as the application evolves with time. Therefore, IDEA periodically resolves inconsistency in the system to improve the consistency in the background, hence the name background resolution.

One of the salient features of IDEA is its ability to resolve an inconsistency in a timely manner as a result of its efficient two-layer resolution mechanism, which holds

true for both background and active resolutions, and will be evaluated in Section 4.6.

Now we illustrate the process of background resolution, followed by that of active resolution.

The background resolution process is triggered by IDEA periodically to improve the consistency among replicas on a regular and continuous basis without users' intervention. Once this process is started, one replica (chosen by IDEA) in the top layer for a certain file acts as the initiator and collects all the version information of the members in the top layer by sequentially visiting them and then determines a consistent replica, by following the resolution polices discussed in the previously. It then informs all the members of information about the new consistent replica and the members will update their copies by acquiring any missing updates to reflect this change.

Active resolution, unlike the background consistency resolution, is triggered when a user explicitly requests an inconsistency to be resolved. That is, active consistency resolution is a backup of the background consistency resolution and only kicks in when the periodical background consistency resolution fails to satisfy some users' needs.

When active consistency resolution is triggered, the nearest replica (including the user's local copy) takes the responsibility of initiating inconsistency resolution. More specifically, we use a two-phase protocol. First, the initiator sends a request to all the members in the top layer in parallel to call for attention to the upcoming resolution process. Second, only after it gets all positive acknowledgements (*i.e.*, no one else is initiating the same process), it starts the resolution procedures; if someone else has already sent the same request out, they will back-off and retry after a random amount of time. Here, the back-off process is used to suppress redundant resolution process to save bandwidth: in the retry period, if one receives another's notice before it tries, it will simply cancel its own resolution process. When this first phase succeeds, the

same resolution process as the background resolution process will then be triggered.

4.4.7 Adaptive Consistency Control

Different applications naturally have different meanings of adaptability and here we discuss how adaptive consistency control works from an application's point of view. That is, how IDEA caters to application semantics in practice. Here we list three possible application types that can benefit from IDEA and explain how IDEA works for them based on their semantics, respectively.

- **On-demand.** In this scheme, users explicitly request consistency resolution when they are not satisfied with the current consistency level. Otherwise, they depend on the background consistency resolution. One possible application is the distributed white board system in which each newly posted message will contain its consistency level generated by IDEA. Then, when the users feel that the consistency level is unacceptable, they tell IDEA to adjust the weights of the three metrics, or to keep the same weights but boost the overall consistency, or to do both.

- **Hint-based.** This scheme asks users to give hints about their approximate consistency requirements. When a consistency level is derived, IDEA only triggers the active consistency control when the consistency level drops below what was hinted by the user. In this mode, users in a distributed white board system indicate their tolerance levels and IDEA will keep the consistency level above that. However, if users later feel that the pre-set hint level is not high enough, they can communicate with IDEA to change the hint level to a higher one.

- **Fully automatic.** This scheme improves consistency with best effort, by adjusting the frequency of background resolution, under certain constrains. Possi-

ble applications include e-business applications. For example, in an airline ticket booking system, if the consistency overhead constraint to be at or below 20% of available system capacity (to save enough network bandwidth for customers' requests), the frequency of background resolution needs to be adjusted based on the system's current total available capacity. Also, as explained in Section 4.3.2 earlier, such a system should also not cause either too much underselling or too much overselling, which has undesirable economical consequences. To prevent these from happening, IDEA needs to learn the two bounds of the frequency of the background resolution beyond which too much underselling and too much overselling will occur. When IDEA adjusts the frequency of background resolution based on the current system load (for example, to consume less than 20% of the total available bandwidth), the adjustment will be within the two bounds: it will not be above the higher bound in order to prevent underselling and not be under the lower bound to prevent overselling.

4.4.8 IDEA APIs

IDEA has two interfaces, to the developers and to the end users respectively, that serve different purposes. On the one hand, the developer interface lets developers use IDEA to serve their particular applications, be it distributed white board or others. On the other hand, the end user interface lets users interact with IDEA during the runtime environment of IDEA. For services other than consistency control, end users are supposed to interact with applications directly. The difference between the two interfaces is illustrated in Figure 4.5 as follows.

Because we have discussed IDEA's interface to end users extensively in previous sections, we devote this sub-section to discussing IDEA's interface to application developers. This interface, in the form of APIs (Application Programming Interface),

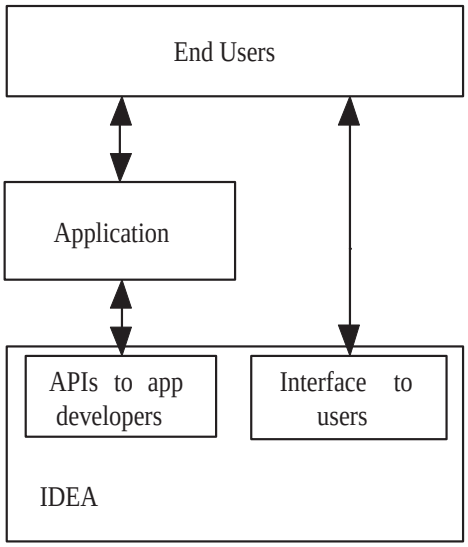


Figure 4.5: Two interfaces of IDEA

Functions
set_consistency_metric (a, b, c): cast applications to IDEA infrastructure
set_weight (a, b, c): set weights for the three consistency metrics
set_resolution (r): set the resolution strategy
set_hint (h): set the initial hint level
demand_active_resolution ( ): call for active inconsistency resolution
set_background_freq (f): set the frequency of background inconsistency resolution

Table 4.1: APIs for configuring IDEA

is for application developers to interact with IDEA. Currently supported APIs are listed in Table 4.1 and we explain how they are used as follows.

- **Casting applications to IDEA's consistency metric.** We use the triple consistency<*numerical error, order error, staleness*> as a generic form to derive a consistency level, and the system administrators need to explicitly define the meaning of the three metrics in an application's context. For example, they may need to define the granularity of an application's objects, define the kind of error to be considered, *etc.* This is to cast applications to the IDEA infrastructure through the *set_consistency_metric* function.

- **Setting weights of metrics.** This is done through the *set_weight* function. To derive a single numeric value for the consistency level, the system administrators need to define the weight of each metric in the consistency quantification using a triple weight<*numerical error, order error, staleness*>. For example, to treat each metric equally, they can indicate weight$< 0.33, 0.33, 0.33 >$. If one metric, such as order error, is not suitable for one particular application, it can be signified by indicating its weight as 0, such as weight$< 0.4, 0, 0.6 >$ in this case.

- **Setting resolution strategy.** This is done through the *set_resolution* function. The parameter is a single integer number that indicates the preferred inconsistency resolution policy. Suppose there are four policies, as explained in Section 4.4.5, then the possible value will be 1, 2, 3, or 4.

- **Setting hint for hint-based applications.** This is done through the *set_hint* function and is only used in hint-based applications, one type of applications discussed in Section 4.4.7. A valid parameter should be between 0 and 1, such as 0.85. In particular, by setting this value to 0, the administrator indicates that this is not a hint-based system; setting this value to 1 means that the user does not tolerate any inconsistency.

- **Demanding active inconsistency resolution.** Applications use function *demand_active_resolution* to explicitly ask IDEA to actively resolve the conflicts through a resolution strategy defined through the *resolution strategy* API.

- **Setting frequency for background resolution.** Applications set the frequency for background resolution performed by IDEA through the function of *set_background_freq*.

Characteristics	While Board	Airline Ticket Booking
Collaboration type	Synchronous	Asynchronous
Important factor	Order preservation	Order preservation and numerical error
Adaptive control scheme	Hint-based	Fully automatic

Table 4.2: Characteristics of two targeted applications

4.5 Case Studies of Applying IDEA to Applications

In this section, we discuss how consistency levels can be measured and how adaptability is achieved through IDEA for a distributed white board system and an airline ticket booking system. The main characteristics of these two applications are summarized in Table 4.2.

4.5.1 Distributed White Board System

We first discuss how consistency level is measured. As stated in the design section, IDEA uses the <*numerical error, order error, staleness*> triple to indicate the consistency level. In the case of a distributed white board system, the numerical error parameter denotes certain metadata gap between two replicas (such as the sum of the ASCII value of the last several updates); the order error parameter measures the degree by which updates appear out of order in one node, which in white board is the most confusing for users because these updates make sense only when they are read in order; finally, the staleness parameter represents the time gap between now and the last time a replica is consistent.

It is worth mentioning that staleness is different from response time, a performance metric to be used later to evaluate IDEA. The key difference is that staleness denotes how long the replica has been in an inconsistent state, while response time is the

transmission delay for a *requested consistent view* (*i.e.*, content of a shared file/object) to arrive. So, even if staleness equals a long delay, the response time of IDEA as a whole can still be minimized because they evaluate different behaviors.

Given the triple values, the consistency level can be quantified, as in the formula 4.1. By adjusting the weight given to each member of the triple, IDEA can reflect applications' different characteristics. For example, users in a white board scenario may prefer more order preservation (all messages appear in the same order at different nodes) than staleness, so IDEA will give more weight to order error, such as 0.7 to order error and 0.1 to staleness.

After discussing how to evaluate consistency level, now we briefly describe about how users can interact with IDEA to achieve adaptability.

First of all, with IDEA, the inconsistency among different sites can be detected and IDEA derives a consistency level for a given replica in a timely manner. Then IDEA checks whether the consistency level is acceptable based on either users' predefined tolerance levels or the interaction with users in real time. If the participant considers the current consistency level tolerable, no further action is needed. Otherwise (for example, the order preservation is poor and annoying), the user can explicitly ask the inconsistency to be resolved.

There are three ways through which users can communicate with IDEA about why an consistency level is unacceptable and how they want it to be improved: changing the weights of individual parameter, boosting overall consistency level without changing individual weights, or both. More concretely, the users change the weight when they feel frustrated about one particular metric, but not others. For example, they may feel that order preservation is fine but the staleness is too high. They can then ask to increase the weight for staleness. Alternatively, they can simply ask IDEA to boost the overall consistency level if they are satisfied with the assignment of weights.

Finally, the users can ask IDEA to do the two at the same time: first changing the weight assignment and then boosting the overall consistency level.

While it is easy to see how user can tell poor order preservation, we have to say something about staleness because users cannot detect high staleness without knowing when it was the last time their replicas are consistent. In practice, we can let the system record the time when a replica is consistent (this can be done because IDEA knows when a replica is consistent or not). Then, with this reference point, the system can let the users know how long their replicas have been in inconsistent states.

If users demand inconsistency resolution, IDEA will do so by returning a consistent view afterwards. As explained in Section 4.3, in the same time, IDEA will learn the new acceptable consistency level and try to avoid annoying users again by keeping the consistency level above this new one in the future.

Overall, by periodically detecting inconsistency with sufficient frequency behind the scene, but only resolving them when users demand, IDEA keeps the system running smoothly without interruption to the application. However, when the need arises, IDEA is able to bring the consistency level back to acceptable states in a timely manner as well as to dynamically adapt the consistency measurements parameters to prevent annoying users again.

4.5.2 Airline Ticket Booking System

As in a white board scenario, the consistency level of an airline ticket booking system can be measured by the weighted sum of the triple values. In airline ticket online booking, however, order preservation may not be the sole focus because staleness and numerical error can potentially affect profits too. In this scenario, an order error means a wrong sequence of the booking order from users, which can cause

conflicts when the order matters, such as assigning seats when clients purchase tickets. Staleness denotes the delay of a booking record that appears on other nodes, which causes conflict because a replica may authorize a sale without the full knowledge. And numerical error represents the gap of the system's overall sales revenue on different web server. Hence, the weights given to the three members of the triple should properly reflect the individual significance of each parameter. For example, we can give the weight of 0.33 to each of them. As in the white board application, the weights can be dynamically adapted during runtime.

In terms of adaptability, IDEA does not directly interact with the application's clients because it is the booking servers that ultimately commit updates. However, it is difficult to decide the preference of each booking server because it is the overall system's performance that matters. For this reason, IDEA runs a background inconsistency resolution protocol among the booking servers periodically to improve the consistency from time to time. Clearly, there is a tradeoff between the frequency of background inconsistency resolution and the overhead of consistency control in that the more frequently the resolution protocol runs, the better consistency the system can achieve. On the other hand, high resolution frequency will incur high overhead and increase the likelihood of underselling as the system may be overwhelmed by a highly frequent resolution process.

In an e-business environment, neither underselling nor overselling is desirable. Thus a higher consistency level is not necessarily always better. Hence, the frequency of background resolution cannot be too high even if the system can sustain it. In practice, an ideal frequency can be deduced or learned (*e.g.*, through machine learning techniques [67]) from a long period of running in the following manner. First, IDEA sets an initial frequency and adjusts it on the fly based on system's load. Second, when the frequency is too low (and the consistency level is low) and causes overselling, IDEA

will increase the frequency beyond the current level and keep the frequency above this one to avoid overselling. Similarly, when the frequency is too high (and the consistency level is high) and causes underselling, IDEA will decrease the frequency below the current level and keep the frequency under this one to avoid underselling. Overtime, IDEA will learn the two boundaries within which it can adjust the frequency.

4.6 Evaluation

To evaluate the adaptability and performance of IDEA, we implemented IDEA and two emulated applications (a virtual white board and an online airline ticket booking application) that run on top of IDEA and are deployed on the Planet-Lab [77]. The two applications are emulated by following their operational sequences. In the case of a distributed white board application, we abstract the distributed white board as a set of objects that are replicated on each participating node. Then, we treat each update on the white board as a write operation on its local replica. Similarly, for an airline ticket booking application, each booking server has a replica of it and each update is considered as a write operation in its local replica. Due to the lack of available traces, we use a synthetic workload that assumes uniform distribution of the updating frequency for both applications. After updates are issued, IDEA works to maintain the overall consistency level of the virtual white board according to the protocol. Because our purpose of the experiments is to evaluate the performance and effectiveness of IDEA, we assume that these updates are all conflicting with one another (otherwise, IDEA needs not to care about them). While the two applications look similar at this abstract level, they differ in how the consistency is maintained: a participant in a distributed white board either gives a hint about their consistency requirement or interacts with IDEA on-demand; booking servers in an

airline ticket booking application, however, only depends on automatic consistency resolution whose frequency can be adjusted because, unlike participants in a white board, each booking server does not care about its view of consistency but, instead concerns with the overall consistency that affecting the business goal that matters.

We use three metrics–namely, delay, consistency level, and incurred overhead–to measure the performance of IDEA. Delay information is important because it determines the performance of IDEA. Consistency level is also an important metric because it controls the QoS perceived by participants. Finally, we evaluate the incurred communication overhead, measured in number of protocol messages, by IDEA to demonstrate its scalability (the lower the overhead IDEA incurs, the more scalable it is). As well, all the results shown in this section are the average value of five runs.

As mentioned earlier, we focus on two aspects of IDEA: its adaptive interface and its performance. To evaluate the adaptive interface, we use an emulated distributed white board application and let users interact with it in an on-demand fashion. To evaluate the performance, we first investigate the response time of consistency resolution in a distributed white board scenario and then evaluate the communication overhead in an airline ticket booking system. As for correctly re-order conflicting updates, we simply choose the one with higher ID as the reference consistent state, one of three policies discussed in Section 4.4.5.

In this section, we focus on investigating the feasibility of the IDEA approach and exploring the characteristics of IDEA from different perspectives. It would also be interesting to see how IDEA compares with other protocols, such a lock-based strong consistency control protocol, experimentally. We would like to pursue this comparison in the future.

Also, in Section 4.4.4, we discussed the rollback mechanism that is triggered when the detection in the bottom layer returns an actual consistency value much worse than

the one returned from the top layer. In this evaluation portion of the study, however, we do not consider the rollback mechanism for two reasons. First, according to our previous analysis, the possibility that the top layer fails to detect an inconsistency is indeed very small (less than 5% in a variety of scenarios and as small as 0.04% in certain cases). Second, this evaluation serves the purpose of validating the design of IDEA and the rollback mechanism is not essential for this purpose because the rollback mechanism uses TTL to control the detection delay in the bottom layer and we do not expect it to be a performance bottleneck.

4.6.1 The Adaptive Interface of IDEA

Here we use a hint-based application to show the effectiveness of the adaptive interface of IDEA. In this application, each user indicates a certain tolerance level to the inconsistency level, which is the *hint*. The assumption is that when the system's consistency level is above the hint level the user is satisfied. Thus, IDEA only resolves inconsistency when the consistency level drops below the hinted level.

The experimentation is run on 40 Planet-Lab nodes, of which four are assumed to be concurrent writers of a given file. After warming up, the four writers form a top layer of four nodes. Because these 40 nodes span US and Canada, we believe that it is representative of an Internet-scale distributed system. While a top layer of four nodes is not a large one, it is sufficient for our investigation purpose because they are carefully chosen so that they are far apart from each other. Also, based on data collected from this setup, we will later extrapolate the result to predict the performance of IDEA in a more dynamic system (with more simultaneous writers).

After the warm-up process, the four nodes on the top layer start to update the same file every 5 seconds during a 100 second period, which amounts to a total of 20 updates. This experiment is run with two different hint levels. First, we set a user's

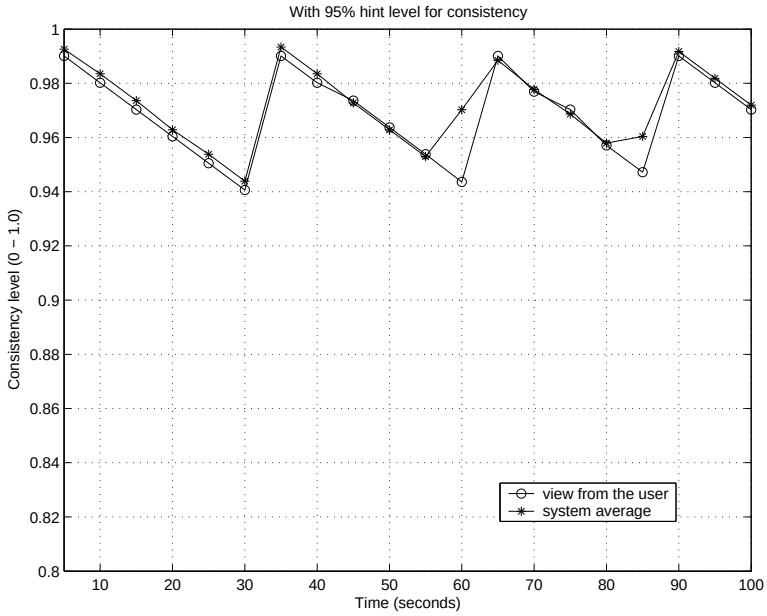


Figure 4.6: Setting hint level at 95%

hint level to 95%, which allows IDEA to kick in when the user's consistency level is lower than 95%. Second, we set the user's hint level to 85%, where IDEA kicks in when consistency level is below 85%. The results are summarized in Figure 4.6 and Figure 4.7, respectively, in which the "view from the user" is the consistency level of the writer that triggers the IDEA protocol and the "system average" is the average value of the consistency level of the four writers.

As shown in the two figures, the consistency level is improved right after IDEA kicks in, by evoking the active resolution scheme. In both scenarios, IDEA was able to bring the consistency level back to satisfactory states fairly quickly. While these two figures shwo that the consistency level is brought back to acceptable states after five seconds, IDEA actually brings the system's consistency level back to acceptable states in less than one second, as shown in Section 4.6.2. This is because we sample the system's consistency level every five seconds in this experiment.

However, this scheme will cause the user to suffer for at least a short period of

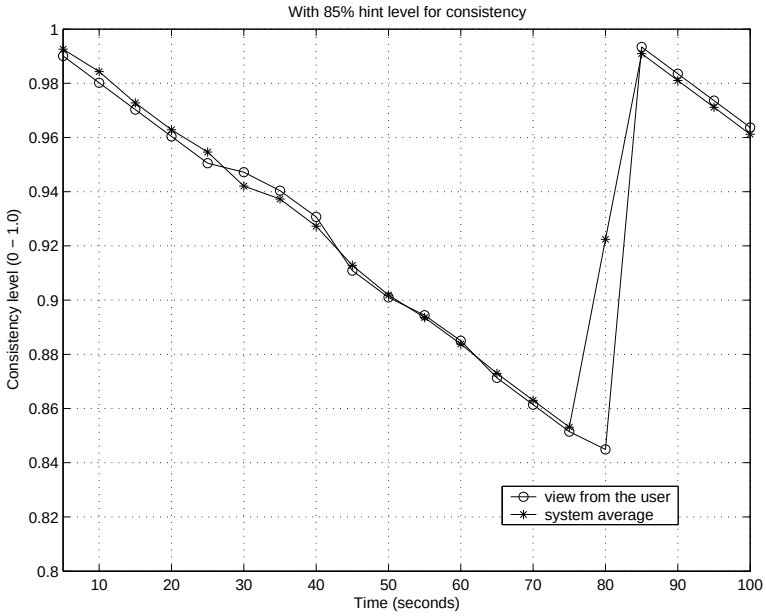


Figure 4.7: Setting hint level at 85%

time during which the user is in an inconsistent state, an undesirable event. To cope with this, the user can set a hint level slightly above its real acceptable consistency level. In this way, IDEA starts to resolve any inconsistency early enough to keep the system's consistency level above the user's *real* hint level all the time. The lowest consistency levels for users in the two experiments are 94% and 84% respectively. Thus, if a user's real hint level is 94% or 84%, he/she can set the hint level to 95% or 85%, respectively, to avoid suffering from being in inconsistency states all together.

