# Secure File Assignment in Heterogeneous Distributed Systems

by

Yun Tian

A dissertation submitted to
the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 4, 2013

Keywords: Security, Performance, Heterogenous, Distributed Systems

## Abstract

There is a growing demand for large-scale distributed storage systems to support resource sharing and fault tolerance. Although heterogeneity issues of distributed systems have been widely investigated, little attention has yet been paid to security solutions designed for distributed systems with heterogeneous vulnerabilities. This fact motivates us to investigate the topic of secure file assignment in heterogeneous distributed systems.

Firstly we propose a secure fragment allocation scheme called S-FAS to improve security of a distributed system where storage sites have a wide variety of vulnerabilities. In the S-FAS approach, we integrate file fragmentation with the secret sharing technique in a distributed storage system with heterogeneous properties in vulnerability. Storage sites in distributed systems are categorized into a variety of different types of storage node based on vulnerability characteristics. Given a file and a distributed system, S-FAS allocates fragments of the file to as many different types of nodes as possible in the system. Data confidentiality is preserved because fragments of a file are allocated to multiple storage nodes. We develop storage assurance and dynamic assurance models to evaluate quality of security offered by S-FAS. Analysis results show that fragment allocations made by S-FAS lead to enhanced security because of the consideration of heterogeneous vulnerabilities in distributed storage systems.

In order to consider performance while providing higher quality of security for large scale distributed systems with heterogeneous features, we develop a Secure Allocation Processing (SAP) algorithm for the S-FAS scheme to improve the security level and consider its performance using the heterogeneous features of a large distributed system. To improve the security, the design of SAP is guided by the experimental results from S-FAS; to improve performance, we not only consider the heterogeneity of the storage nodes and the whole system, but also the heterogeneous features of the requests. The SAP allocation algorithm considers load balancing, delayed effects caused by the workload variance of many consecutive requests, and the heterogeneous features (such as CPU speed and network bandwidth) of the storage nodes in the system.

In order to use practical implementations to demonstrate the ideas on actual systems with real-world applications, we developed a prototype using the multi-threading technique and C language for the S-FAS scheme with the SAP algorithm to guide the file allocation. The prototype is built in the distributed cluster environment with heterogeneous storage nodes, in which the Network File System (NFS) and Linux are installed. We did some experiments on system throughput and testing against real world traces. The evaluation results show that the proposed solution can not only improve the security level, but also improve the throughput and performance of the distributed storage systems with heterogeneous vulnerabilities by using the multi-thread technique.

To further explore the security solution while considering system availability, we propose a solution called Reef by integrating fragment replication into the proposed S-FAS and SAP solution for distributed systems with heterogeneous features. The Reef scheme is extended based on the S-FAS scheme. In the proposed Reef scheme we consider the system failure mode caused by hardware diversity when categorizing the storage nodes into different groups. The storage assurance model for Reef is developed to evaluate the security quality offered by Reef when all fragments have the same replication degrees. Then we developed a secure fragment replication allocation process algorithm called R-SAP illustrating how to use the proposed Reef scheme. The evaluation results show that the proposed Reef scheme and R-SAP algorithm can improve both availability and security for distributed storage systems with heterogeneous vulnerabilities.

Acknowledgments

I would like to acknowledge and thank the many people whom, without their support, this work would not have been possible.

First of all, I would like to express my deep and sincere gratitude to my advisor, Dr. Xiao Qin. Without his wide knowledge, detailed and constructive guidance, generous support and warm encouragement, I would not have completed my PhD study and this dissertation research would have never been possible. His passionate attitude towards research and wonderful personality will have a remarkable influence on my entire career.

I warmly thank Dr. Drew Hamilton and Dr. Cheryl Seals for their valuable advice on my dissertation. The extensive discussions and their insightful comments have significantly helped in improving the quality of this dissertation. I also wish to express my warm and sincere thanks to Dr. Shiwen Mao for serving as the outside reader and proofreading my dissertation.