Then we combine the two settings by running the experimentation for 200 seconds. Same as above, the four writers update the same file every 5 seconds, which amounts to a total of 40 updates per writer. We initially set the users' hint levels to 95% and reset the hint levels to 90% after 100 seconds. The result is summarized in Figure 4.8. The achieved lowest consistency level for writers (even for the one with the worst consistency) in the experiment is about 95% in the first 100 seconds and 90% in the second 100 seconds.

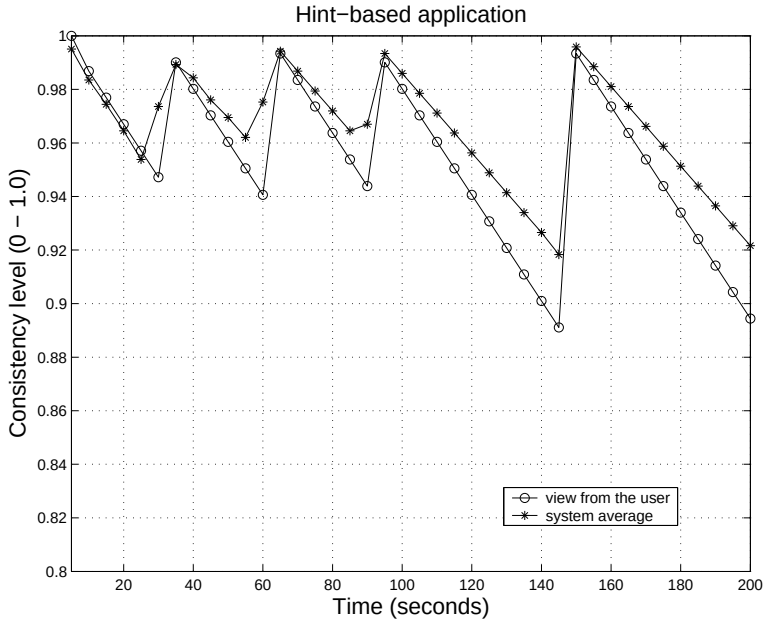


Figure 4.8: Hint-based application

Collectively, these experiments and results clearly show the feasibility and effectiveness of the IDEA's adaptive interface.

4.6.2 IDEA's Response Time

To evaluate the performance of IDEA's active consistency resolution scheme in terms of response time, we consider a simple distributed white board application in which four concurrent writers form the top layer. Because we treat distributed white board as an on-demand application, a node triggers active resolution when it feels that the consistency level is not satisfactory. We run the consistency resolution scheme four times, and each time we pick a different writer to initiate the request for active consistency resolution. We use the average of the four runs as the final result.

Table 4.3 shows the response time breakdown for the two phases involved in an active consistency resolution. As elaborated before, phase one is a call-for-attention and phase two resolves inconsistency among the top-layer nodes by visiting them

Phase	Delay for one round of active resolution
Phase one	0.46825 ms
Phase two	314.241 ms

Table 4.3: A breakdown of two phases involved in active resolution

sequentially.

The result shows that phase one is much shorter than phase two. This is due to two reasons. First, the operation in phase one is only a call-for-attention, thus there is little computing overhead involved; on the other side, phase two involves collecting replicas' information (such as comparing version vectors) and resolving the potential inconsistencies, which has higher communication as well as computation overhead. Second, the call-for-attention operations for different nodes are executed in parallel, which further improves its speed; for the second phase, though, it traverses all the top layer members sequentially to resolve the inconsistencies one by one. In this design, we choose to run the second phase sequentially because it simplifies the active writer's job–it just needs to communicate with one other active writer at a time. However, if performance is a concern, it is not difficult to exploit parallelism for the second phase (*i.e.*, letting an active writer contact all the other active writers at once).

Now we use this result to estimate the scalability of active resolution as follows. Because phase one is executed in parallel, its performance does not change significantly with the top layer size. Since phase two is executed sequentially, its response time increases approximately linearly with the top layer size. The result in Table 4.3 is from a top-layer of size four where there are three nodes in top layer that the initiator needs to contact, thus on average, the cost for each additional member in the top layer is roughly 104.747 ms (314.141 / 3, because there are only three nodes need to be traversed). Thus the response time of active resolution for a top layer of size n is extrapolated as in Formula 4.2:

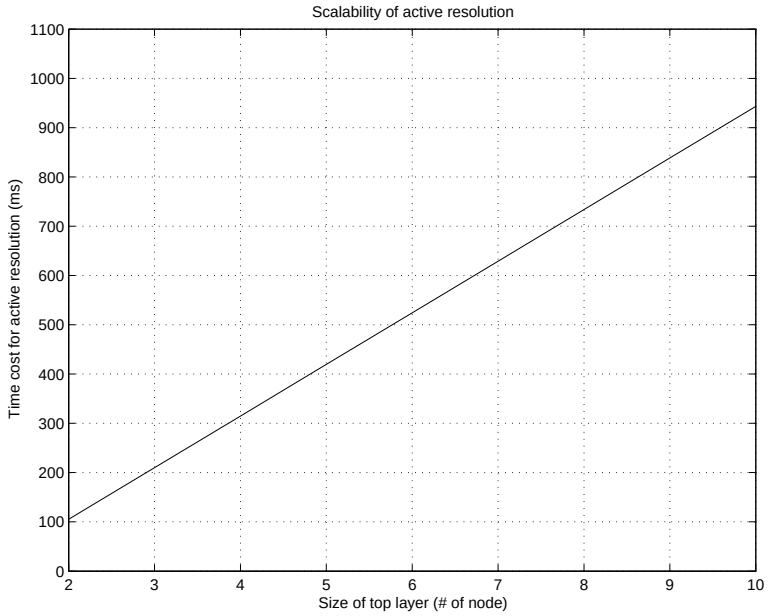


Figure 4.9: The overview of IDEA

$$Delay = 0.46825 + 104.747 \times (n - 1) \tag{4.2}$$

We depict the cost for active consistency resolution with top layer size up to ten in Figure 4.9. From the figure we can clearly see that, even with ten simultaneous writers, which is highly unlikely in a short period of time (in order of seconds) in practice, the cost of active resolution is still below one second. In an Internet-scale system, we believe that this is a reasonably good performance because it is not uncommon that, in a large-scale distributed system, a message is to be delayed for seconds or more [40], thus offsetting the impact of delay caused by IDEA. Nonetheless, as explained earlier, a parallelism mechanism can be easily deployed to further improve the responsiveness of IDEA, which is useful in a scenario where the number of active writers is rather large.

We elaborated in Section 4.4.6 that background resolution essentially consumes the same amount of time incurred by the phase two of active resolution (first to collect

all the updating information; second to send the consistent replica image information back), the delay of background resolution can thus be presented approximately as in Formula 4.3.

$$Delay = 104.747 \times (n - 1) \tag{4.3}$$

Clearly, the cost is even smaller than the one of active resolution. Together, the two measurements indicate that neither the active nor the background consistency resolution scheme in IDEA slows down the system even with a relatively large number of simultaneous writers.

4.6.3 IDEA's Communication Overhead

To measure the communication cost in an appropriate context, we deploy IDEA in an automatic airline ticket booking system that mainly depends on the background resolution scheme to maintain consistency among nodes. Running periodically, the background resolution scheme brings the system's consistency level back to satisfactory states periodically.

Naturally, consistency resolution implies communication overhead, which is what we are going to measure here. In this application, however, the frequency of running the background resolution scheme is also a design tradeoff: the more frequently it is run, the better the system's average consistency level, but the overhead could become formidable. Thus, as stated in Section 4.4.7, there is a need to control the total overhead of IDEA below a certain ratio of the currently available bandwidth. Hence, after evaluating the absolute communication cost, we will further explore the derivation of an optimal rate of running background resolution based on system's total capacity here.

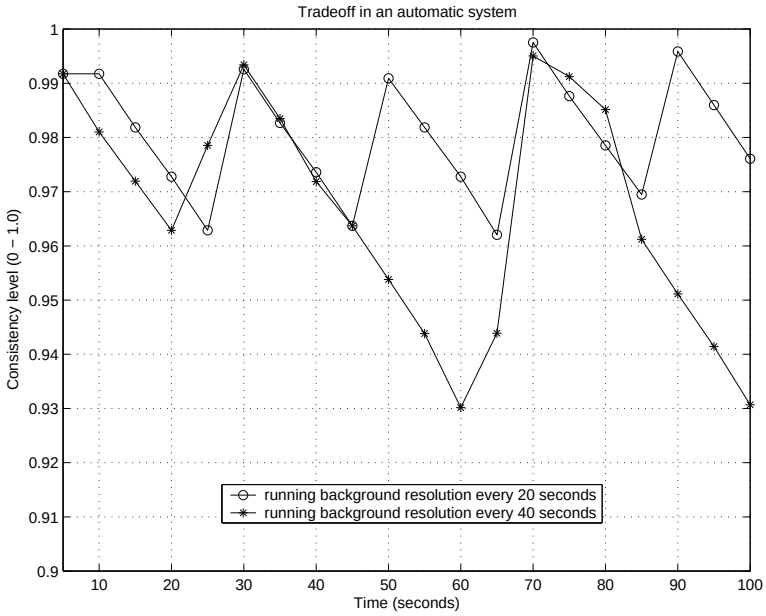


Figure 4.10: An automatic system

Frequency	Overhead (# of exchanged messages)
20 seconds	168
40 seconds	96

Table 4.4: Overhead

We run this experimentation in the same environment as in the previous section with two settings: first, we allow the background resolution scheme to kick in every 20 seconds; second, we allow the background resolution scheme to kick in every 40 seconds. The results are shown in Figure 4.10 in which the consistency level is the one perceived by all the top layer nodes. The incurred overhead, in terms of the number of exchanged messages, is summarized in Table 4.4.

If we assume that each packet has a size of 1KB (this is a reasonable assumption because the version vector only needs several bits to store its information), the total overhead of the first run (every 20 second) is 168KB and, after dividing it by the 100 second of running time, equals to 1.68KB/s, or 13.5bps, which is a very minimal bandwidth cost even for dial-up connections.

This experiment also clearly shows the tradeoff between overhead and achieved consistency level. That is, with the increased frequency of background checking and resolution–thus the increased overhead, the average system's consistency level becomes higher, at the expense of higher overhead (Table 4.4). Here we try to derive a formula to determine an optimal rate of running background resolution as follows.

First we assume the existence of a monitoring program on the server side to monitor the current total available bandwidth, which we believe is a reasonable assumption. Then all that is needed in order to control the overhead of IDEA under a certain percentage of the current total available bandwidth is the communication cost of one round of IDEA background resolution. For example, if the current total available bandwidth is b Mbps, the maximal percentage of the bandwidth that can be used by IDEA is $x\%$, and the one round communication cost is c Mb, the optimal rate of the background resolution can be presented as:

$$\text{Optimal rate} = \frac{b \times x\%}{c} \tag{4.4}$$

To derive an optimal rate of background resolution according to Formula 4.4, we need to know the communication cost of one round c. From Table 4.4, we have total six runs in these two experiments and we can approximate one round of background resolution as:

$$\#\text{ of messages} = \frac{\text{Total number}}{\text{rounds}} \tag{4.5}$$

and the final value is (168+96)/6, which is 44.

Second, because the average size of exchanged messages varies from application to application, we use a parameter s to denote it and practitioners should substitute it with any real value they have. Thus the one round communication cost of the

background resolution in the experimentation setup is $c = 44 \times s$. At this point, practitioners can use the derived c value to derive an optimal rate based on system's ongoing load by the following Formula 4.4.

We need to mention that the derivation of the optimal rate is to illustrate that how to derive one optimal rate given certain system data. This derivation is to show the possibility of deriving an optimal rate and the result is indeed to a large extent specific to this particular application.

Finally, because the communication cost scales linearly with the size of top layer, the communication cost (thus the optimal rate of background resolution) for a particular application can be extrapolated according to its typical top layer size.

4.7 Summary

In this chapter, we presented IDEA, the adaptive consistency control protocol of IDF that is built on top of the inconsistency detection mechanism. IDEA achieves adaptability by resolving the detected inconsistencies based on applications' semantics: if the users can tolerate the inconsistency (in return for better system response time), IDEA will not resolve the inconsistency; if the users indicate that this inconsistency is not tolerable, IDEA resolves the inconsistency in a timely manner. IDEA is evaluated by prototyping on Planet-Lab and the results show that: (1) IDEA achieves adaptability by adjusting the consistency level according to users' preference on-demand; (2) IDEA achieves low inconsistency resolution delay; and (3) IDEA incurs minimal communication cost even for dial-up connections.

Chapter 5

CVRetrieval: Consistent View Retrieval

As we have discussed in the previous chapter, consistency control among participants in distributed online collaboration applications has been an active research area [16, 50, 82, 88, 103, 105, 106]. As well, applying relaxed, but not overly loose, consistency control to achieve a more scalable system has become a mainstream method [106].

While we agree with this kind of tradeoff, we believe that the current mode of consistency maintenance is still not efficient enough because most consistency control schemes in use today still rely on applying the same protocol on all participants, which could induce high communication overhead [16]. Here, when we say "participants", we mean the users who are actually interested in the consistency of the shared object.

In this dissertation, we refer to the enforcement of consistency through communication among all the participants as *consistency maintenance*. With this strategy, the maintenance cost grows with the number of participants. In a truly large-scale system, such as online gaming, the consistency maintenance cost can be formidable. There are a number of systems available to support consistency maintenance, such as

our own related work IDEA of Chapter 4 of this dissertation and others [16, 105].

A straightforward way to reduce the maintenance cost is to reduce the number of participants that a consistency maintenance module needs to include. We believe that this is both doable and preferable. First, this is doable because not all participants in a collaboration application are equally active or engaged. In a virtual white board scenario where students listen to lectures, for example, the lecturers are more likely to issue updates while a majority of the students are observers–they monitor the white board and only occasionally issue updates. From a consistency maintenance point of view, the lecturers are more important than passive students. So there is really no real need to consider the passive student group as far as consistency maintenance is concerned at most of the time. The rationale behind this is that if a participant does not have frequent updating activities it is far more cost-effective to satisfy his or her needs on-demand. Second, reducing the number of participants instead of redesigning consistency protocols is preferable because the former does not change the way most current consistency control protocols work, and hence is easier to be adopted. In this dissertation, we refer to the strategy of satisfying passive participants' consistency need on-demand as *consistency retrieval*.

In this chapter, we present the *Consistent View Retrieval* (CVRetrieval) framework that supports the functions of consistency retrieval. To support the retrieval functions, CVRetrieval deploys publishers and subscribers (forming a so called publish-subscribe infrastructure that is used throughout this chapter) in the system to serve as rendezvous points, similar to the publish-subscribe schemes [3]. CVRetrieval chooses publishers and subscribers based on applications' semantics to capture the common interest among participants.

It is worth mentioning that, while it is easy to statically separate passive participants from active participants and only maintain consistency for active participants,

CVRetrieval is significantly different from such a scheme in two aspects. First, CVRetrieval is not merely differentiating active and passive participants once and staying with a fixed differentiation permanently. Instead, differentiation in CVRetrieval is a dynamic one, meaning that the active and passive participants are relative concepts and can change from time to time. The ability to capture this dynamics is a salient feature that sets CVRetrieval apart from any static approaches. Second, CVRetrieval assumes that passive participants do occasionally care about consistency, instead of assuming that they are not interested in the shared objects at all.

Since CVRetrieval does not actively maintain consistency for passive participants, CVRetrieval has to provide a way for these passive participants to access consistent objects when the need arises. From the passive participants' point of view, they only need to know where to find a consistency object. We treat this as a find-the-entry-point problem. To solve this problem, CVRetrieval deploys a publish-subscribe infrastructure to publish entry point information to the passive participants. In this way, CVRetrieval satisfies passive participants' consistency needs with an on-demand fashion.

In the rest of the this chapter, we first discuss architecture and the detailed design of CVRetrieval. Then, we analytically evaluate the scalability of CVRetrieval and experimentally evaluate its performance through prototyping on the Planet-Lab testbed respectively. After that, we discuss the targeted applications of CVRetrieval.

5.1 CVRetrieval Architecture

CVRetrieval is designed to improve the efficiency of consistency control by providing a consistent view retrieval service for the observers of an application while letting an existing consistency maintenance protocol to maintain the consistency for writers.

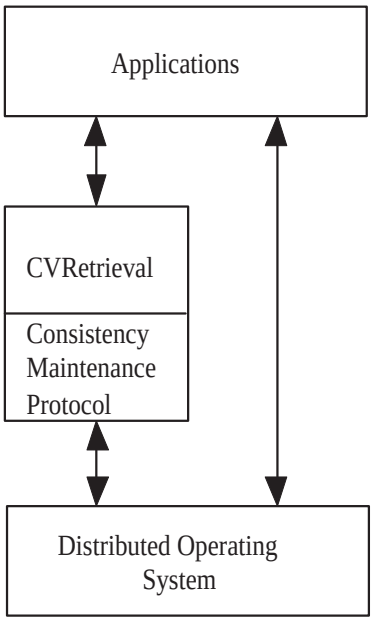


Figure 5.1: The Architecture of CVRetrieval

The architecture of CVRetrieval is depicted in Figure 5.1.

As shown in the figure, CVRetrieval is between the application layer and a general distributed operating system. When the application needs a certain consistency guarantee, it interacts with CVRetrieval. CVRetrieval depends on a consistency maintenance module to maintain consistency among writers and to guarantee the consistency level of the retrieved view. Finally, applications interact with the distributed operating system directly when no consistency issue is involved. In the figure, it is the requests that are passed between the modules. For example, the arrow from application to CVRetrieval means the application asks CVRetrieval for consistency information; the arrow from CVRetrieval to applications means that CVRetrieval returns consistency information back to application. The details of CVRetrieval, including the publishers and subscribers, are discussed in the following design section.

5.1.1 Using Any Consistency Maintenance Protocol

Theoretically, CVRetrieval can work with any consistency maintenance protocol. This is because, to use CVRetrieval, a consistency maintenance protocol only needs to do two things, namely, differentiating observers from other active participants and defining the entry point for CVRetrieval to retrieval the consistent view. The observers use CVRetrieval to retrieve consistent view by using the published entry-point information, while the consistency maintenance protocol enforces consistency among the writers.

However, there are two practical issues with regard to the status of participants that need to be handled. First, an observer may observe 90% of the time, and then active 10% of the time. Second, an observer may become an active participant for one hour, and then go back to being an observer for another two hours, and then come back to be an active participant again.

For the first issue, a participant needs to register with CVRetrieval about the change between active participant and an observer. For example, if a participant changes from an active participant to an observer, it needs to register with a proper subscriber so that it can start receiving the published information, which is used for retrieval purposes; if a participant changes from an observer to an active participant, it needs to unregister with its subscribe. Since we use IDF, especially IDEA, to maintain consistency among active participants, the active participants (old ones and new ones) will be captured by IDEA implicitly.

To handle the second issue, as shown in later sections, we further divide active participants into active writers and passive writers. These two types of writers can change status fairly fast. This change will be tracked and represented properly by the two-layer structure proposed in IDF. (We use IDEA for active participants, so the two-layer structure is inherent.)

5.1.2 Using IDEA as Consistency Maintenance Protocol

CVRetrieval uses IDEA as the consistency maintenance module, the main enabling technology, due to its adaptability and our familiarity with it. As described in the previous chapter, IDEA provides a detection-based two-layer infrastructure that checks the consistency level of a file and resolves the inconsistency among all potential writers of that file. Further, IDEA implicitly divides the writers (*i.e.*, active participants) into two groups: active writers and passive writers. This differentiation is based on each writer's updating frequency: if its frequency is above a pre-defined threshold, it is classified as an active writer and is put into the top layer. Other writers are classified as passive writers and are put into the bottom layer. The rationale behind this distinction is that, by focusing primarily on active writers in the top layer where most inconsistencies arise, IDEA can detect and resolve inconsistencies more efficiently. Also, under a variety of conditions, the two-layer infrastructure can capture most inconsistencies in the top layer with minimal delays.

Also, based on this efficient inconsistency detection mechanism, IDEA maintains consistency of a system in the following manner: (1) IDEA resolves inconsistency in the background periodically to improve the consistency level continuously; and/or (2) upon any participant's request, IDEA can actively resolve the inconsistency on demand. CVRetrieval uses IDEA to guarantee the consistency level of the retrieved view of a distributed online collaboration application.

The main function of CVRetrieval is to retrieve a consistent view for passive participants on demand by publishing entry point information to them. Relying on IDEA, the entry point is any active writer due to IDEA's ability of resolving inconsistencies from any active writer, as explained previously. However, the locations of the active writers are not fixed and their number could be fairly large, as explained below.

- **The member of active writers is not fixed.** This is because the participants' updating patterns change from time to time. Current active writers may not be active writers a moment later. While history data can be collected, it is still not clear how they can help predict future active writers.

- **The number of active writers is potentially large when multiple files are considered.** While the number of active writers for a particular file is usually small, the number can quickly add up when we consider hundreds or even thousands of files in a truly large system. The challenge is to handle this large number of active writers without incurring high communication cost.

And this calls for an efficient and flexible publish-subscribe infrastructure.

5.2 System Design

Several important design issues are addressed in this section:

- How do participants join the system and how to map the participants to the IDEA infrastructure?

- How does IDEA communicate with the publishers so that the latter have the updated information of the top layer nodes (that include all active writers) for different object?

- How do subscribers subscribe on behalf of their clients?

- How does the publish-subscribe scheme work?

Throughout this section, we use a virtual white board application to make the discussion concrete.

5.2.1 A Virtual White Board Scenario

We consider a distance education scenario in which several lecturers give lectures and a group of students join the discussions by manipulating a virtual white board (logically centralized and physically distributed on each participant's site). Other students who are not part of the discussion group will passively observe the discussion by watching the virtual white board via their local replicas.

In this scenario, the lecturers and the students in the discussion group conduct active discussions by issuing updates on the white board. Due to the nature of discussion, not all the members in the discussion group will speak up at the same time. During the discussion, membership of the active white-board-based speaker/writer group will change constantly, and such change is usually unpredictable because of the spontaneity of an active discussion.

5.2.2 Participants Join the System

We assume that there is a mechanism for participants to know the ID of the white board session and the time when the session starts. In practice, this can be done by some offline method, such as through an email list.

After all the participants log in, they form a group. Each participant modifies his or her own white board and those updates will show on others' white boards.

5.2.3 Mapping between Participants and the IDEA Infrastructure

As illustrated in Figure 5.2, we differentiate three types of participants: active writers, passive writers, and observers. They are mapped to IDEA as follows.

First, CVRetrieval differentiates observers from writers. When participants log in

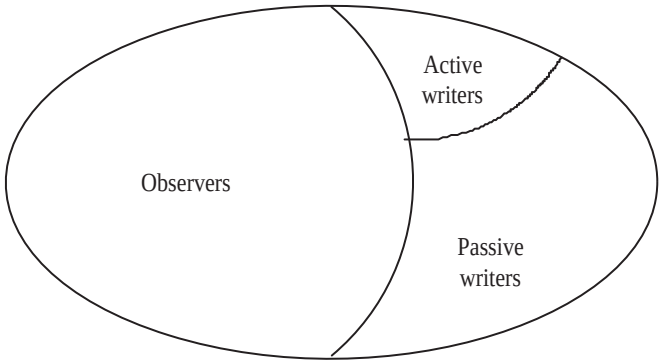


Figure 5.2: Three types of participants

the white board application, they are required to indicate whether they are members of the discussion group. If yes, they are characterized as writers; if no, they are classified as observers.

Second, IDEA differentiates active writers from inactive writers after the system starts to run. IDEA tracks active writers (by its top layer) and passive writers (by the bottom layer) based on their updating frequency.

It is worth mentioning that the three user classes are relative concepts and their memberships can change from time to time, especially in an Internet-scale distributed systems which are quite dynamic.

5.2.4 The Workflow

Figure 5.3 shows the workflow of the publish-subscribe mechanism as well as the retrieval process.

The basic publish mechanism is shown in Figure 5.3(a). In step 1, the active writers notify publisher about their presence; in step 2, a publisher notifies its subscriber about the up-to-date active-writer group; finally, in step 3, a subscriber notifies its clients (the observers) about the active-writer group.

The active retrieval process is shown in Figure 5.3(b). In step 4, an observer issues

a retrieve request to its subscriber. If the subscriber has a valid cache, it will return the local copy to the observer (step 7); otherwise, it requests a consistent view from one of the active writers (step 5) and, after receiving the view (step 6), it returns the copy to the observer (step 7) and caches the view locally.

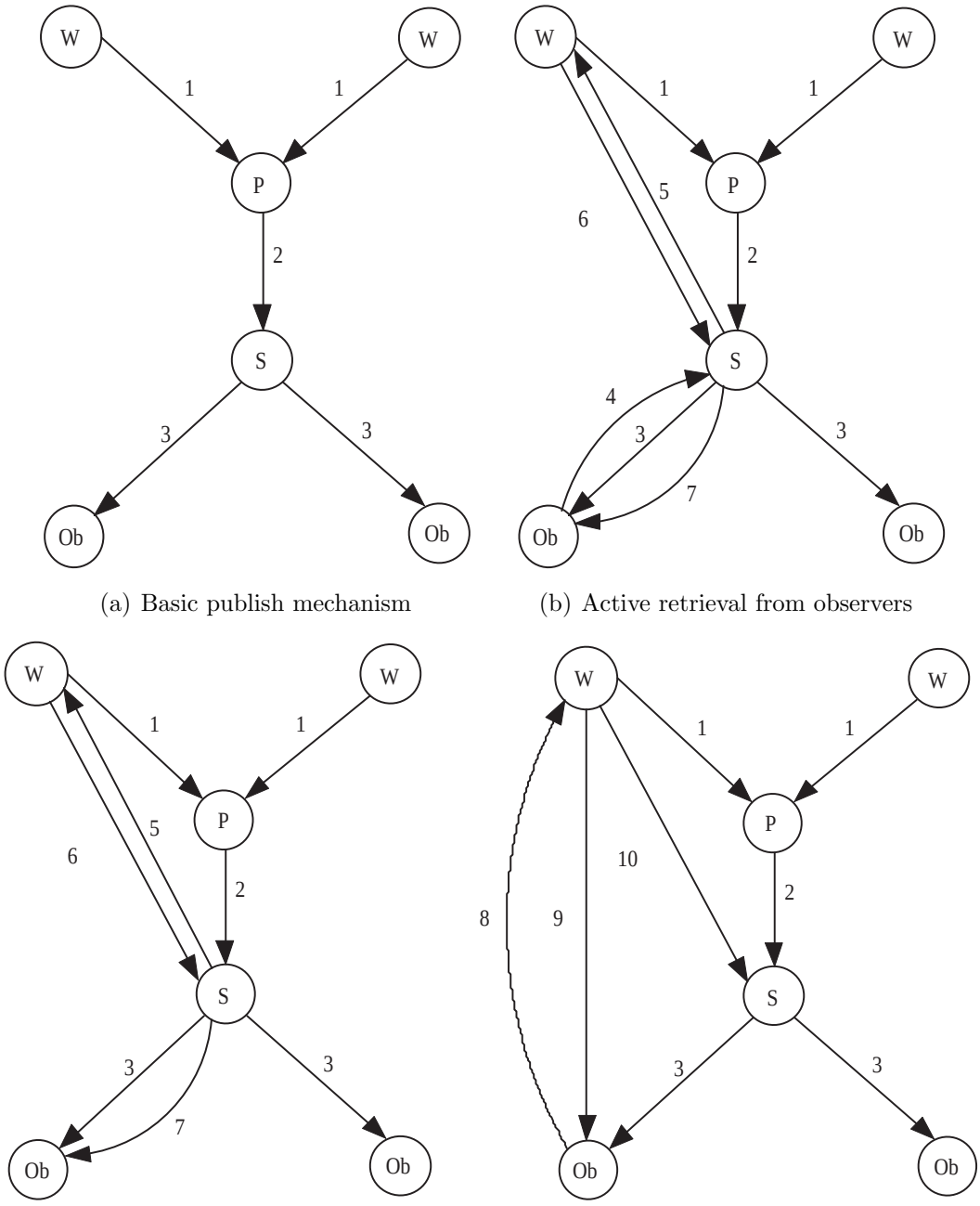
An observer can also indicate his or her preference to retrieve a consistent view periodically. In this case, the observer does not need to explicitly issue a retrieval request on-demand. As shown in Figure 5.3(c), this process is similar to that in Figure 5.3(b) except that there is no step 4, and steps 5 through 7 are executed periodically.

If the subscriber is already overwhelmed by the retrieval requests or publishing, there is no point of sending more retrieval request to it, and that is where the active-writer group information received by observers in step 3 comes into play. As shown in Figure 5.3(d), an observer can use its knowledge of the active-writer group to contact a nearby active writer directly (step 8 and 9). As an optional step, the active writer can forward a copy to the subscriber so that the subscriber will have a fresh copy as long as it is able to handle more requests again (step 10).

Finally, the complete process is illustrated in Figure 5.4. We will discuss the key components of the process in more details in the rest of this section.

5.2.5 Communication between IDEA and Publishers

In CVRetrieval, each object has a designated publisher that is responsible for publishing the top-layer nodes' information on behalf of the object. In a white board application, for example, the shared object can be the different figures that people are simultaneously drawing. Because white board is usually used to do collaboration among a small group of people, the number of shared objects is small, comparing to that in other applications, such as online-gaming. There are two issues associated with publishing objects information: (1) how to map an object to a publisher? (2)



(a) Basic publish mechanism

(b) Active retrieval from observers

(c) Automatic/Periodic retrieval from observers

(d) When subscribers are already overwhelmed

Figure 5.3: Workflow of CVRetrieval (W: active writer; P: publisher; S: subscriber, Ob: observer)

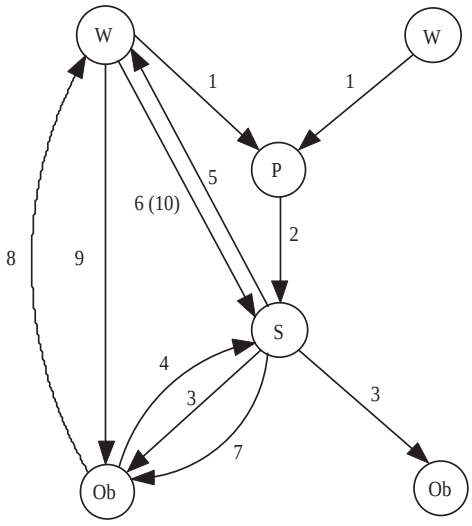


Figure 5.4: The complete process

how do publishers learn the top-layer nodes' information from IDEA?

There are two ways to map an object to a publisher based on the total number of shared objects. If the number of shared objects is small in an application, such as in the white board application, the shared objects can be mapped to a single publisher. If the number of shared objects is large, such as in online gaming, certain mechanism is needed to balance multiple publishers' load. Hashing table based scheme (choose publishers based on the hashed value of the object IDs), such as DHT [83, 85, 93], is desirable for both its load balancing ability and its easy lookup (subscribers can find the right publishers by hashing the object IDs themselves).

The publishers learn the top-layer nodes as follows. From the mapping procedure, the top-layer nodes of an object know where their corresponding publisher is. Here, the top-layer nodes, a term from IDF, are those actively updating participants of a shared object. In CVRetrieval, the top layer nodes will communicate with their publisher whenever it joins or leaves the top layer. The publisher will then publish these updates to its subscribers subsequently.

However, this published information may become obsolete due to the propagation

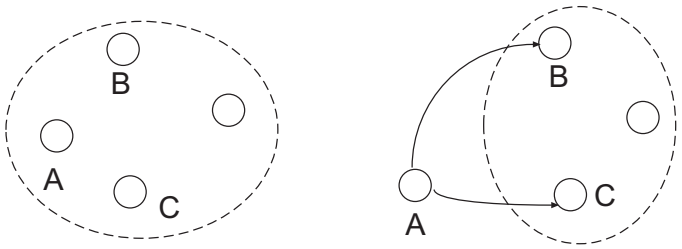


Figure 5.5: Use pointers to handle stale information

delay. For example, it may state that A is in the top layer of object f when in fact A is no longer in the top layer anymore. We use pointers to solve this problem. In an example illustrated in Figure 5.5, we let A keep two pointers of its fellow members when it is in the top layer of object f (left half of Figure 5.5) and, when A is no longer in the top layer, it can at least forward the request to the other top layer nodes (B or C in this case, see the right half of Figure 5.5). Because it is very unlikely that all three nodes would leave the top layer during the time of the propagation delay, this kind of information staleness is hidden from the users. In the case that this mechanism does not work, the request can always be returned back to the subscriber that can then pull updated information from the publisher (see Section 5.2.7).

5.2.6 Choosing Subscribers for Clients

We assume that stable servers, such as ISPs (Internet Service Providers), instead of the clients themselves, will be used as the subscribers for two reasons. First, stable servers such as the ISPs are much more stable than the clients, thus making the publish-subscribe structure (*i.e.*, the positions of publishers and subscribers) much more stable. Second, ISPs' interests do not change often comparing to the clients because ISPs' interests do not change with respect to how many and which clients are interested in an object, as long as some client is interested in it. Using clients as subscribers causes frequent membership change for a publisher and the publisher

that in turn needs to adjust its publishing scheme to reflect that change.

When a client becomes interested in an object, it informs its subscriber, which will subscribe the object's information if it hasn't done so. If the subscriber has already subscribed for that object, it will just add the client into its client list and inform the client about all the future updates about that object's top layer nodes. When a client loses interest in an object, it informs its subscriber who then deletes the client from its client list. After the deletion of its last client of an object, the subscriber will unsubscribe the object.

In CVRetrieval, a subscriber has two key responsibilities. First, it informs an active writer to periodically push new updates to it at a predefined rate on behalf of its clients who are so inclined, so that it can immediately forward the update to its clients whenever a new update arrives. Second, when a client explicitly asks the subscriber to retrieve a consistent view, the latter either returns the view from its cache (if one exists) or retrieves the view directly from the writer.

One key challenge is that the shared objects in our applications, unlike mp3 music files, may be perishable in that, as time goes by, a perfectly consistent view can potentially become very inconsistent. Thus, directly sharing a previously retrieved view without the consideration of its timeliness is pointless. In CVRetrieval, the sharing can be done with time-conscious caching: when a subscriber receives a new consistent view, it caches the view after it has forwarded the view to the interested clients; later, when another client is asking for this view, the subscriber decides whether the cached view is still satisfactory for this new request by considering the time gap. This way, different participants can share a retrieved consistent view through their local subscribers.

5.2.7 The Publish-Subscribe Scheme

We use a multicast tree [14, 20, 110] to send information from publishers to their subscribers. Each publisher builds a multicast tree and an interior node forwards the packets further down the tree only if there are some nodes in its subtree that have subscribed to it. While there are other publish-subscribe schemes available, we choose the multicast tree structure because it provides a stable infrastructure appropriate for our targeted, long-lasting applications where publishers-subscribers relationship is/remains stable.

In a naive form, the publisher sends the whole top layer information down the tree structure to all the subscribers. To improve the system's scalability and efficiency, CVRetrieval incorporates the following optimizations.

First, a publisher in CVRetrieval only sends a subset of the list of the top layer nodes to each subscriber to preserve the network bandwidth. This raises two questions: how to choose a subset for a given subscriber and how to disseminate different subsets of top-layer node information through a multicast tree?

When choosing the subset, the publisher has several factors to consider. First, the active writers in the subset should be physically close to the subscribers so that the retrieval can be done efficiently. Second, one or two remote active writers can be included in each subset to provide redundancy because physically close active writers tend to go down at the same time because physically close machines (active writers are machines too) tend to go down at the same time (for example, due to loss of electricity power). Third, the publisher needs to consider load balance so that no active writer is overwhelmed by retrieval requests. Here, the load balancing for active writers is not for the write operation carried out by the active writers; instead, it is for balancing the amount of retrieval requests handled by active writers. We need this load balancing scheme to serve the scenario where subscribers will pull from active

writers directly about the consistent view, as shown in Figure 5.3(d).

Now we illustrate how to disseminate the different subsets via a multicast tree. First of all, the subscribers report their physical locations to the root in a bottom-up fashion and the messages are aggregated at each interior node. Second, the publisher chooses different subsets for its *immediate* children in the multicast tree based on these children's subtree's interests (*i.e.*, the collective interest of the nodes in its children's subtree) and disseminates the subsets. For each interior node, it further divides the subset for its own immediate children. This process continues until the leave nodes are reached.

When a client explicitly retrieves a consistent view, as explained in Section 5.2.6, its subscriber will either return the view from its cache (if one exists) or pull one from a writer. While rare, there is a possibility that the writer is no longer an active one and the subscriber has no way of reaching another one. In this case, the subscriber will contact its publisher for the most up-to-date information about the active writer list.

5.3 Analysis of CVRetrieval's Scalability

In this section, we compare the communication cost of the CVRetrieval approach with other consistency maintenance approach. This analysis is crucial because the main hypothesis of CVRetrieval is that it can save communication cost, thus making the consistency control as a whole more scalable.

However, due to the long history of research on consistency maintenance, there exist a large number of consistency maintenance protocols that are suitable for various scenarios. The large number of protocols is a challenge to this analysis because the significant diversity in these existing protocols' application and operational envi-

ronments as well as design philosophies makes it hard, if not impossible, to compare CVRetrieval with all of them under one unified evaluation framework. To cope with this challenge, we simplify the analysis as follows without the loss of generality.

First, we classify the consistency maintenance protocols into four major categories by extending previous work in this area (categorizing consistency maintenance protocols for distributed collaboration systems) and compare CVRetrieval with each category. This is a simplification because these four categories cannot possibly capture all existing consistency control protocols. In the past several decades, there are many consistency control protocols have been proposed in literature. In this analysis, we do not, and do not intend to, capture all of them into the four categories.

Second, we realize that, to accurately compare communication cost of the protocols, we need to consider a number of factors, including the average communication message size, the traveling distance of each message, and the total number of messages. However, considering such details will make the analysis intractable because of dynamism and diversity in a large scale distributed system. Thus, in this analysis, we assume that all the protocols incur the same average message size and, on average, each message travels the same distance. Hence, the differentiator of the protocols is reduced to the total number of messages incurred by each protocol.

5.3.1 Categorization of Consistency Maintenance Protocols

Our categorization follows the research work by Yang and Li [105], but extending their work in two aspects. First, it covers more recent research work, including our previously developed detection-based IDEA protocol. Second, this categorization focuses primarily on consistency maintenance protocols in the online distributed collaboration systems, which makes it more focused than theirs. In this analysis, we consider four categories: locking, serialization, operational transformation, and

detection-based consistency maintenance.

Locking. Locking mechanisms control consistency by locking a data object and only allowing one user to modify the data at a time. Depending on whether the mechanism allows users to continue their work while a lock is requested and released, locking can be further divided into three types: pessimistic (work is blocked in both lock requesting and releasing), semi-pessimistic (work is blocked only in lock releasing), and optimistic (work in not blocked in either case). While locking is widely used in small networks, we believe that it is not suitable for distributed online collaborations because users usually do not tolerate long delay caused by the locking operation.

Serialization. In this mechanism, all the users are allowed to modify their replicas, but their updates need to be serialized at a single point to maintain a consistent state. There are two flavors of serialization: pessimistic serialization and optimistic serialization. While users are not allowed to continue their work until their previous updates are serialized in the former, the latter allows users to continue their work and will rollback their inconsistent updates when needed. Because we consider a replica-based distributed system, we assume that this is a distributed serialization. An example of this model is Deno [16], a peer-to-peer voting protocol in which each writer's update travels across the whole replica group to detect and resolve any inconsistency. During Deno's serialization process, further updates are allowed, so this is an optimistic serialization. While the enforcement mechanism of a bounded inconsistency level by TACT [106] can be generally considered as an optimistic serialization because they let each server loosen its consistency control to the degree that the total inconsistency across the server group is still within a predefined inconsistency bound, its enforcement mechanism is not a unified one. Rather, TACT developed a set of schemes to enforce the bound for different aspects of consistency. For this reason, the

mechanism of TACT is not directly comparable to that of CVRetrieval because the latter targets at a unified consistency maintenance protocol.

Operational Transformation. This mechanism differs from optimistic serialization in how it reacts to inconsistencies. While optimistic serialization repairs inconsistency when it arises by rolling back inconsistent updates, operational transformation does not undo the effects to reduce overhead. This operation is useful when the inconsistency is either not repairable or it is insignificant. Essentially, this is an extreme optimistic operation and, because it does not guarantee any level of consistency, we believe that it is not suitable for distributed online collaborations in which unbounded inconsistency can cause confusion and make meaningful collaboration impossible.

Detection-based scheme. We previously presented IDEA as the first detection-based consistency maintenance protocol for large-scale distributed systems. Instead of enforcing a fix consistency protocol beforehand, IDEA detects inconsistencies when they arise and resolve them based on the applications' ongoing need for consistency. IDEA achieves adaptability for the applications by considering application semantics, and supports flexibility by allowing the end users to adjust their consistency level on the fly. IDEA is suitable for distributed online collaboration because it allows the user to control to adjust the perceived consistency level.

Among the above four categories, locking and operational transformation are not comparable to CVRetrieval since the former causes long delay and the latter does not guarantee consistency level at all. Thus a meaningful comparison will be among the optimistic serialization, with Deno as a representative protocol, IDEA, and CVRetrieval with the goal of determining whether the added communication overhead of CVRetrieval is much smaller than the communication cost it saves by decreasing the communication cost of consistency maintenance.

5.3.2 Assumptions

To analytically evaluate the communication cost savings by CVRetrieval, we make the following assumptions and definitions.