I have been working in a great research group. I would like to thank the group members: Xiaojun Ruan, Jiong Xie, Zhiyang Ding, Shu Yin, Yixian Yang, Jianguo lu, James Majors, Ji Zhang, Xunfei Jiang, Sanjay Kulkarni, Ajit Chavan, and Tausif Muzaffar who have helped me a lot in my research and study. Working with them is beneficial and pleasant.

I also owe much gratitude to Dr. Daniela Marghitu, Dr. Kai Chang, and Dr. David Umphress for shaping my graduate student career for the better. I would also like to acknowledge the efferts of Ms. Michelle Wheeles, Ms. Jo Ann Lauraitis, and Ms. Carol Lovvorn in helping me keep my school and immigration paper work in order.

Furthermore, I have received warm friendship, generous help and patient mentoring from some other students also from the Department of Computer Science and Software

Engineering. I would like to specially thank Qing Yang, Chengjun Wang, Shaoen Wu and Haiquan Chen for their help during my study life at Auburn.

I owe my loving thanks to my husband Yuyun Zhan for his sincere understanding, considerate caring, always support and unconditional love. It would have been impossible for me to complete this dissertation without his encouragement and support.

Last but not least, I am infinitely grateful to my mother, Ailian Wei and my father, Shouzhi Tian. They provided me with unconditional love, advice, and support. My parents have presented me with admirable role models of diligence, wisdom and perseverance. Despite our long distance relationship, my sisters, brothers and all the relatives in our big family overwhelmed me with more love and support throughout my years in college and graduate school than what I could have ever deserved. Especially thanks go to my sister for her unwavering support, mentoring and encouragement. She always has more reasons to appreciate life than to complain no matter whatever happens.

Table of Contents

List of Figures

List of Tables

# Chapter 1

# Introduction

With the wide use and development of distributed computing, parallel computing and cloud computing, the security issues for such applications have emerged as primary and even bottlenecks for systems dealing with large amount of sensitive data. With the increasing of the storage node number in distributed storage systems, the heterogeneous feature is becoming more common. There are a lot of solutions exist to solve the security problems based on different aspects of the systems. Although heterogeneity issues of distributed systems have been widely investigated, little attention has yet been paid to security solutions designed for distributed storage systems with heterogeneous vulnerabilities. Based on the heterogeneous trend of distributed systems, we believe that there are methods to make use of heterogeneity of such systems to improve system security while considering system performance and availability. The objective of this dissertation work is to investigate how to use heterogeneity among distributed storage nodes to improve system security and at the same time to improve system performance and availability. This chapter first presents the problem statement in Section 1.1. In Section 1.2, we describe the scope of this research. Section 1.3 highlights the main objectives of this dissertation, and Section 1.4 outlines the dissertation organization.

## 1.1 Problem Statement

In this section, we start with an overview of the security problem and techniques of distributed systems. In Section 1.1.2, we describe the heterogeneity trend of larger and scalable distributed storage systems. Section 1.1.3 specifically discusses the heterogeneous vulnerabilities of distributed systems. Then, we present our another two design goals (except security) of performance and availability in distributed storage systems in Section 1.1.4.

### 1.1.1 Security of Distributed Systems

Distributed systems which provide information sharing and large scale data storage, have enormous impact on our society. The security demand for such distributed systems is becoming increasingly important. The consequences of the failures of the distributed systems can have high cost, from financial resources loss to even human lives loss. Modern large scale distributed systems services must provide guarantees for protecting services against malicious threats [99]. However, distributed systems are more vulnerable to threats than centralized systems, since it is difficult to control processing activities of the distributed systems and information can be accessed over networks [142].