1. c: the average number of simultaneous writers.
2. n: the total number of nodes in the system that join the consistency control process.
3. n_1: number of writers.
4. n_{hot}: number of active writers among the n_1 writers.
5. f_1: number of updates made by active writers during a given period of time t.
6. n_{pass}: number of passive writers among the n_1 writers, where $n_{hot} + n_{pass} = n_1$.
7. f_2: number of updates made by passive writers during a given period of time t.
8. n_2: number of observers, where $n_2 = n - n_1$.
9. p: total number of publishers in CVRetrieval.
10. s: total number of subscribers in CVRetrieval.
11. k: the number of objects to which each observer subscribes in CVRetrieval.
12. q_1: number of publishings during a given period of time t.
13. q_2: number of retrievals during a given period of time t.
14. C_deno: total number of messages exchanged in Deno.
15. C_idea: total number of messages exchanged in IDEA.

16. C_r: total number of messages exchanged in CVRetrieval.

As shown above, we use the number of messages exchanged as a metric to analyze the communication cost saving by CVRetrieval.

5.3.3 The Analysis

We conduct the analysis in three steps. First, we derive the communication cost associated with the consistency maintenance protocol Deno, followed by the derivation of communication cost of IDEA. Second, we derive the communication cost associated with CVRetrieval. Finally, we compare the three mechanisms. In this analysis, we consider the consistency control for one single object because this simplifies the analysis and, based on its result, it is easy to extend the analysis to multiple objects.

Communication cost of Deno

In Deno, each update travels the whole group and, when it meets another conflicting update, the update will be resolved at that time. In this analysis, each time an update reaches a node, we consider it as a new message because the node that is reached essentially regenerates the original message by relaying it. Thus, given an update, it only stops traversal when it meets another conflicting update. From the assumption 1, we know that there are c conflicting updates in the system at any given time on average. For simplicity, we further assume that the updates propagate along a linear structure (without this assumption, the updating process becomes intractable). Then, on average, an update travels $\frac{1}{c}$ of the network to meet a conflicting update and stops.

Now we calculate the communication cost as follows. Because there are n nodes in the system, each update needs to travel $\frac{n}{c}$ hops, which equals to $\frac{n}{c}$ messages per

update. In a given period of time t, there are $n_{hot} \times f_1 + n_{pass} \times f_2$ updates, so the total number of messages generated in a given period of time t is:

$$C\_deno = \frac{n}{c} \times (n_{hot} \times f_1 + n_{pass} \times f_2) \tag{5.1}$$

Communication cost of IDEA

In IDEA, the updates from active writers will be detected among the active writers in the top layer and those from the passive writers will need to go through the whole network to be detected in the bottom layer.

Similarly to the analysis in Deno, we assume the existence of c concurrent conflicting updates at any given time. However, in the case of IDEA, the updates from active writers stay in the top layer, implying that the active writers actually see less than c concurrent updates because the updates from passive writers do not appear in the top layer at the same time. So, while passive writers still see c concurrent updates, we assume that the active writers sees only c_{hot} concurrent updates, where $c_{hot} < c$. Then an update from a active writer will generate $\frac{n_{hot}}{c_{hot}}$ messages, and an update from a passive writer will generate $\frac{n}{c}$ messages. There are $n_{hot} \times f_1$ updates from active writers and $n_{pass} \times f_2$ updates from passive writers in a given period of time t.

For the communication cost associated with observers, we follow the calculation used in the Deno case and conclude that the overhead is two messages (one for request, one for reply) for each retrieval-type request. Then, because we have assumed that, on average, each observer will issue q_2 requests in time t, the total communication overhead is $2 \times n_2 \times q_2$.

Putting the communication cost of writers and observers together, the communication cost of IDEA is:

$$C\_idea = \frac{n_{hot}}{c_{hot}} \times n_{hot} \times f_1 + \frac{n}{c} \times n_{pass} \times f_2 + 2 \times n_2 \times q_2 \tag{5.2}$$

Communication cost of CVRetrieval

The communication cost of CVRetrieval involves three parts: (1) the detection of inconsistency among active and passive writers; (2) the cost associated with the publish-subscribe scheme, which includes the communication cost between writers and publishers, between publisher and subscriber, and between subscribers and their clients; and (3) the retrieval operation for observers.

First, CVRetrieval detects inconsistency among active writers in the same manner as IDEA because it depends on IDEA to maintain consistency. Thus, the communication cost incurred by active writers is $\frac{n_{hot}}{c_{hot}} \times n_{hot} \times f_1$. For passive writers, however, they need not to go through the whole network; instead, they only need to detect among the writers' group (with n_1 writers) that excludes the observers. Thus, the communication cost associated with the updates from passive writers is $\frac{n_1}{c} \times n_{pass} \times f_2$.

Second, for the communication cost associated with publish-subscribe scheme, we first derive the cost for one round of publish and then multiply it by the publish rate q_1 to get the total communication cost in a given period of time t. Because an active writer only notifies its publisher when it becomes a active writer and when it becomes a passive writer. Here we conservatively assume that, in one round of publish, half of the active writers are new ones (this is indeed a very extreme scenario because we essentially assume 50% of the active writers leave the group and the same number of new active writers join the group). Thus, in one round of publish, there are n_hot messages exchanged between writers and publishers because each old active writer or new active writer needs to inform exactly one publisher. Then, there are s messages exchanged between publisher and subscribers because there are s subscribers in total

and each needs to be informed exactly once. Finally, let's conservatively assume that all the n_2 observers will need to be informed about its subscription. Then we know that n_2 messages are exchanged in one round. Adding the three parts of cost together and then multiplying the publish rate, we get the total communication cost associated with the publish-subscribe scheme in time t, which is $q_1 \times (n_{hot} + s + n_2)$.

Third, each observer will retrieve a consistent view for the object he or she is interested in, which results in n_2 retrievals. Here we assume that most retrieval requests can be satisfied by two messages (one request to subscriber and one reply from subscriber) because we do not expect a majority of the requests to trigger a new retrieval from the subscriber, so there are $2 \times n_2$ messages exchanged in one retrieval operation. Finally, because we assume that each observer retrieve q_2 consistent views in time t, the total number of message exchanged in t is $2 \times q_2 \times n_2$.

So the total communication cost for CVRetrieval in a given period of time t, incorporating all three parts, is:

$$\begin{aligned} C\_r &= \frac{n_{hot}}{c_{hot}} \times n_{hot} \times f_1 + \frac{n_1}{c} \times n_{pass} \times f_2 + q_1 \times (n_{hot} + s + n_2) \\ &+ 2 \times n_2 \times q_2 \end{aligned} \tag{5.3}$$

Note that parameter s is related to n_2 because there are s subscribers serving the n_2 clients (recall that each observer subscribes to k objects). Although there is no ground rule about how many clients a subscriber should have, it is intuitive that the number of clients should not overwhelm the subscribers. Considering that the information that is being published is rather small in quantity (it is only a list of active writers and the message is only a few KBs), we believe that each subscriber can support at least up to 50 clients, which incurs less than 1MB data traffic and

should not be a burden for a subscriber. Thus, in the following analysis, we use $\frac{n_2}{50}$ as the value for s.

Further, the value of q_1 is associated with how frequent the active writer group changes and q_2 is associated with the observers' interests. Because CVRetrieval deals with loosely coupled distributed online collaboration applications, we believe that, in a short period time of t, it is sufficient to assign a small numerical value for q_1. For q_2, we believe that it should be reasonably large so that it can satisfy observers' needs for consistent views. However, q_2 cannot be too large, which implies smaller inter-retrieval time, because there is no point of issuing the second retrieval before response to the first request has arrived. Thus, we believe that it should be reasonable to make q_2 two to three times as large as q_1 as we feel it provides a sensible tradeoff.

The comparison

In this comparison, we first do an asymptotic analysis to compare the overall growth rate of Deno, IDEA, and CVRetrieval. Since the asymptotic analysis is approximate in nature, we then use a sensible setting of the parameters to calculate and compare the three protocols.

We conduct the asymptotic analysis as follows. In the equation 5.1 for the communication cost of Deno, n_1 and n_2 are fractions of n, so n_1 and n_2 grows as fast as n. Then, f_1 and f_2 are updates in a period of time and is not supposed to be a large number and won't grow with n, so we can safely treat them as rough constants. Hence, the cost of Deno would be $O(n^2)$.

For the analysis of the communication cost of IDEA, we follow the analysis the same way as that of Deno–n_1, n_2 have similar growth as n, f_1 and f_2 are more like constant. Then, from equation 5.2, the cost of IDEA is $O(n^2 + n)$, which is also $O(n^2)$. Similarly, the cost of CVRetrieval, derived from equation 5.3, is also $O(n^2)$.

Sets	n	n_1	n_{hot}	c	c_{hot}	f_1	f_2	q_1	q_2	s	Deno	IDEA	CVR
1	1000	50	10	4	3	5	3	2	5	19	42500	39667	13125
2	1000	100	20	4	3	5	3	2	5	18	85000	69667	17543
3	1000	200	50	4	3	5	3	2	5	16	175000	124667	36399

Table 5.1: Analytical Results

So the main message here is this, while there are differences in the communication cost among all the three protocols, the difference is not an exponential one. This makes sense because all three protocols, to some extent, depend on intercommunication of a group of nodes, which results the $O(n^2)$ result. The real difference is how large the group is–the larger the group, the more communication cost will be incurred. From this aspect, Deno has the largest group (the whole system), IDEA has a smaller number (only for the group of active writers). CVRetrieval has the same group size as that of IDEA but has a much smaller size of passive writers, hence achieving the smallest communication cost.

We now proceed to the second step of this comparison by comparing C_deno, C_idea, and C_r by assigning real numbers to the parameters in their respective expressions. In particular, we set $s = \frac{n_2}{50}$ and assign 2 and 5 to q_1 and q_2, respectively. We also set c_{hot} as $3 \times \frac{c}{4}$, which is actually quite conservative and puts IDEA and CVRetrieval at a disadvantage considering that most updates should come from active writers. The analytical results are summarized in Table 5.1.

As shown in Table 5.1, CVRetrieval incurs much lower communication cost than pure consistency maintenance protocols in all three sets of data. This observation indicates that the majority of CVRetrieval's overhead comes from the consistency maintenance of writers, which validates our hypothesis that, by separating observers from writers, the consistency control overhead can be substantially reduced.

Additionally, the overhead of CVRetrieval increases in a slightly slower speed than those of Deno and IDEA when the number of updates increases (reflected by

the number of active writers). Comparing the results of set 1 and set 3 and we can see that the overhead of CVRetrieval in set 3 is 2.8 times as large as that in set 1, while that ratio is 4.1 for Deno and 3.1 for IDEA respectively. We believe that this is an indication that CVRetrieval scales better than the other methods.

5.4 Experimental Results

We implement a prototype of CVRetrieval on top of the Planet-lab and use it to evaluate the performance of CVRetrieval. More specifically, we want to evaluate the response time of CVRetrieval in comparison with other consistency maintenance protocols. This measurement is important because it determines how fast an end user can perceive a certain level of consistency and naturally the faster the response time, the higher the user's satisfaction.

For a consistency maintenance protocol, the response time is defined as the time difference between the point when an update of an object is first committed and that when a participant receives that update (with a certain level of consistency guarantee). In the case of CVRetrieval, however, the response time has different definitions for writers and observers. For writers, the definition of response time is the same as that in a consistency maintenance protocol. For observers in CVRetreival, however, the response time is between the point of time when an observer issues a retrieval request for a consistent view of an object and that when it receives the view.

5.4.1 Experiment Setup

We emulate a white board application for evaluation purposes. The application is emulated by following its operational sequences. More specifically, we abstract the distributed white board as a set of objects that are replicated on each participating

node. Then, we treat each update on the white board (from the writers group) as a write operation on its local replica. After updates are issued, IDEA works to maintain the overall consistency level of the virtual white board above a certain degree. Because our purpose of the experiments is to evaluate the consistency control, we assume that these updates are all conflicting with one another (otherwise, users need not care about them).

In the current setting, each writer informs its publisher when it becomes or ceases to be an active writer. The publisher then informs its subscribers (those who subscribe on behalf of their clients)periodically. Through the publish/subscribe infrastructure, subscribers get the list of the active writer group, which, with a very high probability, have the most consistent view of the shared object. For observers, they specify and inform their subscribers about their interest. The subscribers, based on the information received from publishers (the active writer list), choose a nearby active writer as the source for retrieval purposes. When an observer is not satisfied with the retrieved view, it can issue a "retrieval" request directly to the subscriber to retrieve the most recent consistent view.

We conduct the experiment on the Planet-Lab test-bed. In the current setting, there are ten writers among which four are active writers and the other six are passive ones. There are one publisher and four subscribers. Each subscriber serves three observers. In other words, this is a 22-node system, excluding the publisher and subscribers.

During the experiment, each active writer issues one update every 5 seconds until the experiment ends. These updates got disseminated among active writers immediately and, once it starts to propagate to passive writers, each hop will only disseminate the updates once every 5 seconds (to save bandwidth by combining multiple updates). We let each observer retrieve the consistent view every 20 seconds during

Type	Max (seconds)	Min (seconds)	Average (seconds)
active writer	1.73	1.41	1.59
passive writer	11.8	10.2	10.98

Table 5.2: Response time for writers

the experiment. The experiment runs 300 seconds.

We also implemented a Deno-like protocol for comparison. In the Deno-like protocol, we organize the 22 participants (here, we don't consider the publisher and subscribers as participants because they are only facilitating CVRetrieval) in a linear fashion in which the updates are propagated from one to the other. To make the results comparable, we assume the same updating patterns for the ten writers.

5.4.2 Response Time for Writers

We measure response times for active writers and passive writers respectively. The experiment was run ten times and the average response time, as well as maximum and minimum values, are measured and shown in Table 5.2.

From the result, we can see that the response time of active writers is very small. This is because the dissemination of updates is instant among active writers. While it is usually very costly to disseminate update instantly among participants, CVRetrieval can afford to do so because, via differentiation, there are only a relatively small number of active writers in existence.

As shown here, the average delay for passive writers is over 10 seconds, which looks rather high. However, this is because we set a five-second delay between any two consecutive dissemination of updates among passive writers. In practice, system administrators can choose a shorter delay to improve the response time for passive writers at the expense of increased bandwidth overhead.

Cache hit rate	Max (seconds)	Min (seconds)	Average (seconds)
50%	0.48	0.33	0.37
66.7%	0.3	0.24	0.28
75%	0.16	0.12	0.14

Table 5.3: Response time for observers

5.4.3 Response Time for Observers

There are two aspects of response time for observers, namely, the time that it takes for them to receive the periodically published updates and the time it takes for an observer to get the most up-to-date view through an explicit retrieval operation. Because the first of delay primarily depends on the publishing rate, we do not measure it here.

The delay of explicit retrieval depends on whether the observer can find the view in its subscriber's local cache (because another observer retrieved the same view a moment ago). Intuitively, the more retrievals can be satisfied with the subscriber's cache (a higher cache hit rate), the smaller the response time is. In this experiment, we give three settings of the cache hit ratio: 50%, 66.7%, and 75%. For each setting, we run ten experiments and the results are summarized in Table 5.3.

The result shows that the retrieval process is indeed very efficient and this efficiency increases with cache hit rate in subscribers.

5.4.4 Comparison to a Deno-like Consistency Maintenance Protocol

We now compare the performance of CVRetrieval with a pure consistency maintenance protocol. For a pure consistency maintenance protocol, we assume that all participants are treated equally. In terms of update dissemination, there are two types: active ones that disseminate a received update to other participants as soon as it

Max (seconds)	Min (seconds)	Average (seconds)
2.45	1.77	2.07

Table 5.4: Response time of a pure consistency maintenance protocol with active update dissemination

arrives and passive ones that only periodically disseminate all the updates it received so far. Because the passive ones work similarly to the way CVRetrieva/IDEA treats passive writers, but with more participants, it is doubtless that CVRetrieval/IDEA will have a better performance. For this reason, we only experimentally compare CVRetrieval to the active ones.

The consistency maintenance protocol we considered here is Deno-like and thus has all the 22 participants (not including the publisher and subscribers because they are add-on features of CVRetrieval) we used in the CVRetrieval evaluation. Because this protocol actively disseminates updates, each participant relays a received update as soon as it is received. Finally, the writers have the same updating patterns as in previous experiments. We run this experiment ten times and the results are shown in Table 5.4.

From this table, we can see that the response time of the pure maintenance protocol is larger than that of CVRetrieval's active writers (comparing to the data in Table 5.2). However, the absolute value of the response time is not that large. We suspect that this is because, due to the heavy load of planet-lab nodes, the write operation alone needs too much time to be committed. To validate our hypothesis, we profile one run of the experiment and record the response time for all 21 participants (this does not include the writer who committed this update) and the result is depicted in Figure 5.6.

From this figure, we can clearly see that the first hop delay dominates the system's response time. While this figure is for CVRetrieval, we expect that Deno has the same

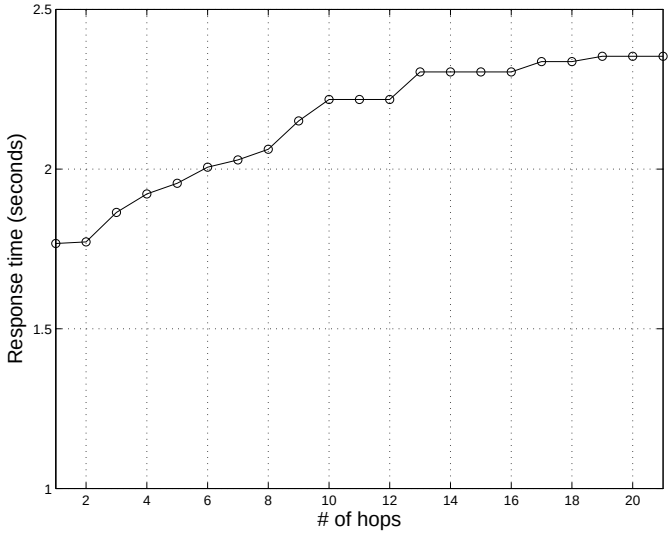


Figure 5.6: Response time for different hops

first hop delay because the write operation should cost the same amount of computing power for different protocols. Since, if the write operation takes much longer than inter-node communication delay, it is the write operation that determines the overall response time, we expect the advantage of the CVRetrieval approach to be much more pronounced when the experimentation is run on lightly loaded computers that can minimize the cost of committing updating operations.

It is worth noting that most current protocols use passive update dissemination method, with which the advantage of CVRetrieval will become more pronounced. Furthermore, the most important advantage of CVRetrieval is its saving of communication cost, especially in a system with a large number of participants, as analyzed in Section 5.3. We believe that the two features–efficiency and scalability–together make CVRetrieval a viable alternative to pure consistency maintenance protocols.

	White Board	Online Game	Bulletin Board	E-Business
Collaboration type	Synchronous	Synchronous	Asynchronous	Asynchronous
# of active writers	Small	Large	Small	Small
# of passive writers	Medium	Large	Large	Medium
# of observers	Large	Large	Large	Medium
# of shared objects	Small	Large	Small	Medium
Order preservation	Low	Low	High	High
Low latency	High	High	Low	Medium
Correctness	Low	Medium	High	Very high

Table 5.5: Four representative distributed online collaboration applications

5.5 Discussion of Targeted Applications

CVRetrieval targets on Internet-scale distributed online collaboration applications. In this section, we discuss four representative distributed online collaboration applications, as summarized in Table 5.5 that lists some key application characteristics. Through this discuss, we show that there are a large range of applications that can benefit from CVRetrieval.

White board. In this application, participants appear online at the same time to collaborate. For example, a group of people can draw on the same white board to communicate with one another.

Online gaming. Online gaming has become more and more popular. Popular games, such as World of Warcraft [104] and SecondLife [89], have hundreds of thousands of paid subscribers. While people usually think games as leisure activities, they have in fact become a common platform for people to interact. A recent *Business Week* cover story reported that people earn big real money from online gaming, such as SecondLife, and even traditional companies are entering the gaming arena for both

advertisement and direct sale [35].

Bulletin board. People read the post in a bulletin board and post messages when they want to communicate with one another.

E-business. Take as an example, an airline ticket booking system that is handled by multiple servers. The content of an update should be the same on all sites. In e-business applications, the writers are indeed the servers that handle transactions because they issue updates on behalf of the customers.

In Table 5.5, we characterize qualitatively the four applications based on eight metrics. First of all, white board and online gaming are synchronous collaborations because the participants usually appear online at the same time; bulletin board and e-business are asynchronous collaborations because the participants come and go: people post on the bulletin board and check back later; in e-business, each business center has different request pattern, thus issuing updates asynchronously [17].

The number of active writers for a given object is large in online gaming because, due to the usually large number of participants, even a small portion of them acting as active writers can result in a large number comparing with that of other three applications. The white board application has a small number of active writers because there are usually a small number of participants who lead the collaboration (think the lecturers in the remote education scenario). The other two applications have small numbers of active writers because, due to the nature of an asynchronous collaboration, not all writers are active at the same time. The numbers of passive writers in white board and e-business are medium comparing with that of online gaming and bulletin board for the following reasons. For the white board application, the total number of participants is usually small because it is intended for a moderate group of people to communicate, hence the small number of passive writers. For e-business applications, as mentioned before, the servers that handle transactions are the writ-

ers. Thus, at least currently, the number of these servers is not large, so we conclude that the number of passive writers will be small too.

All applications, except for e-business, have a large number of observers. In e-business, unlike the other three applications in which participants communicate with one another and a lot of people care about the perceived qualify of consistency, only members of the relatively small management team are interested in the consistency of the e-business application. Finally, the number of shared objects is large in online gaming as modern games become more and more sophisticated. White board and bulletin board have smaller numbers of shared objects than that of e-business because these two applications serve special features for sharing (*i.e.*, dedicated white board and bulletin board) while it is possible for an e-business application to run multiple types of transactions at the same time.

In terms of the consistency requirement, according to TACT [106], a pioneer research on defining metrics to quantify consistency degrees, there are three metrics that can be used to evaluate and quantify consistency in this context: (1) Delay-a remote update should appear on a participant's site with very little delay; (2) Order preservation-multiple updates should appear on all the sites in the same order; and (3) Correctness-the content of an update should be the same on all sites, including the site where it originated.

For our targeted applications, white board and online gaming have relatively low requirement for order preservation (comparing to the other two applications) because people prefer fast responses and are usually willing to figure out the errors by themselves. For example, a recent study of online gaming shows that in chat room people prefer to receiving conversation word-by-word (faster speed) than receiving the finished sentence at once (slower speed) [10]. For the same reason, white board and online gaming require low latency than bulletin board and e-business. With the in-

creased emphasis on accuracy, for example, an error in e-business can cost a lot of money for the company, the requirement for correctness increases from the left to the right side in Table 5.5.

Note that CVRetrieval can potentially benefit all these four applications for the following reasons. First, the white board, online gaming, and bulletin board applications all have a large number of observers, which makes reducing the consistency control overhead for these observers a significant improvement in scalability for the system as a whole. Second, while the absolute number of observers is small in e-business in contrast to those of other three applications, it is still larger than, or at least comparable to the number of writers (both active and passive ones that are essentially the e-business servers). Thus, CVRetrieval can also benefit an e-business application in that it can significantly reduce the consistency control overhead.

5.6 Summary

In this chapter, we discussed the design and evaluation of CVRetrieval (CVRetrieval). CVRetrieval recognizes that, in a variety of applications, a large number of participants are not actively collaborating for most of the time. CVRetrieval then separates active participants from passive ones. More specifically, CVRetrieval uses traditional consistency maintenance protocol–in this case, IDEA–to actively maintain consistency for active participants, and uses an efficient, low cost, consistency retrieval mechanism to satisfy passive participants' consistency need on-demand.

CVRetrieval is evaluated by both theoretical analysis and implementation. The analytical study showed that CVRetrieval incurs significantly less communication overhead than other consistency maintenance protocols. The results from prototyping on the Planet-Lab test-bed showed that CVRetrieval achieves shorter response times

for active participants at the expense of a longer response time for passive participants.

Chapter 6

FairOM: Fair Overlay Multicast

In this chapter, we present FairOM (Fair Overlay Multicast). Targeting an Internet-scale distributed system, an important design goal of overlay multicast protocols is to achieve fairness among participants to encourage participation. However, current overlay multicast protocols only consider fairness from a single multicast session's point view, thus unable to support multiple simultaneous multicast sessions. To improve the performance of overlay multicast, FairOM redefines the definition of fairness and, based on the new definition, provides an ability to support multiple simultaneous multicast sessions.

In this chapter, we first discuss the motivation of FairOM and formulate the problem. Afterward, we discuss the design of FairOM. FairOM is then thoroughly evaluated in three steps. First, we compare FairOM and non-FairOM approaches theoretically to show the relative advantages of FairOM. Second, the simulation results of a large multicast group with 1000 nodes show that FairOM achieves the goal of enforcing proportional contributions among nodes and does not overwhelm any of these nodes. The simulation also shows that FairOM achieves low delays for nodes and attains high path diversity. Third and finally, we investigate FairOM's practica-

bility by running a prototype of FairOM on 40 Planet-Lab nodes that span US and Canada; the results show that FairOM functions well in a real environment.

6.1 Motivation

In Internet-scale distributed systems, such as the large-scale Grid and Planet-Lab, reliable and efficient data dissemination plays an important role, with examples ranging from the massive data distribution to multimedia delivery. In these systems, overlay multicast [14, 20, 110] is a better choice than IP level multicast for several reasons. First, overlay multicast does not need the special configuration of network routers that are often not enabled by system administrators due to security reasons. Second, it can be configured at the application level, thus providing opportunities to capture the semantics of the applications. And finally, it is easy to use and configure in practice.

The biggest challenge in applying overlay multicast to an Internet-scale environment is to meet the nodes' requirement of fairness [14]. In these environments, no node is supposed to contribute dramatically more or less than others. The conventional single-multicast-tree structure does not satisfy the fairness requirement as the leaves in the tree have no contribution to the multicast effort while the interior nodes contribute all the forwarding bandwidth [14]. To tackle this problem, the notion of multicast forest, or multiple multicast-trees, has been explored in several studies [14, 73]. A good example of these systems is SplitStream [14], which builds a multicast forest and ensures that on average each node only serves as an interior node once (as a contributor in one tree) and is a receiver in all other trees.

However, even if we have a multicast forest in which each node contributes some (by being an interior node in one multicast tree, for example) and no node is over-

whelmed, is there any chance that the multicast is still unfair in the sense that it results in relatively poor performance? We argue that simply letting each node contribute once and satisfying each node's outgoing bandwidth constraint is not enough for enforcing fairness for the sake of performance.

Performance-wise, enforcing proportional contribution provides an environment to support multiple simultaneous multicast sessions that may not otherwise be achievable by simply asking every node to contribute arbitrarily. Consider the following example in which nodes A and B are both going to multicast a movie and each multicast will span all the nodes in the network. Suppose that A builds its multicast forest first and one node, C, is assigned to contribute 90% of its outgoing bandwidth to it. Then when B tries to establish its multicast forest, chances are that C just does not have enough bandwidth to support it because it has contributed too much to the first multicast session. In this case, the construction of a forest for B becomes either infeasible or, barely feasible by saturating C's outgoing bandwidth and making C a hot-spot/bottleneck. In this case, if we instead let each node contribute roughly the same percentage of its outgoing bandwidth, say 20%, then C has a chance to support the two multicast sessions simultaneously.

Thus we believe that a better way to define fairness is to enforce the requirement that nodes' contributions be proportional to their total available outgoing bandwidths, which is analogous to taxation or donation. In taxation or donation, it is desirable for people to give the same percentage of their available capital as their contributions to the society (here, we assume all the people are in the same tax bracket).

To this end, we present FairOM (Fair Overly Multicast) that enforces proportional contributions among nodes through a two-phase forest construction process.

It is noteworthy that, in terms of bandwidth constraint, we are only concerned

with the outgoing bandwidth. The reasons are twofold. First, current broadband technologies, such as ADSL, have limited outgoing bandwidth on each node that is typically smaller than its incoming bandwidth. Second, each node should always have enough incoming bandwidth to accept all the incoming data otherwise it cannot benefit from multicasting.

FairOM is useful when there is sufficient bandwidth available so that it can utilize the bandwidth effectively for multiple sessions. In this scenario, without FairOM, there is no guarantee that multiple sessions can be supported (due to the in-proportionality); with FairOM, however, there is a high probability that multiple sessions can be supported. So we do need FairOM when there is sufficient bandwidth. The bottom line is that sufficient bandwidth does not guarantee the support of multiple session, although it is a requirement.

On the other hand, if the network bandwidth is so constrained that all the bandwidth is needed, there is no need to concern about the proportionality. However, with the proliferation of wireless-enabled laptops and high-speed Internet connections, we believe that there will be certain amount of excessive bandwidth available in large-scale distributed systems in the near future that should be effectively exploited to benefit the overall performance.

6.2 Problem Formulation

We represent each node's total outgoing bandwidth as its total contribution capacity. Then, we make the following three assumptions:

- Each data package to be multicast is encoded into n equal sized stripes and each node has enough incoming bandwidth to absorb all the n stripes. This is essential to successfully build a multicast forest because otherwise the receiver

cannot receive all the stripes no matter what multicast scheme is used.

- The total available outgoing bandwidth of nodes is sufficient to build a forest to multicast data to the nodes. Again, this assumption is to make the forest building feasible.

- There is excessive outgoing bandwidth in this multicast group. While this assumption is not essential to the correctness of the protocol, it provides the opportunity to show its advantages. If there is little excessive bandwidth left, all nodes will have to contribute almost all their capacities, thus reducing to a special case of this protocol and making it identical or similar to other schemes.

Before we state our design goal, let us first formally define several terms with the assumption that there are a total of n nodes in this multicast group.

- T_i**:** Total available outgoing bandwidth for each node i, or, the maximum number of stripes it is capable of forwarding.

- C_i**:** The forwarding load of node i, in term of the number of stripes it is assigned to handle.

- R_i**:** defined as $\frac{C_i}{T_i}$, is the contribution ratio of node i.

- $StdR$: The standard deviation of the contribution ratios (R) of all the n nodes. That is,

$$StdR = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(R_i - \bar{R})^2} \tag{6.1}$$

A complete multicast forest must satisfy the following two conditions:

- Multicast satisfaction: each node should receive all the n stripes.

- Bandwidth limitation: the forwarding load of each node i should be less than or equal to its total available outgoing bandwidth, or $C_i \leq T_i$.

The design goal of FairOM is to minimize the standard deviation of all the nodes' contribution ratio $StdR$ in a complete multicast forest.

Goal: minimize $StdR$.

6.3 Design of FairOM

FairOM assumes that a new node knows at least one other member in the current multicast group when it joins, implying that FairOM does not directly deal with the bootstrap mechanism. Further, FairOM assumes that all the nodes know when the forest construction starts and the number of trees they need to join. In practice, the source and the nodes can exchange this information through web page announcements or emails. As well, a node can learn this information from its neighbors.

The workflow of FairOM protocol is illustrated in Figure 6.1. FairOM has two major steps: warm up process and forest construction. During the warm up process (step 1), nodes establish neighborhood, which builds local information for each node. Then the source initiates the forest construction process, which in turn is done in two phases (sub-steps): building a forest skeleton (step 2-1) and making the forest complete (step 2-2).

A salient feature of FairOM is the enforcement of proportionality. To enforce proportional contribution, we propose and use a Staged Spare Capacity Group (SSCG) as a key data structure for nodes to track their contribution levels so that proportional contributions can be enforced. In SSCG, "staged" means proportionality. SSCG is an improvement over SCG (Spare Capacity Group) that has been proposed in [14] because SCG does not track nodes' contribution levels while SSCG does.

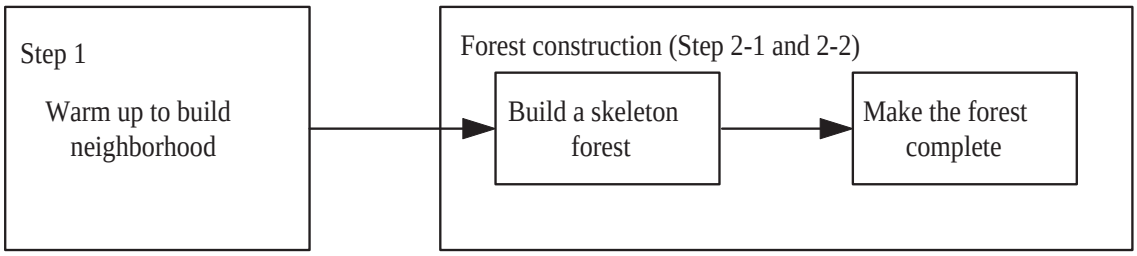


Figure 6.1: Workflow of FairOM

In the rest of this section, we first discuss our proposed SSCG data structure. Then we discuss the main body of FairOM protocol: the warm up process and the two phases of the forest construction. Other related issues are then discussed.

6.3.1 Staged Spare Capacity Group

Staged spare capacity group is a key data structure in FairOM to enforce proportional contributions. Suppose that the spare capacity group has five stages, where each stage represents a percentage range of the capacity (*e.g.*, stage 1 represents [0%, 20%], stage 2 (20, -40%], etc), then the source will put each of the registered nodes into an appropriate stage. To illustrate this concept, we consider a simple example as illustrated in Figure 6.2.

In Figure 6.2, suppose that node A has a total outgoing bandwidth of 20 (*i.e.*, it can forward 20 stripes of data) and has already contributed 3 units of the total, then its current contribution is 15% (3/20). Because A's contribution is less than 20%, it is put into stage 1. B is put into stage 2 because its contribution is in the range (20%, 40%]. Follow the same criteria, C and D are put in stages 1 and 5, respectively.

It is worth noting that the source maintains an independent staged spare capacity group for each stripe (each multicast packet will be divided into several stripes and each stripe will be multicast along a different multicast tree). So if a node has a contribution of more than one stripe, it needs to register the contribution information

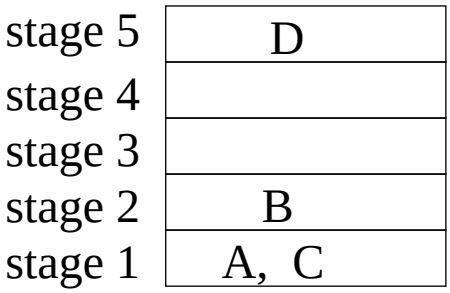


Figure 6.2: Layout of the staged spare capacity group for A, B, C and D while the contributions of them are 15%, 25%, 10% and 82%, respectively.

for each stripe independently. Clearly, the space complexity of the source's SSCG is a linear function of the number of nodes. However, we don't expect this to be a burden for the source as shown by the following example. Suppose there are 1000 nodes, each session has 16 stripes, and each registration takes 1KB. Then the total storage cost is only 16MB, which can be easily put into its main memory. Nevertheless, the storage overhead can be further reduced by the distributed algorithm that we will propose in Section 6.3.7.

6.3.2 Establishment of Neighborhood

After joining the multicast group, a new node will eventually establish its neighbor list by running a periodical neighborhood establishment procedure. In each round of this procedure, the node contacts its neighbors (there is at least one bootstrap neighbor by assumption) and checks this neighbor's neighbor list. If its neighbor's neighbors do not appear in this node's own neighbor list, it acts as follows. When its neighbor list is not full (each node defines the length of its neighbor list), it puts the new nodes into its neighbor list. Otherwise, it compares the new nodes with the ones already in its neighbor list according to the routing latency between the nodes and itself. If a new node has smaller latency, this node replaces a current neighbor by the new one with a certain probability (currently we use 0.8) to prevent hot spot.

If this node adds a new node to its neighbor list, it sends a notice to the new node about this.

This way, each node will establish its own neighbor list after a certain number of rounds. After that, this periodical process serves as a way to adjust nodes' neighbor lists and to maintain the neighborhood among nodes.

6.3.3 Phase I of Forest Construction: Initial Construction

Now we describe the first phase of the multicast forest construction among all the nodes. The purpose of the initial forest construction is by no means to build a complete forest. Instead, it servers as a good start and provides a skeleton. Then the second phase can improve the skeleton and eventually complete the forest building. This is a quota-driven system in which the quota represents the designated percentage of a node's contribution (out of its total available bandwidth), and the system has a predefined initial quota (*i.e.*, a predefined percentage of contribution). As well, each node is willing to contribute as much as it can within this predefined quota.

More specifically, the source first sends out short forest-establishing request messages, one for each data stripe to be multicast, which are then forwarded to different neighbors to achieve path diversity. For each node that receives a request, it forwards the request to as many neighbors as it can within the predefined quota. If a node receives multiple transmission requests of the same stripe, it picks one and treats the sender of it as its multicast parent for that particular stripe, and rejects others. At this stage, let us simply assume that a node picks the node that notifies it first. We have more discussion on this issue in Section 6.3.5.

When a multicast relationship between a parent and a child has been established (the parent picks the child and the child accepts it), the parent and the child both record this relationship locally. Then they both start to run a heartbeat checking

procedure to detect any failure. For each node, when it receives a stripe request and has gotten all the responses from the children candidates it picked, it calculates its contribution and registers it in the staged spare capacity group maintained at the source by sending a message to the multicast source.

There is no clear line between the first phase and the second one in a global scale. Instead, it is each node's own decision on when to start the second phase that we describe next.

6.3.4 Phase II of Forest Construction: Making the Forest Complete

After the forest building process starts, each node checks with those nodes that treat it as a neighbor (recall that a node sends a notice to its neighbor after the neighbor is added to its neighbor list in the neighborhood establishment procedure). If all nodes it contacts have already gotten some stripes and did not choose it as a child in the initial forest construction, it will seek help from the source.

In the message it sends to the source (here, source means multicast source, which is the sender of the data in the multicast), the node indicates the number of times it has requested for spare capacity, starting with one (the first time). When the source receives the message (with number one), it only looks for parents for this node in the first stage of the spare capacity group by randomly picking one eligible parent that has this stripe. Then it updates the parent's contribution. If the new contribution ratio is beyond the quota limit of its stage (20% for the first stage), the parent's record is moved from the current stage to the next higher one (stage two in this case).

If a parent is found, the parent will receive the adoption request of a potential child from the source and then send a request to the potential child that needs to be adopted. Thus the node with missing stripes can get what it wants.

If the source cannot find a parent in this stage, the node with missing stripes will wait a predefined period of time before it starts the next round of request. When the next round starts, the node sends another message that has a request number of two, which tells the source to search for an adoption up to stage two. However, the source still starts to look for adoption from the first stage in hope to find some new spare capacity from smaller contributors. Thus each round provides opportunities for the source to find a node with smaller contribution ratio to adopt.

We need to mention that the delay within each round of request for spare capacity and the way the source looks for adoption (always starts from the first stages, and climbs to higher stages gradually) are essential to the effectiveness of FairOM to enforce proportional contributions because they provide an opportunity to utilize resources from newly joined smaller contributors.

To prevent a node from waiting forever, a node contacts the source anyway if a predefined deadline, which represents a maximum waiting time, has passed (see the description below).

By following this protocol, the source relaxes the quota gradually and finally builds a complete forest in which every node is in all trees. A pseudo code of this process is illustrated in Algorithm 1. In this code, num_of_try is a system parameter. We set the num_of_try at 5 because it worked well and did not cause too much delay in both simulation and implementation.

Node:

```
if (current_time > deadline) and (has not received all stripes) then
    num = 1;
    while num_of_try ≤ 5 do
        send_to_source(id of all missing stripes, num);
        wait(waiting_time);
        if still has missing stripes then
            num++;
        end
        else
            return success;
        end
    end
    return failure;
end
```

Source:

```
while true do
    recv_from_nodes(id of all missing stripes, num);
    foreach missing stripe do
        parent = find_adoption_up_to_level(num);
        if parent ≠ NULL then
            send_to_the_chosen_parent(node_info, id of the missing stripe);
            if parent.contribution > limit of the current stage then
                move parent to the next higher stage in SSCG;
            end
        end
    end
end
```

Algorithm 1: Pseudo code for making the forest complete

It is worth noting that this algorithm does not guarantee the complete forest construction 100% because we treat the forest construction as an optimization problem. Our goal is to build a complete forest with a very high probability. In the case where the algorithm fails to build a forest, we can always retry. The up-side of this approach is that, without needing to guarantee a 100% probability, the algorithm is fast. If we want to guarantee a 100% probability, the protocol may need longer time to run and/or consume more communication cost.

6.3.5 Incorporating Multicast Delay Information into Consideration

The algorithm discussed so far does not consider multicast delay when it performs the forest construction. In this section, we try to minimize the multicast delay of the forest by incorporating the delay information into each of the two phases of the forest construction. Here, the delay perceived by a node with regard to a particular stripe is defined as the time delay for it to receive the stripe from the source. This delay information at each node can be obtained in the following manner: first, the source has zero delay; second, the delay for an immediate child of the source is its communication latency from the source, which can be measured by timestamping techniques; third, all the other nodes' delay can be measured by adding a node's parent's delay and the communication latency between the node and its parents. Performance-wise, the shorter the delay, the better the performance is perceived by this node.

In the initial forest construction process, each node sends its delay information along with the message it sends to its neighbors. When a node receives multiple transmission requests of the same stripe, it picks the one with the smallest delay and drops others. This way, cycles are prevented from being formed in the multicast tree.

Consider three nodes A, B and C. Suppose that A chooses B as its parent, B chooses C as its parent, and C chooses A as its parent. In this case, delay D_a of A is larger than delay D_b of B, that is, $D_a > D_b$. Similarly, we have $D_b > D_c$ and $D_c > D_a$, implying that $D_a > D_a$, which is impossible. Thus the scheme to incorporate delay information is cycle free.

In the second phase of the forest construction, when a node requests the missing stripes from the source, the source chooses several parents for this child (we current use three) and the child then chooses the one that incurs it the smallest delay.

6.3.6 Handling Node Join and Departure

When a node joins the multicast group after the forest has been built, it first establishes its neighbor list by following the neighborhood establishment procedure described in Section 6.3.2. Then it seeks for adoption from its neighbors. Upon receiving this request, a neighbor grants the request if this adoption will still keep itself in its current stage in the spare capacity group to ensure that it still only contributes its fair share of the multicast (thus preserving the proportionality). Otherwise, it rejects this request. If no neighbor is willing to adopt the new node, it contacts the source for spare capacity and the source will pick a parent for it.