A variety of factors have impact on distributed system security including network topology, storage nodes, data placement, distributed system physical security environment, distributed system management structure, and interactions between different security mechanisms [88]. Except the popular and traditional security techniques like authentication [135] and access control [107] [86], many security approaches have been developed to improve security level of distributed systems corresponding to the above mentioned influential factors on security. Some of the security techniques are based on the system architecture level. For example, Benson, G. and Appelbe, W. et al. proposed a hierarchical model for distributed system security [18]; Naqvi, S. and Riguidel, M. developed a security architecture for heterogeneous distributed computing systems [87]; Soshi, M. and Maekawa, M. proposed a security architecture for open distributed systems [120]. Some security techniques are developed by building secure system models. For example, Noeparast, E.B. and Banirostam, T. proposed a cognitive model of immune system for increasing security in distributed systems [90]; Ching Lin and Varadharajan, V. developed a hybrid trust model for enhancing security in distributed systems [73]; Varadharajan, V. and Black, S. proposed amultilevel security model for a distributed object-oriented system [130].

Intrusion detection is a security technique to improve system security by identification of unauthorized use, or intrusion attempt in computer network environment [113] [113] [146] [3] [103]. Intrusion detection is one of the most common security measures that enterprises use to protect themselves from malware, worms and all other types of cyber attacks. Vokorokos, L. and Chovanec, M. et al. proposed a technique of security distributed intrusion detection system based on multisensor fusion [132]. Their approach is based on behavior prediction of attacker and legitimate user of computer network and they suggests use of customized DIDS (distributed intrusion detection systems), which combines distributed monitoring using external sensors and centralized data analysis for more accurately identification of security events. Many intrusion detection methods are based on different agents [28] [57] [26] [152] [136] [111] [75] [58]. A variety of other intrusion detection techniques have been proposed [83] [83] [49] [110] [104] [123].

Intrusion detection is not enough in many cases. It must have some idea of the attack signature before it can defend against it in order for an intrusion detection system (IDS) to be effective. Networks remain especially vulnerable to new forms of attacks, since it is impossible to have a signature for an attack that hasn't been seen yet. In addition, there are 120K malware incidents identified per day by these tools, with 5 - 20 new malware strains missed every day. With all these new threats, not to mention the highly targeted Advanced Persistent Threats, IDS is simply incapable of protecting the distributed system enough.

Fault tolerance is another important and common technique to improve distributed system security and availability. It is the property that enables a distributed system to continue operating properly in the event of the failure of (or one or more faults within) some of its components [143] [51] [126] [105]. A variety of techniques providing fault tolerance for distributed systems have been proposed [150] [147] [40] [77] [14] [55] [65]. Replication is one of the methods to provide fault tolerance [31] [114] [119] [50].

Although these techniques are able to provide a certain level of security for distributed systems, the conventional security techniques lack the ability to express heterogeneity in security services [142]. With the increasing of size and scalability modern large distributed systems, heterogeneity is a more and more obvious trend.

### 1.1.2 Heterogeneity Trends

A distributed computing system is a collection of independent computers linked by a computer network that appear to the users of the system as a single coherent system [2]. Cluster computing is a type of distributed computing. A computer cluster consists of a set of loosely connected or tightly connected computers that work together so that in many respects they can be viewed as a single system [1]. Currently distributed cluster computing is becoming more popular in a lot of application fields both in academic and industry such as bioinformatics, weather forecasting, social networks and other web-based applications. One of the advantages for distributed computing systems is that they can easily grow incrementally by adding more machines to the systems as requirements on processing power grow. The large data and high demand drive the sizes of distributed systems to increase very quickly and to become more scalable and heterogeneous.

When the sizes of distributed cluster computing systems grow, the heterogeneous features [4] also grow such as available bandwidth, processor speed, disk capacity, security, failure rate, and pattern of failures among the storage nodes and network condition. On the other hand heterogeneous features of the different applications that run on such systems are increasing at the same time. The data for different applications may have different size, access rate, and different quality of security and performance requirements.