In the case of node departure, we differentiate two kinds of departures: decent departures and failures. For a decent departure, the departing node notifies its multicast parents and children so that the parents can reclaim their contribution and its children can start seeking for adoptions by first contacting their neighbors and then contacting the source if none of their neighbors grant their requests. In the case of a failure, this failure will eventually be detected by the heartbeat checking procedure that is run between each multicast parent-child pair. After the failure is detected, the failed node's parents and children can react accordingly, similar to the case of a

decent departure.

6.3.7 Distributed Algorithm for the Second Phase

The second phase of the forest construction described in Section 6.3.4 is centralized in nature, where the source is responsible for managing the staged spare capacity group. One concern for this approach is the stress put on the source. While the evaluation results of Section 6.5 show that the stress will not overwhelm the nodes, including the multicast source, we nevertheless propose a distributed forest construction algorithm (for the second phase, that is) that can further alleviate the stress put on the multicast source and make the process more scalable.

The basic idea behind the distributed protocol is that, in the forest construction process, after the source sends out the stripe requests, a group of nodes work together to help one another (by adopting other nodes in the forest) so that they can eventually and quickly form a complete multicast forest. This way, the responsibility of completing forest construction is shifted from the source to all the multicast members.

To enforce proportionality, each node maintains its own data structure and tracks the contribution information of its neighbors and its neighbors' contributions will be refreshed periodically via neighbor-wise communication. Thus, the staged spare capacity group becomes distributed in nature. To prevent nodes from contributing much more than their neighbors, a node only contributes if its current contribution is not within the top $x\%$ among those of its neighbors. For example, if $x\%$ is 20% and the node has four neighbors, the node will not contribute if its contribution ranks the first among the 5 nodes. We call this $x\%$ rule. However, the $x\%$ rule can potentially fail when all nodes have the same contribution. To cope with this, a node first checks whether this is the case before it applies the $x\%$ policy. If this is the case, the node will contribute, thus breaking the tie. The feasibility of this distributed algorithm–the

probability that it results in a complete FairOM forest–is analyzed as follows.

Feasibility of the distributed algorithm

Here we give an analysis about this distributed algorithm, which shows that it can build a multicast forest where each node has proportional contributions with high probability. Here we assume that there is sufficient bandwidth to render the forest construction feasible.

Assume that a node A is missing a stripe j and fails to find a node to adopt it through a demand tree, it will ask its neighbors for help. If a particular neighbor fails to adopt it, this neighbor will forward this request to another neighbor, and so on. Because we assume that the network is a connected graph, the node will eventually find a node that has the particular stripe. Now there are two possibilities: (1) It accepts the node; (2) It refuses to accept this node because its contribution is much higher than its neighbors (*i.e.*, the $x\%$ rule), an event whose probability is denoted as P_r. Suppose a node has an average of I neighbors and the nodes' contributions are randomly and uniformly distributed, we have:

$$P_r = x\% \tag{6.2}$$

Suppose that there are N nodes in the multicast group and n nodes that have stripe j. Then node A fails to get stripe j when all the n nodes fail to accept it. We denote the probability of this event as P_{f-one}. Thus we have:

$$P_{f-one} = (P_r)^n \tag{6.3}$$

Assume that there are k stripes for each session, then the number of the missing stripes seeking adoptions is at most $N \times k$, that is, each of the N nodes is seeking

the k stripes. Since the probability of any event P or event Q occurring is less than or equal to the probability of P plus the probability of Q, the upper bound of the failure probability of the forest construction is thus:

$$Upper\_bound = N \times k \times P_{f-one} \tag{6.4}$$

We denote the probability of the failure of forest construction as $P_{f-total}$, and we have:

$$P_{f-total} \leq N \times k \times (\frac{x}{100})^n \tag{6.5}$$

The probability of failure is indeed very small even with a small number of neighbors and a small n. For example, the probability of failure is less than 0.17% when $x\% = 20\%$ and n=10, while N=1000 and k=16. While k is usually predefined and is independent of the network size, n and N are highly related. In practice, in a network as large as 1000 nodes, we expect n–the number of nodes that have a particular stripe–to be much larger than 10. Thus we expect that the probability of failure is even lower in a real environment, so the probability of building a FairOM forest should be larger than 99.83%.

6.3.8 Discussions about the Security Issue

With the assumption that the nodes are trustworthy, FairOM performs well and finally builds a fair-sharing multicast forest. However, when some nodes do not follow the rule and cheat on their contributions, the multicast forest would not be fair any more. To prevent this from happening, a distributed audit mechanism, such as that proposed by Ngan *et. al* [70], can be deployed to detect cheating and to remove the node that cheats from the multicast group. Further discussion of this issue is beyond

the scope of the current FairOM and we plan to explore this issue in the future.

6.4 Analytical Comparison between FairOM and Non-FairOM Approaches

To evaluate the advantages and disadvantages of FairOM scheme, we analyze FairOM and compare it with non-FairOM approaches from two aspects: tree height and the number of sessions that can be supported simultaneously. These are two important metrics because they measure the performance of FairOM from a single-session's and multiple-session's point of view respectively.

The tree height measures the performance of FairOM from a single-session's point of view because it determines the maximum delay perceived by end users, thus presents the worse case scenario. In this section, we analyze the upper bound and lower bound of this measure for FairOM and non-FairOM schemes. We also propose two mechanisms to push FairOM towards the lower bound in the run time to minimize the multicast delay.

The number of multicasting sessions FairOM and non-FairOM can support measures the capacity of these multicasting protocols. Intuitively, the larger the number is, the more useful and effective the protocol is in a real environment. This metric is important because, due to the increasing complexity of modern computer systems, more and more systems need to support multiple tasks simultaneously.

6.4.1 Assumptions and Definitions

In this analysis, we assume a forest-based overlay multicasting scheme. Basically, a forest-based scheme builds multiple trees (hence the name *forest*) for a multicasting session and each tree covers the whole multicasting group. Suppose that there are n

trees in each session, then each packet will be divided into r stripes and each tree is responsible for delivering one stripe. Usually, a node only needs to receive m stripes ($m < n$) to reconstruct the r stripes when certain coding scheme, such as erasure code [9], is used. In this analysis, we make the following assumptions and definitions that will be used throughout this section.

1. The total number of nodes in the overlay network is n and the multicasting should cover all the n nodes. The n nodes are denoted by $N = N_1, N_2, \ldots, N_n$.

2. Total available bandwidth of nodes, in terms of number of stripes, are $T = T_1, T_2, \ldots, T_n$.

3. The number of sessions is denoted as m and the m sessions are denoted as $S = S_1, S_2, \ldots, S_m$.

4. There are r stripes in each session.

   Because the FairOM protocol is designed to enforce proportional contributions, it ensures that each participant contribute a fixed percentage, $a\%$, of its total available bandwidth for one session. Then we introduce the parameter $a\%$ so that

5. Each node in FairOM contributes $a\%$ of its total available bandwidth for each session.

6.4.2 Analysis of Tree Height

We analyze the tree height of the FairOM and non-FairOM schemes. However, because it is very difficult, if not impossible, to accurately model the topology of a large-scale network, we only derive the upper and lower bounds of the tree height

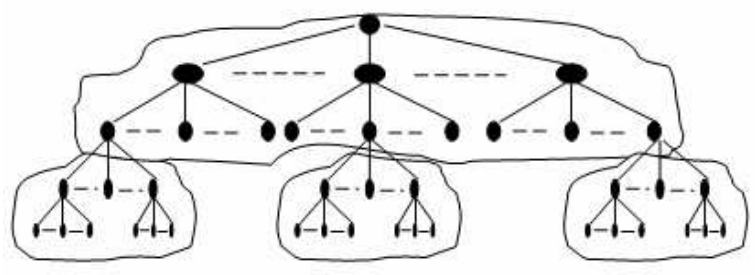


Figure 6.3: Approximate best case scenario, used to derive the lower bound of tree height

instead of the average. Nonetheless, we believe that the upper and lower bounds provide a sensible measurement for the tree height in general.

Best case and worst case scenarios

Before we can analyze the lower and upper bounds, we first clarify the best and worst case scenarios.

In the best case, nodes are organized as a balanced tree, as shown in Figure 6.3. In the worst case, the nodes are organized in the fashion as shown in Figure 6.4. While we understand that, in a truly worst case scenario, the tree could be linear (each node has only one child, and so on), it may not always be the case. To be general, we let a node bear a parameter of the number of children, which may not be exactly one. As shown in Figure 6.4, an interior node can potentially have more than one child but the main structure is still linear (if we remove all the leave nodes). The best and worst case scenarios apply to both FairOM and non-FairOM approaches. We use the two scenarios for the analysis of both FairOM and non-FairOM approaches.

Analysis of tree height

First of all, we classify the nodes with different numbers of children as follows. We use n_i to denote the number of nodes with q_i ($q_i \in T$) children. Further, we suppose that

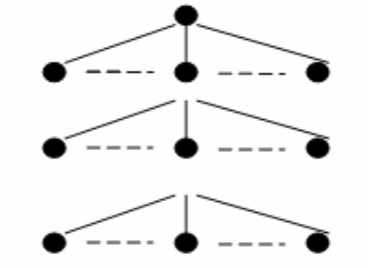


Figure 6.4: Approximate worst case scenario, used to derive the upper bound of tree height

there are k types of capacities. Because the multicasting covers the whole network, we have

$$\sum_{i=1}^{k} n_i q_i = n \tag{6.6}$$

Based on the theory of balanced tree, if all interior nodes have the same number of r children, the tree height is $\log_r^n$. For simplicity, we approximate the lowest tree height, h_{min}, as the tree height if we restrict the nodes' capacity to be the same as the average capacity, represented by q_min_{avg}, as shown in Figure 6.3. Thus in the case of non-FairOM approaches, we have

$$h_{n\_min} \approx \log^n_{q\_min_{avg}} \tag{6.7}$$

In the worst case, as shown in Figure 6.4, the tree height is determined by the number of interior nodes, so we have:

$$h_{n\_max} = \sum_{i=1}^{k} n_i \tag{6.8}$$

Suppose that, in the worst case scenario, the average number of children of all the interior nodes is q_max_{avg}, which is an abstract value used by the analysis. Then from Formula 6.6 we have

$$\sum_{i=1}^{k} n_i q\_max_{avg} = q\_max_{avg} \sum_{i=1}^{k} n_i = n$$

we have

$$\sum_{i=1}^{k} n_i = \frac{n}{q\_max_{avg}}$$

Thus, we finally have

$$h_{n\_max} = \sum_{i=1}^{k} n_i = \frac{n}{q\_max_{avg}} \tag{6.9}$$

Now we consider the case of FairOM. In this case, each node is only allowed to allocate at most $a\%$ of its total available bandwidth for each session, hence for each stripe. According to the FairOM protocol, q_min_{avg} and q_max_{avg} would be $a\%$ of those values in formula 6.7 and 6.9 because nodes' capacity in FairOM is $a\%$ of their capacity in non-FairOM. For this reason, the lowest and highest tree height can be expressed as:

$$h_{f\_min} \approx \log^{n}_{a\% \times q\_min_{avg}} \tag{6.10}$$

$$h_{f\_max} = \frac{n}{a\% \times q\_max_{avg}} \tag{6.11}$$

According to the FairOM protocol, $a\% \times q\_max_{avg}$ in Formula 6.11 must be larger than 1. Thus we can roughly compare the tree height between the FairOM and non-FairOM approaches. The main result is as follows.

$$ratio_{min} = \frac{h_{f\_min}}{h_{m\_min}} = \frac{1}{1 + \log^{a\%}_{q\_max_{avg}}} \tag{6.12}$$

$$ratio_{max} = \frac{h_{f\_max}}{h_{m\_max}} = \frac{1}{a\%} \tag{6.13}$$

From the above equations, we can see that both $ratio_{min}$ and $ratio_{max}$ will be always larger than 1 because the log operation always results a negative value because $a\% < 1$ and $q\_max_{avg} > 1$. Hence, no matter how big or small $a\%$ and q_max_{avg} are, the ratios are always larger than 1. This means FairOM results a larger tree height, hence long propagation delay and worst performance. The size of $a\%$ and q_max_{avg} only affects how worst FairOM could be.

To get a numerical sense about the two ratios, we set $a\%$ as 25% and q_max_{avg} as 16. The results are that $ratio_{min}$ equals to 2, and $ratio_{max}$ equals to 4.

Pushing the tree height toward the lower bound

From this analysis, we can see that FairOM has larger tree height compared with non-FairOM protocols, for both the lowest and highest height. Being aware of that, we propose two mechanisms here to push the tree height of FairOM toward the lower bound. Our intention is that, by pushing toward the lower bound, FairOM can achieve the best performance within its framework. However, the proposed schemes have not been implemented into our FairOM prototype due to time constraint. The proposed mechanisms are as follows.

Preventing linear structure The basic idea is to prevent the worst case, the linear structure, from happening as early as possible. With this proposed mechanism, FairOM monitors the tree construction process and, whenever a linear structure is discovered, it will modify the construction process on the fly, probably by randomly picking other nodes as children rather than the ones that have been chosen.

Threshold-controlled optimization The first mechanism continuously optimize the tree height, but it can hurt the performance of FairOM potentially because it could slow down the forest building process. To strike a balance between forest building speed and the final tree height, we can use a threshold that defines a range close to the lower bound. The optimization process is active when the current projected tree height (*i.e.*, the expected final tree height) is beyond the range and is inactive when the current projected tree height is within the range.

6.4.3 Analysis of the Protocols' Capacity

Now, we show the advantages of the FairOM approach over the pure forest-based approach that does not take proportionality into consideration. For simplicity, we assume that all sessions have the same bandwidth requirement (they have the same number of stripes and all stripes have the same bandwidth requirement). As discussed previously, we suppose that each node in FairOM uses $a\%$ of its bandwidth for each session, then given the node with the smallest capacity, denoted as T_{min}, in terms of the stripes it can forward, the following constraint must apply since a stripe is the smallest unit of transmission: $T_{min} \times a\% \geq 1$ *and* T_{min} *is an integer*. Therefore, the number of sessions that FairOM can support, denoted as NF, is:

$$NF = \lfloor \frac{100}{a} \rfloor \leq \lfloor T_{min} \rfloor \tag{6.14}$$

The metric of comparison is the probability that a non-FairOM approach can support the same number NF of sessions as FairOM. The hypothesis is that, if a non-FairOM approach is very unlikely to match the number of sessions FairOM can support, it will have an even smaller probability to support more sessions than FairOM.

For a non-FairOM approach, we examine a given node i that has a bandwidth of T_i. For each session, the contribution of node i in terms of the number of stripes it forwards is an integer between one and T_i, since in a non-FairOM approach a node contributes arbitrary amount of its capacity. Suppose a non-FairOM approach can support NF sessions, we denote its contribution to the NF sessions as $C_1, C_2, \ldots, C_{NF}$. Because each contribution has T_i options, the number of all possible combinations is:

$$Total_i = T_i^{NF} \tag{6.15}$$

In all these combinations, not all the possible combinations satisfy the requirement that the sum of all contribution is less than or equal to T_i–to make the forwarding load within node i's capacity. We assume that the number of feasible combinations (*i.e.*, combinations that can make the forest feasible) is F_i. To calculate F_i, we treat T_i as T_i 1s (for example, if $T_i = 10$, we say T_i has 10 1s, meaning the node's bandwidth can be divided into 10 units, with each unit corresponds to one stripe forwarding) and NF contributions as NF bins. Because contribution has to be at least 1, we first pick NF 1s and put them to the NF bins to make them non-empty, then randomly put the remaining 1s to the NF bins. Thus, C_i is determined by the placement of the $(T_i - NF)$ remaining 1s. Because C_i has at most $(T_i - NF)$ options, we have

$$F_i \leq (T_i - NF)^{NF} \tag{6.16}$$

In this formula, $(T_i - NF)$ is positive because of formula 6.14. Clearly, we have $F_i < Total_i$. Then the probability P_i that node i can support NF sessions is:

$$P_i = \frac{F_i}{Total_i} \leq (\frac{T_i - NF}{T_i})^{NF} \tag{6.17}$$

Suppose that there are n nodes in the multicasting group and P_{max} is the maximum

value of P_i $(i = 1, 2, \ldots, n)$, the probability P_{All} that a non-FairOM approach can support the same number of sessions as FairOM is:

$$P_{All} \leq (P_{max})^n \tag{6.18}$$

Please notice that P_{All} depends on the value of n that is usually very large. Even when P_{max} is very close to 1, P_{All} can still be very small with even a small number of n. For example, when P_{max} is 0.99 and n is 500, P_{All} is 0.0066. Thus, FairOM has clear advantages over pure forest-based approaches in terms of the maximum number of multicasting sessions that it can simultaneously support.

6.5 Experimental Results

We use both simulation and prototype deployment to evaluate FairOM, for different purposes: the simulation is used to investigate the scalability of FairOM and the prototype deployment is used to show its practicability. In the simulation, we choose the Transit-Stub model [109] to simulate a physical network. The Transit-Stub model generates 1452 routers that are arranged hierarchically, like the current Internet structure. Then we generate 1,000 end nodes and attach them to routers randomly with uniform distribution. Further, our simulator models nodes' bandwidths by assigning each node a number that equals the maximum number of stripes it can forward, which serves as the node's total outgoing bandwidth. For all the simulations, each node's total outgoing bandwidth is randomly chosen between 10 and 20. For the following experimentations, we use five stages of spare capacity and each stage accounts for 20% of contribution.

We then deploy a FairOM prototype on 40 Planet-Lab nodes that span US and Canada to investigate its effectiveness in a real environment. Finally, for the second

phase of the forest construction, we use the centralize algorithm in all simulations and use the distributed algorithm in our Planet-Lab experiment. This is because, after investigating the effectiveness of the centralized algorithm through simulation, we switched to the distributed algorithm when we implemented the FairOM prototype. Nevertheless, we do not expect this difference to impact the conclusions that we will make.

In the following evaluation, we chose different sets of parameters in different experimentation for different purposes. To investigate how FairOM performs in terms of the number of stripes, we use three configurations. For the following evaluations, we do not expect the conclusions to change with different number of stripes. Thus we run FairOM with less number of configurations in the following evaluations.

6.5.1 Effectiveness of Enforcing Proportional Contributions

We measure the effectiveness of enforcing proportional contribution by $StdR$, the standard deviation of the nodes' contributions ratios. In this simulation, we run three configurations with numbers of stripes being 2, 4 and 8, respectively.

In all the simulations, the algorithm satisfies the requirement to build a complete forest and satisfies all nodes' bandwidth constraints. Then the mean value and $StdR$ are calculated and summarized in Table 6.1. The results can be interpreted as follows.

First, we should look at StdR, not mean, as the performance benchmark because our design goal is trying to minimize StdR, as stated in the problem statement section. The mean value will always go up with the number of stripes because more stripes implies more data to be forwarded, hence higher contribution ratios for all nodes.

Second, with number of stripes changes from 2 to 8, the StdR becomes larger and larger, meaning that FairOM performs worse. This makes sense because: (1) with each stripe, there will be some over- and under-contribution happen which is due to

Statistics	FairOM (2)	FairOM (4)	FairOM (8)
Mean	0.131	0.257	0.521
StdR	0.047	0.090	0.106

Table 6.1: Mean and *Std* of contribution ratios

the fact that contributions are discrete value (for example, you can contribute one stripe or two stripe, but not 1.2 stripes. Thus if, for example, the perfect contribute for a node is 1.4, the node cannot realize this kind of contribution and have to settle on 1, the closed discrete value); (2) with more stripes, the errors add up, resulting larger StdR.

Third, as just mentioned, there is error in each stripe. So StdR probably will increase linearly if we run each stripe separately. However, FairOM considers all the stripes together and, based on the results shown in Table 6.1 that the increase of StdR slows down (0.043 when increase from 2 to 4, and only 0.016 when increase from 4 to 8). Thus we speculate that FairOM cancels some errors (for example, let a node who under-contributes in one stripe over-contribute in another). So this result clearly shows that FairOM performs very well when we change the number of stripes from 2 to 8.

6.5.2 Forest Construction Overhead

To evaluate the overhead of the forest construction in FairOM, we use the number of messages received by each node during the forest construction phase. In the relevant literature, this metric is also denoted as "node stress".

In a typical run, all the nodes, except the source, have node stress less than 300. The node stress for the source is 6585. While 6585 appears extremely high compared with other nodes' stress, it is amortized during the whole forest construction and it does not induce much bandwidth cost for the source as shown in the following

analysis.

In the experiment, the duration of the forest construction was 192 seconds. Let us conservatively assume that each message is of size 1KB, which, according to our experience, is much larger than really needed. These 6585 messages amount to a total size of 6.6MB and receiving these packets in 192 seconds requires a bandwidth of 34.4KB per second, which is the incoming bandwidth overhead. However, a more important factor of bandwidth cost is the outgoing bandwidth overhead because most internet connections, such as ADSL, provide much larger incoming bandwidth than outgoing bandwidth. From the outgoing bandwidth's point of view, there is even less data traffic involved at the source during the forest construction phase because certain messages, such as spare capacity registration, do not require response but account for a large portion of total messages received by the source. So this should not be a burden for a media server that usually has high-speed Internet connections.

6.5.3 Multicast Performance

Figure 6.5 shows the cumulative distribution of nodes' delay in a typical run with 4 stripes. In this figure, a point (x, y) indicates that a fraction y of all the nodes have delays of less than or equal to x. In this simulation, all nodes receive all the stripes within 8 seconds and the average delay is 4.1 seconds. We believe that this is a reasonable performance because all the nodes receive the message within similar amount of time, thus the multicast forest being roughly evenly shared by all the nodes.

6.5.4 Path Diversity of the Multicast Forest

Path diversity refers to the diversity between the paths from each node to the multicast source. Ideally, the paths should be disjoint with each other so that one node's

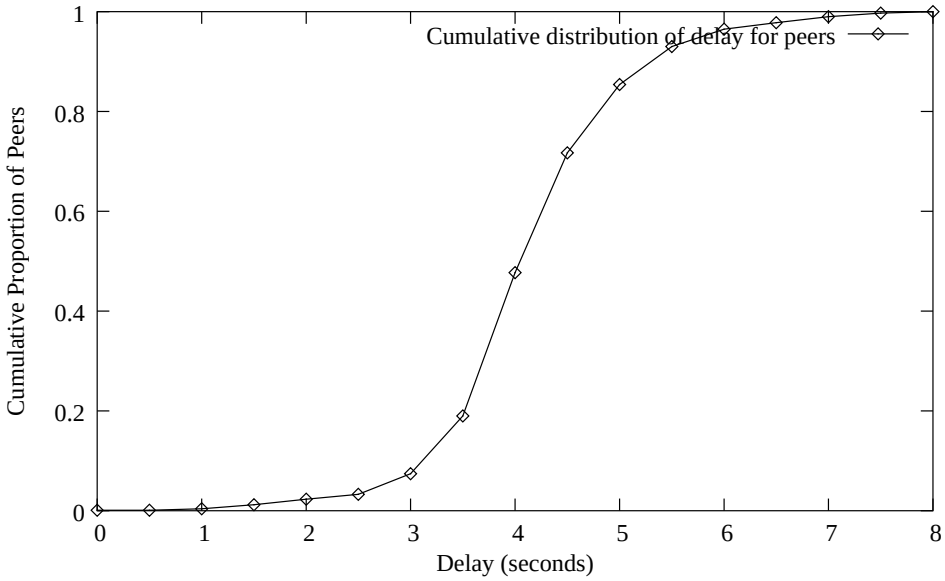


Figure 6.5: Cumulative distribution of delay for nodes

Statistics	FairOM (4)	FairOM (8)
Max	2	3
Mean	1.02	1.66
Median	1	1

Table 6.2: Max, mean and median number of lost stripes with a single node failure

failure only causes the loss of one stripe for the receiver. Without proper path diversity strategy, if we fail one node and this node serves as an interior node for four stripes, for example, four stripes will be lost (all node under the failed node will not receive the data). On the other hand, if we fail one node, at least one stripe will get lost. So it will be impressive to achieve close to one stripe loss.

In the simulation, we randomly fail one non-source node in the multicast group. We run the simulation with two configurations. First, we run FairOM with four stripes and then FairOM with eight stripes. The result is shown in Table 6.2.This result validates the effectiveness of the path diversity of FairOM, which successfully builds such a forest where one node's failure only costs the loss of a small number of stripes.

Setting	1 session, 4 stripes without FairOM	1 session, 4 stripes with FairOM
Standard deviation of R_i	0.222964	0.13309

Table 6.3: Planet-Lab results

While FairOM mainly uses randomization to achieve diversity, the enforced delay between each round of requests for spare capacity also contributes to the path diversity because it provides opportunities for a receiver to get stripes from different parents.

6.5.5 Planet-Lab Results

Now we run the prototype of FairOM on 40 Planet-Lab nodes that span US and Canada. We run two settings in this experiment: First, we run 1 session of 4-stripe multicasting task without the FairOM mechanism; second, we run 1 session of 4-stripe multicasting with the FairOM mechanism.

In this experiment, we we choose to run four stripes only because we don't expect the conclusions to be any different from the run with two or eight stripes. As well, we did not compare that because we have established that FairOM can support more session than non-FairOM approaches in the analysis and this experiment is to validate the analytical results so we only run one configuration.

The performance metric we used is the standard deviation of all nodes' contribution ratio (R_i). The result is summarized in Table 6.3.

From Table 6.3, we can see that the FairOM mechanism clearly has an advantage because the standard deviation is almost half of that without the FairOM mechanism.

6.6 Summary

In this chapter, we presented FairOM (Fair Overlay Multicast). Motivated by the observation that current fairness scheme only considers one multicast session, FairOM redefines the definition of fairness so that fairness can also translate to the ability to support multiple simultaneous multicast sessions.

FairOM is evaluated by analysis, simulation, and implementation. The results show that FairOM can support much more simultaneous multicast sessions than other protocols without sacrificing significant multicast performance. FairOM also incurs minimal communication overhead.

Chapter 7

Conclusions and Future Work

In this dissertation, we proposed IDF and CVRetrieval to improve data consistency management, and FairOM to improve overlay multicast in Internet-scale distributed systems. This chapter concludes this dissertation by highlighting its main contributions and discussing work that we would like to pursue in the future.

7.1 Conclusions

In the past decade, the Internet has become the dominant platform for doing research and business. With this new platform, data consistency management and overlay multicast are the key issues to improve data management and data delivery.

The current data consistency management schemes deploy predefined consistency levels before systems start to run and cannot change the consistency levels on the fly. This scheme cannot effectively support multiple applications to run simultaneously with different consistency requirements, nor does it support applications whose consistency requirements change from time to time.

In the context of overlay multicast, enforcing fairness among participants is an important research issue because, in Internet-scale distributed systems, participants

have equal status and usually do not want to contribute much more than others. While different fairness schemes have been proposed in the literature, they only consider one multicast session and cannot effectively support multiple sessions.

To improve data consistency management, we have presented IDF and CVRetrieval; to improve overlay multicast, we have presented FairOM. The main contributions of this dissertation are summarized below.

7.1.1 Improved Adaptivity in Consistency Control

Conventionally, data consistency is maintained by specifying a predefined consistency level before a system starts to run. This strategy works well for a small scale system that run certain types of applications. However, Internet-scale distributed systems impose two challenges. First, the unreliable communication channels make it almost impossible to maintain a well defined consistency level. Second, it is not uncommon for a modern computer system to run multiple applications simultaneously and these applications can have different consistency requirements.

All these challenges call for an adaptive consistency control scheme. In this dissertation, we have proposed IDF to provide this functionality [53, 54, 58, 59, 60]. IDF advances the state of the art in consistency control research by using a unified framework to simultaneously support multiple applications with different consistency requirements and variable consistency requirements of a given application, and providing the ability to adaptively adjust consistency levels perceived by users on the fly based on application semantics.

IDF lets a system start running and, in the mean time, relies on an efficient inconsistency detection mechanism to detect any inconsistencies. Based on a two-layer infrastructure, this detection mechanism is able to capture most inconsistencies (>90% in a variety of scenarios [58]) with minimal delays. On top of the inconsistency

detection mechanism we have built IDEA, an adaptive consistency control protocol [59, 60]. IDEA adjusts the consistency level on the fly through interaction with users. Upon the detection of inconsistencies, IDEA resolves the detected inconsistencies if the current consistency level does not satisfy applications' requirements; otherwise, IDEA will not resolve the inconsistencies except when the system is lightly loaded. The advantages of this approach are twofold: it can adjust the consistency level on the fly by resolving inconsistencies on demand; and more importantly, it gives the users the ability to control their perceived consistency level.

7.1.2 Improved Scalability in Consistency Control

Most consistency control protocols directly and explicitly involve all participants when enforcing consistency. The drawback of this approach is its huge communication cost when the system is large and participants are of large number and distributed. Based on the observation that, in an Internet-scale distributed system, not all participants are equally active or engaged in an online collaboration application at a given time, we have proposed CVRetrieval to separate active participants from passive ones and use different strategies to satisfy each group's consistency requirement [56].

More specifically, CVRetrieval uses a conventional consistency maintenance protocol, such as IDEA, to actively maintain consistency for the relatively small number of active participants. For the large number of passive participants, CVRetrieval has proposed an efficient publish-subscribe mechanism to satisfy the passive participants' consistency requirement on-demand.

CVRetrieval advances the state of the art of consistency research in two ways. First, it recognizes the existence of the large number of passive participants in Internet-scale distributed systems and, through the separation of active participants and passive participants, has reduced the communication overhead for consistency control

and thus improved the scalability of consistency control. Second and more importantly, the separation of active and passive participants in CVRetrieval is dynamic, meaning that the active and passive participants are relative concepts and can change from time to time. The dynamism enables CVRetrieval to cope with an ever changing system that is not unusual in an Internet environment.

7.1.3 Improved Fairness and Performance in Overlay Multicast

The enforcement of fairness is important in Internet-scale overlay multicast because it encourages participation. Knowing that they will not contribute significantly more resource than others in a cooperative overlay multicast, participants will have more incentive to join the system.

The current fairness scheme requires that each participants to contribute once in one multicast session. While this guarantees that each participant will contribute something in one multicast session, it does not answer the question of what will happen when multiple multicast sessions are present. Based on our analysis, the current fairness scheme cannot guarantee the support for multiple simultaneous multicast sessions. With more and more applications run on the same network simultaneously, we believe that it would be advantageous to support multiple simultaneous multicast sessions while ensuring fair contributions among participants. However, current overlay multicast protocols do not provide this ability.

In this dissertation, we have proposed FairOM [52, 55, 57] to support multiple simultaneous multicast sessions while enforcing fair contributions. The novelty of FairOM is that, instead of treating fairness simply as letting each participant contribute once, it redefines fairness as such that each participant, for one particular multicast session, contributes the same proportion of its available outgoing band-

width. This way, FairOM advances the state of the art of overlay multicast research by achieving both improved fairness and performance.

7.2 Future Work

In the course of designing, implementing, and evaluating IDF, CVRetrieval, and FairOM, we have found several interesting issues related to data consistency management and overlay multicast that we would like to pursue in the future.

7.2.1 Data Consistency Management in Large-Scale Mobile Networks

IDF and CVRetrieval target Internet-scale distributed systems, in which only a limited number of participants are mobile. While this assumption is valid for the current corporate organizations, it is believed that in the near future more and more people will rely on mobile devices to communicate [42, 65]. The facts point to the direction of a large-scale mobile computing network.

Internet-scale distributed mobile networks pose two challenges for data consistency management. First, mobile devices have limited battery power and must disconnect from the network from time to time to conserve energy or otherwise recharge, which results in frequent network topology changes. Second, wireless network links have lower network bandwidth than wired network and wireless channels are less stable as well, which prohibits high communication overhead [12].

We believe that, with certain enhancement, IDF and CVRetrieval can be used to support these large-scale mobile networks. First, to tackle the issue of a frequently changing network topology, we can cluster a small group of mobile devices together and let the elected cluster head/leader track the dynamically changing topol-

ogy within the cluster. In the case of IDF, the cluster head shall report the updating activities of its group nodes; in the case of CVRetrieval, cluster head can subscribe updating information on behalf of the whole group and then inform any new members who have recently joined or returned.

Second, to further the reduce incurred network bandwidth overhead to suit the requirement of mobile networks, IDF can work on two levels. On the inconsistency detection level, we can let each mobile device decide when and how to report their updating activity, instead of reporting these activities as they occur. In this way, a mobile device can choose to report less information and less frequently when its network bandwidth is limited, and report more information and more frequently otherwise. On the inconsistency resolution level, the current IDEA protocol depends on active writers, which may be overwhelmed (the active writers are possibly mobile devices too). IDF can instead either let the one who issues updates handle the responsibility of inconsistency resolution or choose some stable super-node to serve as agents to take up this responsibility. To reduce the incurred network bandwidth overhead for CVRetrieval, we can possibly use a prediction model to predict the optimal publishing rate and automatically reduce the rate when the bandwidth becomes limited.

With these strategies, IDF and CVRetrieval can be potentially applied to large-scale mobile networks as well.

7.2.2 Standardization for Data Consistency Protocol Development

Data consistency can assume different meanings in different application domains. That is, for different applications, consistency means different things and their consistency levels cannot be measured with a single formula. In an increasingly converged

world in which more and more applications run on the same platform (the Internet), standardization is needed to solve the data consistency management problem with a unified framework.

Previous research, such as TACT [106], has made limited effort to tackle this issue by proposing metrics to quantify consistency levels for certain applications. In this dissertation, we also made efforts towards this direction by capturing the inconsistencies for a variety of application types with a unified and efficient inconsistency detection mechanism.

However, to further improve data consistency management and foster collaboration, standardization is needed. In our opinions, there are three aspects that need to be standardized: the definition of consistency level for applications, a communication format that is used to exchange consistency requirement among consistency control protocols, and a benchmark to evaluate the effectiveness of different consistency control protocols.

First, we need to standardize the measurement of consistency levels. This standard will help applications communicate with their supporting consistency control protocols. The metric proposed by TACT, which uses a <*numerical error, order error, staleness*> triple to indicate the consistency level of a replica, is a viable starting point for this purpose.

Second, we need to standardize a communication format for exchanging consistency requirements between different consistency control protocols. This is important because it can facilitate automatic translation of consistency requirement. For example, let us suppose that there is an application, A, that runs on one platform and its consistency is maintained by protocol P_a. Then, when there is a platform change and A now runs on another platform and is supported by protocol P_b. Without a communication format standard, the consistency requirement of application A has

to be *manually* translated from the format of protocol P_a to the format that can be understood by protocol P_b. If we have such a standard, this requirement translation can be done automatically; this can then ease the process of platform change.

Third, we need to agree on a benchmark to evaluate the effectiveness of different consistency control protocols. This benchmark, if developed and agreed upon by the research community, can quantitatively measure the research progress.

7.2.3 FairOM+: The Next Stage of FairOM

One key assumption of FairOM is that all multiple sessions cover the same multicast group. While this is true in a typical multimedia dissemination application, there are scenarios in which different multicast sessions can have overlapping but not identical multicast groups. For example, participants may be interested in different sets of movies in a multimedia application. Also, in distributed computing, different computing sites may need different sets of data. To extend FairOM, we propose FairOM+ as the next stage of the FairOM protocol to handle those scenarios. In this section, we first illustrate the application background of overlapping but not identical multicast groups and then present the design challenges and principles of FairOM+.

Applications

FairOM+ strives to support multiple overlapping but not identical multicast groups, which has a rich application background. In this section, we illustrate three such applications–multimedia, distributed computating, and message–oriented systems- and explain how FairOM+ can support them.

Multimedia In multimedia applications, with increasingly available bandwidth, people soon will be able to download multiple movies at home simultaneously [1].

For example, several family members can, through one single Internet connection, download different movies to watch. From the system's point of view, the same subscriber (the single shared Internet connection) joins multiple multicast sessions (the multiple movies). Because of different preferences of different families (*i.e.*, their family members), the families form multiple overlapping yet not identical multicast groups and each multicast group corresponds to a single movie title. Without careful design, it is very hard, if not impossible, to support this kind of application because even a single hot spot in the multicast tree can hurt the overall Quality of Service (QoS). FairOM+ can improve the collective performance (making the multicast tasks feasible) by giving incentive for members to contribute their available bandwidth to help others.

Distributed computating In a large-scale distributed computing environment, such as the Grid, multiple sites cooperate with one another to finish a parallel computing task, in which different data sets are needed by different sites. When multicast is used to disseminate the data sets, these multiple computing sites form overlapping yet not identical multicast groups. In this scenario, the service providers can first call for interest from all the computing sites and, based on the reply (that states which data sets a site is interested in), use FairOM+ to construct a feasible multicast scheme (*i.e.*, multicast forest) to send the data sets to the computing sites.

Message-oriented Systems In a message-oriented publish-subscribe system [3, 6, 71, 81], participants subscribe to the service that they are interested in and the publishers send the updates to the subscribers. There are two kinds of publish-subscribe schemes: subject-based and content-based.

In subject-based publish-subscribe schemes [6, 71, 81], the publishers multicast any updates to the multicast group that covers all subscribers. Because different

participants have different interests, it is not hard to imagine that different multicast groups have overlapping but not identical participants. In this case, FairOM+ can naturally be used to build a feasible multicast scheme.

In content-based publish-subscribe schemes [3], however, multicast is not usually used. Instead, the messages are relayed in a rather peer-to-peer fashion and the content filter that is deployed at each node decides whether to relay the message further or not. While simple, this can cause serious performance problem such as congestion and congestion control mechanism is needed in a real environment [78]. While the congestion control scheme works fine, using a multicast protocol such as FairOM+ can certainly help because, while it is not feasible for FairOM+ to promise a congestion free service, the congestion problem can be dramatically alleviated. Also, in the presence of long-living subscription, the cost of planning ahead (building a multicast forest with FairOM+) is amortized throughout the whole life cycle of the subscription service. Thus we believe that FairOM+ is also valuable in content-based publish-subscribe services, especially the long-living subscription services.

Design challenges and principles

FairOM+ differs from FairOM in that, instead of letting all participants get the same amount of data by contributing their bandwidth, it tries to use all the collective bandwidth as a whole to satisfy users' different needs. In this scenario, a participant can actually receive less data (it joins a smaller number of multicast sessions) but contribute more (its total available outgoing bandwidth is higher than that of other participants). Clearly, without an incentive, participants are unlikely to join. Thus there are two design challenges: providing incentives for contribution and building a successful FairOM+ forest. We briefly discuss our solutions to the two challenges below.

Based on applications' semantics, different incentives can be adopted. For example, in the case of multimedia dissemination, service providers can charge the participants based on their contribution/benefit ratio: the higher the ratio, the less the participants should pay. Virtual currencies [2, 34], instead of real money, can be adopted as an incentive as well. Now we describe the incentive mechanism in a more specific way.

For a given node, we use c to represent the contributions of that node in terms of the number of stripes it forwards. Here, a stripe is defined as the unit of forwarded data. Then we use b to represent the number of stripes that node has received. In terms of virtual currency, we assume that each stripe is worth k units. Then, that node's total contribution and total benefit, in terms of virtual currency, are:

$$Contribution = c \times k \tag{7.1}$$

$$Benefit = b \times k \tag{7.2}$$

Ideally, we want a node's contribution to be equal to the collective benefit it receives, thus preserving fairness. We use an additional virtual currency, denoted as v, as either compensation (if a node is over-contributing) or charge (if a node is under-contributing). Please note that the additional currency v can be both positive and negative, to indicate compensation and charge. Thus we have

$$c \times k = b \times k + v \tag{7.3}$$

If we look at one node (for example, node i) only, formula 7.3 can be rewritten as:

$$c_i \times k = b_i \times k + v_i \tag{7.4}$$

Because the additional currency v is determined by the in-degree and out-degree in the forest structure, the sum of all the v_i should be 0 (one in-degree for one node is one out-degree for another node). We believe that this is a nice feature because, from the system's point of view, no additional deficit will occur. So the bookkeeping is easy to handle. However, it is desirable that the effect of v be minimized to prevent large payment and charge from occurring, so as not to discourage users' participation. Thus we want to minimize the standard deviation of the additional virtual currencies under the condition that the FairOM+ forest can be successfully built.

More formally, we give the following definitions to facilitate the design of FairOM+.

- C_{i_max}: total available outgoing bandwidth for each node i, or, the maximum number of stripes it is capable of forwarding. This defines the upper bound of node i's contribution.
- k: the number of units of currency a single stripe is worth.
- C_i: the contribution of node i, in terms of the number of stripes it forwards.
- B_i: the benefit of node i, in terms of the number of stripes it receives.
- V_i: the additional virtual currency, calculated from formula 7.4.
- $StdV$: The standard deviation of the currency (V) of all the n nodes. That is,

$$StdV = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(V - \overline{V})^2} \tag{7.5}$$

Thus the design goal of FairOM+ is to minimize StdV under the constraint that the forest is complete.

As the next stage of research and development of FairOM, FairOM+ will extend FairOM to broader application domains. We plan to pursue its design and implementation in the future.

7.2.4 Reliability Issues in Overlay Multicast

Most overlay multicast protocols, including FairOM, primarily target best-effort delivery, for some valid reasons. For example, most popular applications, such as multimedia and online gaming, only needs a good enough quality.

However, in certain application domains, such as scientific experimentation, complete data delivery is required to produce meaningful results. Also, when a business is delivering critical data, even a small percentage of error can cause a huge financial loss.

One straightforward way to ensure reliability is to increase data redundancy. For example, with erasure code [9], the original data size is increased and only a small portion of the coded data is required to reconstruct the original data. For example, 2MB data will become 8MB after the coding and the 8MB will be multicasted to receivers. On the receiver side, however, only 4MB of data is needed to reconstruct the original data. Thus, the multicast protocol can still reliably deliver data with up to 50% (4MB/8MB) data loss rate. The drawback of this scheme, however, is that the potentially huge redundant data that are unnecessarily delivered. In a network with limited bandwidth, this drawback can quickly drain the available network bandwidth, making other web services unavailable.

To eliminate unnecessary data traffics, we can propose a prediction-based selective redundancy strategy. More specifically, we plan to predict the network reliability based on history data and only increase the data redundancy when it predicts that there will be data loss; otherwise, only a moderate redundancy will be used. With this

strategy, FairOM (and FairOM+, for that matter) can support reliable data delivery without straining the network bandwidth.

Bibliography

[1] A broadband utopia. *IEEE Spectrum 43*, 5 (May 2006), 48–54.