### 1.1.3 Heterogeneous Vulnerabilities

Heterogeneous distributed systems have been applied to security sensitive applications, such as banking systems and digital government, which require new approaches to security [142]. Heterogeneity issues of distributed systems have been widely investigated [19] [84] [15] [85] [41]. There are a lot of factors that affect distributed system security both in hardware and software [89]. The traditional security techniques for distributed systems include access control, security threat detection, authentication, authorization, and fault tolerance et al..There is an increasing demand to develop large-scale distributed storage systems supporting data-intensive services that provide resource sharing and fault tolerance. The confidentiality of security-sensitive files must be preserved in modern distributed storage systems, because distributed systems are exposed to an increasing number of attacks from malicious users [101].

Although there exist many security techniques and mechanisms (for example, [81] and [148]), it is quite challenging to secure data stored in distributed systems. In general, security mechanisms need to be built for each component in a distributed system, then a secure way of integrating all the components in the system must be implemented. It is critical and important to maintain the confidentiality of files stored in a distributed storage system when malicious programs and users compromise some storage nodes in the system.

The file fragmentation technique is often used in many distributed and parallel systems to improve availability and performance. Several file fragmentation schemes have been proposed to achieve high assurance and availability in a large distributed system [82][137]. In real-world distributed systems, the fragmentation technique is usually combined with replication to achieve better performance at the cost of increased security risk to data stored in the systems. A practical distributed system normally contains multiple heterogeneous servers providing services with various vulnerabilities. Unfortunately, the existing fragmentation algorithms do not take the heterogeneity issues into account.

In addition to cryptographic systems, secret sharing is an approach to providing data confidentiality by distributing a file among a group of $n$ storage nodes, to each of which a fragment of the file is allocated. The file can be reconstructed only when a sufficient number (e.g., more than $k$) of the fragments are available to legitimate users. Attackers are unable to reconstruct a file using the compromised fragments, if a group of servers are compromised and fewer than $k$ fragments are disclosed.

In a large-scale distributed system, different storage sites have a variety of ways to protect data. The same security policy may be implemented in various mechanisms. Data encryption schemes may vary; even with the same encryption scheme, key lengths may vary across the distributed system. The above mentioned factors can contribute to different vulnerabilities among storage sites. Although security mechanisms deployed in multiple storage sites can be implemented in a homogeneous way, different vulnerabilities may exist due to heterogeneities in computational units.

There are a bunch of work and research have been done on distributed system security [64] [52] [60] [67] [16] [102]. However little attention has been paid to security solutions designed for distributed storage systems by making use of the heterogeneous vulnerabilities. This problem motivates us to focus on heterogeneity issues concerning security mechanisms of distributed storage systems.

### 1.1.4 Another Two Design Goals: High Performance and Availability

The design goals of distributed systems include high security, performance, availability, and the like. Many solutions have been proposed to achieve these goals in distributed systems. In the previous section, we summarized the current security techniques for distributed systems. In this part, we introduce the past and current research on high performance and availability in distributed systems.

A handful studies has been done to improve performance for various types of distributed systems to support different applications [125] [43] [23] [133] [76] [23] [140]. The heterogeneous features of distributed systems have been well investigated to improve performance. For example, Jiong et al. proposed a solution to improve MapReduce performance through data placement in heterogeneous Hadoop clusters. [140] Reliability and availability are another two very important requirements for high quality service of distributed systems. There are a wide range of approaches to improving system reliability [33] [21] and availability [92] [13] [7] [53]. Data replication techniques are very popular solutions in distributed systems to improve system availability. Many studies have been carried out to investigate data replication techniques and their applications [29] [78] [153] [100] [106] [20]. In data replication scheme there is a very important parameter called replication degree which indicates the number of replicated copies for each piece of data. The appropriate value of replication degree is very important from the availability's perspective. If too many replica copies are created in distributed system, the security risk is increased due to increased chances of being be attacked by hackers. On the other hand, increasing replica numbers significantly consume distributed system resources including storage space, power and maintenance cost. A few studies have focused on determining the best replication degree for distributed systems [154] [94] [96].