[2] Antoniadis, P., and Courcoubetis, C. Market models for P2P content distribution. In *AP2PC: 1st International Workshop on Agents and Peer-to-Peer Computing* (2002), G. Moro and M. Koubarakis, Eds., vol. 2530 of *Lecture Notes in Computer Science*, Springer, pp. 138–143.

[3] Banavar, G., Chandra, T., Mukherjee, B., Nagarajarao, J., Strom, R., and Sturman, D. An efficient multicast protocol for content-based publish-subscribe systems. In *19th International Conference on Distributed Computing Systems (19th ICDCS'99)* (Austin, Texas, May 1999), IEEE.

[4] Begole, J., Rosson, M. B., and Shaffer, C. A. Flexible collaboration transparency: supporting worker independence in replicated application-sharing systems. *ACM Trans. Comput.-Hum. Interact. 6*, 2 (1999), 95–132.

[5] Bharambe, A. R., Rao, S. G., Padmanabhan, V. N., Seshan, S., and Zhang, H. The impact of heterogeneous bandwidth constraints on DHT-based multicast protocols. In *IPTPS* (2005), M. Castro and R. van Renesse, Eds., vol. 3640 of *Lecture Notes in Computer Science*, Springer, pp. 115–126.

[6] Birman, K. P. The process group approach to reliable distributed computing. *Communications of the ACM 36*, 12 (1993), 37–53.

[7] Birnholtz, J. P., and Bietz, M. J. Data at work: supporting sharing in science and engineering. In *GROUP '03: Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work* (New York, NY, USA, 2003), ACM Press, pp. 339–348.

[8] Bjornsson, M. E., and Shrira, L. Buddycache: high-performance object storage for collaborative strong-consistency applications in a wan. In *OOPSLA '02: Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (New York, NY, USA, 2002), ACM Press, pp. 26–39.

[9] Blahut, R., Ed. *Theory and Practice of Error Control Codes*. Addison Wesley, MA, 1994.

[10] Brown, B., and Bell, M. Cscw at play: 'there' as a collaborative virtual environment. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work* (New York, NY, USA, 2004), ACM Press, pp. 350–359.

[11] Brun, J., Safaei, F., and Boustead, P. Managing latency and fairness in networked games. *Communications of the ACM 49*, 11 (2006), 46–51.

[12] Cao, J., Zhang, Y., Cao, G., and Xie, L. Data consistency for cooperative caching in mobile environments. *IEEE Computer 40*, 4 (2007), 60–66.

[13] Carroll, J. L., and Long, D. D. E. The effect of failure and repair distributions on consistency protocols for replicated data objects. In *ANSS '89: Proceedings of the 22nd annual symposium on Simulation* (Los Alamitos, CA, USA, 1989), IEEE Computer Society Press, pp. 47–60.

[14] Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., Rowstron, A. I. T., and Singh, A. Splitstream: high-bandwidth multicast in cooperative environments. In *SOSP* (2003), pp. 298–313.

[15] Castro, M., Jones, M. B., Kermarrec, A.-M., Rowstron, A. I. T., Theimer, M., Wang, H. J., and Wolman, A. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *INFOCOM* (2003).

[16] Çetintemel, U., Keleher, P. J., Bhattacharjee, B., and Franklin, M. J. Deno: A decentralized, peer-to-peer object-replication system for weakly connected environments. *IEEE Trans. Computers 52*, 7 (2003), 943–959.

[17] Chandler, H. E. The complexity of online groups: A case study of asynchronous distributed collaboration. *ACM Journal of Computer Documentation (JCD) 25*, 1 (2001), 17–24.

[18] Chang, T., Popescu, G., and Codella, C. Scalable and efficient update dissemination for distributed interactive applications. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)* (Washington, DC, USA, 2002), IEEE Computer Society, p. 143.

[19] Charles E. Phillips, J., Ting, T., and Demurjian, S. A. Information sharing and security in dynamic coalitions. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies* (New York, NY, USA, 2002), ACM Press, pp. 87–96.

[20] Chu, Y.-H., Rao, S. G., and Zhang, H. A case for end system multicast. In *SIGMETRICS* (2000), pp. 1–12.

[21] ConferenceXP. http://research.microsoft.com/conferencexp/.

[22] Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. Wide-area cooperative storage with cfs. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles* (New York, NY, USA, 2001), ACM Press, pp. 202–215.

[23] Deering, S. E., and Cheriton, D. R. Multicast routing in datagram internetworks and extended lans. *ACM Trans. Comput. Syst. 8*, 2 (1990), 85–110.

[24] Diklic, D., Velayutham, V., Welch, S., and Williams, R. Remote outsourcing services for multiple branch offices and small businesses via the internet. In *LISA '01: Proceedings of the 15th USENIX conference on System administration* (Berkeley, CA, USA, 2001), USENIX Association, pp. 7–14.

[25] Dubois, M., Scheurich, C., and Briggs, F. A. Memory access buffering in multiprocessors. In *ISCA* (1986), pp. 434–442.

[26] Düllmann, D., Hoschek, W., Jaén-Martínez, F. J., Segal, B., Stockinger, H., Stockinger, K., and Samar, A. Models for replica synchronisation and consistency in a data grid. In *HPDC* (2001), IEEE Computer Society, pp. 67–75.

[27] Eugster, P. T., Guerraoui, R., and Handurukande, S. B. Lightweight probabilistic broadcast. In *Conf. on Dependable Systems and Networks* (Gteborg, Sweden, July 2001), pp. 443–452.

[28] Ford, B. Unmanaged internet protocol: Taming the edge network management crisis. *ACM SIGCOMM Computer Communication Review 34*, 1 (2004), 93–98.

[29] Foster, I. IT as fuel for the innovation engine. In *GridToday* (http://www.gridtoday.com/grid/821956.html, August 21 2006).

[30] Foster, I., and Kesselman, C., Eds. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA, 1999.

[31] Garland, I., Teles, L., and Wang, X. Fostering creativity through cross-disciplinary collaboration in an online dance course. In *Conf. on Computer Support for Collaborative Learning* (Palo Alto, California, 1999), p. Article No. 20.

[32] George Chin, J., and Lansing, C. S. Capturing and supporting contexts for scientific data sharing via the biological sciences collaboratory. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work* (New York, NY, USA, 2004), ACM Press, pp. 409–418.

[33] Gharachorloo, K., Lenoski, D., Laudon, J., Gibbons, P. B., Gupta, A., and Hennessy, J. L. Memory consistency and event ordering in scalable shared-memory multiprocessors. In *ISCA* (1990), pp. 15–26.

[34] Golle, P., Leyton-Brown, K., and Mironov, I. Incentives for sharing in peer-to-peer networks. In *EC '01: Proceedings of the 3rd ACM conference on Electronic Commerce* (New York, NY, USA, 2001), ACM Press, pp. 264–267.

[35] Hof, R. D. My virtual life. *Businee Week* (May 1st, 2006).

[36] Horn, P. Autonomic computing: IBM's perspective on the state of information technology. Manifesto, IBM Research, Oct. 2001.

[37] hua Chu, Y., Chuang, J., and Zhang, H. A case for taxation in peer-to-peer streaming broadcast. In *PINS '04: Proceedings of the ACM SIGCOMM workshop on Practice and theory of incentives in networked systems* (New York, NY, USA, 2004), ACM Press, pp. 205–212.

[38] I-MINDS. http://www.i-minds.com/.

[39] J. Jannotti, D. Gifford, K. J., and Kaashoek, M. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the Symposium on Operating System Design and Implementation (OSDI)* (2000).

[40] Jaiswal, S., Iannaccone, G., Diot, C., and Towsley, D. F. Inferring TCP connection characteristics through passive measurements. In *INFOCOM* (2004).

[41] Kephart, J. O., and Chess, D. M. The vision of autonomic computing. *IEEE Computer 36*, 1 (2003), 41–50.

[42] Kibati, M., and Krairit, D. The wireless local loop in developing regions. *Commun. ACM 42*, 6 (1999), 60–66.

[43] Kistler, J. J., and Satyanarayanan, M. Disconnected operation in the coda file system. In *SOSP* (1991), pp. 213–225.

[44] Kosti, D., Rodriguez, A., Albrecht, J., and Vahdat, A. Bullet: high bandwidth data dissemination using an overlay mesh. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles* (New York, NY, USA, 2003), ACM Press, pp. 282–297.

[45] Kostic, D., Rodriguez, A., Albrecht, J. R., Bhirud, A., and Vahdat, A. Using random subsets to build scalable network services. In *USENIX Symposium on Internet Technologies and Systems* (2003).

[46] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Wells, C., and Zhao, B. Oceanstore: an architecture for global-scale persistent storage. In *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems* (New York, NY, USA, 2000), ACM Press, pp. 190–201.

[47] Lamport, L. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Computers 28*, 9 (1979), 690–691.

[48] Legout, A., Nonnenmacher, J., and Biersack, E. W. Bandwidth-allocation policies for unicast and multicast flows. *IEEE/ACM Trans. Netw. 9*, 4 (2001), 464–478.

[49] Levy, R. M., Nagarajarao, J., Pacifici, G., Spreitzer, M., Tantawi, A. N., and Youssef, A. Performance management for cluster based web services. In *Integrated Network Management* (2003), G. S. Goldszmidt and J. Schönwälder, Eds., vol. 246 of *IFIP Conference Proceedings*, Kluwer, pp. 247–261.

[50] Li, F. W., Li, L. W., and Lau, R. W. Supporting continuous consistency in multiplayer online games. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia* (New York, NY, USA, 2004), ACM Press, pp. 388–391.

[51] Liao, C., Martonosi, M., and Clark, D. W. Experience with an adaptive globally-synchronizing clock algorithm. In *SPAA '99: Proceedings of the eleventh annual ACM symposium on Parallel algorithms and architectures* (New York, NY, USA, 1999), ACM Press, pp. 106–114.

[52] Lu, Y., and Jiang, H. Design and evaluation of a new and effective fairness scheme for multicasting in internet-scale distributed systems. In *14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)* (Research Triangle Park, NC, July 2005), pp. 285–286.

[53] Lu, Y., and Jiang, H. A framework for efficient inconsistency detection in a grid and internet-scale distributed environment. In *14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)* (Research Triangle Park, NC, July 2005), pp. 318–319.

[54] Lu, Y., Jiang, H., and Feng, D. An efficient, low-cost inconsistency detection framework for data and service sharing in an internet-scale system. In *IEEE International Conference on e-Business Engineering (ICEBE 2005)* (Beijing, China, October 2005), pp. 373–380.

[55] Lu, Y., Jiang, H., and Feng, D. FairOM: Enforcing proportional contributions among peers in internet-scale distributed systems. In *ISPA* (2005), Y. Pan, D. Chen, M. Guo, J. Cao, and J. Dongarra, Eds., vol. 3758 of *Lecture Notes in Computer Science*, Springer, pp. 1065–1076.

[56] Lu, Y., Jiang, H., and Lu, Y. CVRetrieval: Seperating consistency retrieval from consistency maintenance. In *Technical Report TR-UNL-CSE-2007-0002* (University of Nebraska-Lincoln, January 2007).

[57] Lu, Y., and Li, X. An analytical study of FairOM: A fair overlay multicast protocol for internet-scale distributed systems. In *First IEEE International Workshop on Networking, Architecture, and Storages (IWNAS 2006)* (Shenyang, China, August 2006), pp. 51–52.

[58] Lu, Y., Li, X., and Jiang, H. IDF: an inconsistency detection framework performance modeling and guide to its design. In *Technical Report TR-UNL-CSE-2006-0003* (University of Nebraska-Lincoln, March 2006).

[59] Lu, Y., Lu, Y., and Jiang, H. IDEA: An infrastructure for detection-based adaptive consistency control in replicated services. In *16th International Symposium on High Performance Distributed Computing (HPDC-16)* (Monterey, CA, June 2007), pp. 223–224.

[60] Lu, Y., Lu, Y., and Jiang, H. IDEA: An infrastructure for detection-based adaptive consistency control in replicated services. In *Technical Report TR-UNL-CSE-2007-0001* (University of Nebraska-Lincoln, January 2007).

[61] Luo, J., Hubaux, J.-P., and Eugster, P. T. Pan: providing reliable storage in mobile ad hoc networks with probabilistic quorum systems. In *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing* (New York, NY, USA, 2003), ACM Press, pp. 1–12.

[62] Lynch, N. A., and Shvartsman, A. A. Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts. In *FTCS '97: Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS '97)* (Washington, DC, USA, 1997), IEEE Computer Society, p. 272.

[63] Malkhi, D., Reiter, M., and Wright, R. Probabilistic quorum systems. In *PODC '97: Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing* (New York, NY, USA, 1997), ACM Press, pp. 267–273.

[64] Marques, J. M., and Navarro, L. WWG: A wide-area infrastructure to support groups. In *International ACM SIGGROUP Conference on Supporting Group Work* (Boulder, Colorado, USA, 2001), pp. 179–187.

[65] McMullan, J., and Richardson, I. The mobile phone: a hybrid multi-platform medium. In *IE '06: Procedings of the 3rd Australasian conference on Interactive entertainment* (Murdoch University, Australia, Australia, 2006), Murdoch University, pp. 103–108.

[66] Mills, D. L. A brief history of ntp time: memoirs of an internet timekeeper. *SIGCOMM Comput. Commun. Rev. 33*, 2 (2003), 9–21.

[67] Mitchell, T. Machine learning.

[68] Muthitacharoen, A., Morris, R., Gil, T. M., and Chen, B. Ivy: a read/write peer-to-peer file system. *SIGOPS Oper. Syst. Rev. 36*, SI (2002), 31–44.

[69] Naor, M., and Wieder, U. Scalable and dynamic quorum systems. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing* (New York, NY, USA, 2003), ACM Press, pp. 114–122.

[70] Ngan, T. J., Wallach, D. S., and Druschel, P. Enforcing fair sharing of peer-to-peer resources. In *International Workshop on Peer-to-Peer Systems (IPTPS), LNCS* (2003), vol. 2.

[71] Oki, B., Pfluegl, M., Siegel, A., and Skeen, D. The information bus: an architecture for extensible distributed systems. In *SOSP '93: Proceedings of the fourteenth ACM symposium on Operating systems principles* (New York, NY, USA, 1993), ACM Press, pp. 58–68.

[72] Padmanabhan, V. N., Wang, H. J., and Chou, P. A. Supporting heterogeneity and congestion control in peer-to-peer multicast streaming. In *International Workshop on Peer-to-Peer Systems (IPTPS)* (2004).

[73] Padmanabhan, V. N., Wang, H. J., Chou, P. A., and Sripanidkulchai, K. Distributing streaming media content using cooperative networking. In *NOSSDAV* (2002), ACM, pp. 177–186.

[74] Parashar, M., and Hariri, S. Autonomic computing: An overview. In *International Workshop on Unconventional Programming Paradigms (UPP), LNCS* (2004).

[75] Parker, D. S., Popek, G. J., Rudisin, G., Stoughton, A., Walker, B., Walton, E., Chow, J., Edwards, D., Kiser, S., and Kline, C. Detection of mutual inconsistency in distributed systems. *IEEE Transactions on Software Engineering SE-9*, 3 (May 1983), 240–247.

[76] Pauw, W. D., Krasikov, S., and Morar, J. Execution patterns for visualizing web services. In *Proceedings ACM International Conference on Software*

Visualization (SoftVis 2006) (New York NY, Sept. 2006), ACM Press, pp. 37–45.

[77] Peterson, L. L., Anderson, T. E., Culler, D. E., and Roscoe, T. A blueprint for introducing disruptive technology into the internet. *Computer Communication Review 33*, 1 (2003), 59–64.

[78] Pietzuch, P. R., and Bhola, S. Congestion control in a reliable scalable message-oriented middleware. In *Middleware* (2003), M. Endler and D. C. Schmidt, Eds., vol. 2672 of *Lecture Notes in Computer Science*, Springer, pp. 202–221.

[79] Pinker, E. J., Seidmann, A., and Foster, R. C. Strategies for transitioning 'old economy' firms to e-business. *Communications of the ACM 45*, 5 (2002), 76–83.

[80] Poon, S., and Swatman, P. M. C. A combined-method study of small business internet commerce. *Int. J. Electron. Commerce 2*, 3 (1998), 31–46.

[81] Powell, D. Group communication. *Communications of the ACM 39*, 4 (1996), 50–53.

[82] Prakash, A., and Shim, H. S. Distview: Support for building efficient collaborative applications using replicated objects. In *CSCW* (1994), pp. 153–164.

[83] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Schenker, S. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2001), ACM Press, pp. 161–172.

[84] Romik, D. Stirling's approximation for n!: the ultimate short proof? *AMM: The American Mathematical Monthly 107*, 6 (2000), 556–557.

[85] Rowstron, A. I. T., and Druschel, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg* (London, UK, 2001), Springer-Verlag, pp. 329–350.

[86] Rowstron, A. I. T., and Druschel, P. Storage management and caching in PAST, A large-scale, persistent peer-to-peer storage utility. In *SOSP* (2001), pp. 188–201.

[87] Sandhu, H. S., and Zhou, S. Cluster-based file replication in large-scale distributed systems. In *SIGMETRICS '92/PERFORMANCE '92: Proceedings of the 1992 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems* (New York, NY, USA, 1992), ACM Press, pp. 91–102.

[88] Schuckmann, C., Kirchner, L., Schummer, J., Haake, and M., J. Designing object-oriented synchronous groupware with COAST. In *Proceedings of ACM CSCW'96 Conference on Computer-Supported Cooperative Work* (1996), Language Support for Groupware, pp. 30–38.

[89] Second Life. http://secondlife.com/.

[90] Sivasubramanian, S., Alonso, G., Pierre, G., and van Steen, M. Globedb: autonomic data replication for web applications. In *WWW '05: Proceedings of the 14th international conference on World Wide Web* (New York, NY, USA, 2005), ACM Press, pp. 33–42.

[91] Sivasubramanian, S., Szymaniak, M., Pierre, G., and van Steen, M. Replication for web hosting systems. *ACM Comput. Surv. 36*, 3 (2004), 291–334.

[92] Srivastava, U., Munagala, K., Widom, J., and Motwani, R. Query optimization over web services. In *VLDB* (2006), pp. 355–366.

[93] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2001), ACM Press, pp. 149–160.

[94] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., and Balakrishnan, H. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networks 11*, 1 (2003), 17–32.

[95] Stonebraker, M. Concurrency control and consistency of multiple copies of data in distributed ingres. *IEEE Transactions on Software Engineering SE-5* (1979), 188–194.

[96] Subbiah, A., and Blough, D. M. An approach for fault tolerant and secure data storage in collaborative work environments. In *StorageSS '05: Proceedings of the 2005 ACM workshop on Storage security and survivability* (New York, NY, USA, 2005), ACM Press, pp. 84–93.

[97] Teo, T. S. H., Tan, M., and Buk, W. K. A contingency model of internet adoption in singapore. *Int. J. Electron. Commerce 2*, 2 (1997), 95–118.

[98] Teo, T. S. H., and Too, B. L. Information systems orientation and business use of the internet: an empirical study. *Int. J. Electron. Commerce 4*, 4 (2000), 105–130.

[99] Terry, D. B., Theimer, M. M., Petersen, K., Demers, A. J., Spreitzer, M. J., and Hauser, C. H. Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proceedings 15th Symposium on Operating Systems Principles* (Cooper Mountain, Colorado, Dec. 1995), pp. 172–183. http://www.parc.xerox.com/bayou/.

[100] Vadapalli, A., and Ramamurthy, K. Business use of the internet: an analytical framework and exploratory case study. *Int. J. Electron. Commerce 2*, 2 (1997), 71–94.

[101] VanderMeer, D. E., Dutta, K., Datta, A., Ramamritham, K., and Navathe, S. B. Enabling scalable online personalization on the web. In *Proceedings of the 2nd ACM Conference on Electronic Commerce (EC-00)* (N.Y., Oct. 17–20 2000), ACM, pp. 185–196.

[102] Vilayannur, M., Nath, P., and Sivasubramaniam, A. Providing tunable consistency for a parallel file store. In *FAST* (2005), USENIX.

[103] Vogel, J., and Mauve, M. Consistency control for distributed interactive media. In *ACM Multimedia* (2001), pp. 221–230.

[104] World of Warcraft. http://www.worldofwarcraft.com/.

[105] Yang, Y., and Li, D. Separating data and control: support for adaptable consistency protocols in collaborative systems. In *Proceedings of ACM CSCW'04 Conference on Computer-Supported Cooperative Work* (2004), Dynamic architectures, pp. 11–20.

[106] Yu, H., and Vahdat, A. Design and evaluation of a continuous consistency model for replicated services. In *OSDI* (2000), pp. 305–318.

[107] Yu, H., and Vahdat, A. The costs and limits of availability for replicated services. In *SOSP* (2001), pp. 29–42.

[108] Yu, H., and Vahdat, A. Consistent and automatic replica regeneration. *Trans. Storage 1*, 1 (2005), 3–37.

[109] Zegura, E. W., Calvert, K. L., and Bhattacharjee, S. How to model an internetwork. In *INFOCOM* (1996), pp. 594–602.

[110] Zhuang, S., Zhao, B. Y., Joseph, A. D., Katz, R. H., and Kubiatowicz, J. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV* (2001), ACM, pp. 11–20.