A lot of work has been done to improve security, performance, or availability of high-quality online services. Unfortunately, there is very little work that has been done to improve security by making use of heterogeneity of distributed systems through file assignment.

## 1.2 Research Scope

To address the above mentioned limitations, we propose a file fragmentation and allocation approach to improving assurance, scalability and performance of a heterogeneous distributed system.

We start to address security heterogeneity issues by dividing storage servers into different server-type groups. Each server type represents a level of security vulnerability. In a server-type group, storage servers with the same vulnerability share the same weakness that allows attackers to reduce the servers' information assurance. Although it may be difficult to classify all servers in a system into a large number of groups, a practical way of identifying server types is to organize these with similar vulnerabilities into one group. If one or more fragments of a file have been compromised, it is still very hard for a malicious user to reconstruct the file from the compromised fragments. Our solution is different from those previously explored, because our approach utilizes heterogeneous features regarding vulnerabilities among servers.

To evaluate our method for fragment allocations, we develop static and dynamic assurance models to quantify the assurance of a heterogeneous distributed storage system handling data fragments. Experimental results show that increasing heterogeneity levels can improve file assurance in a distributed storage system.

We investigate the possible parameters that influence both the system performance and the proposed S-FAS scheme. There are three aspects in a distributed storage system that influence its security and performance, workload, storage nodes and network interconnects. There are different elements of each aspect of the system. We extracted the possible key elements of each aspect by analyzing of the proposed S-FAS scheme.

Based on the both performance and security analysis, we developed a secure allocating processing (SAP) algorithm for the S-FAS scheme to both improve the security level and consider system performance by using the heterogeneous feature of large distributed systems. The SAP allocation algorithm considers load balancing, delays caused by the workload variance of many consecutive requests, and the heterogeneous nature of storage nodes in a system.

In order to use practical implementations to demonstrate the ideas on actual systems with real-world applications, we developed a prototype using the multi-threading technique and C language for the S-FAS scheme with the SAP algorithm to guide the file allocation.

The prototype is built in the distributed cluster environment with heterogeneous storage nodes, in which the Network File System (NFS) and Linux are installed. We did some experiments on system throughput and testing against real world traces. The evaluation results show that the proposed solution can not only improve the security level, but also improve the throughput and performance of the distributed storage systems with heterogeneous vulnerabilities by using the multi-thread technique.

To further explore the security solution while considering system availability, we propose Reef, which extends S-FAS by incorporating replicas of file fragments. Reef is designed to improve the distributed storage system security and availability by integrating the fragmentation technique, secret sharing, and fragment replication. Reef considers heterogeneity features of distributed storage systems during the replica placement phase. The Reef scheme is an extension of the S-FAS scheme. The system model for Reef is similar to that of S-FAS except that Reef address the system failures mode and aims to improve system reliability in addition to security. We build a static assurance model to quantitatively evaluate the system assurance for the Reef scheme. We also developed a replica allocation process algorithm called R-SAP to demonstrate how does the proposed Reef scheme work. In the Reef design, we addressed the distributed system security, performance, as well as availability. To evaluate the assurance provided by Reef, we studied the impacts caused by replication degree, system size, and the number of fragments.

In this dissertation study, we only focus on the static replica placement solution. Dynamic replica reallocation schemes are essential to achieve high performance and availability of distributed systems, especially for internet based applications and services. In a dynamic wide-area environment, client access patterns, network conditions, and service characteristics are constantly changing. We plan to study and propose a dynamic replica reallocation approach for heterogeneous distributed system by extending Reef to address the heterogeneous vulnerabilities in the large scale distributed systems.

## 1.3 Objectives

The following are five main objectives that we plan to realize with this study:

1. We aim to address the heterogeneous vulnerability issue by categorizing storage nodes of a distributed system into different server-type groups based on their vulnerabilities. Each server-type group - representing a level of vulnerability - will contain storage nodes with the same security vulnerability. We will propose a secure fragmentation allocation scheme called S-FAS to improve security of a distributed system where storage nodes have a wide variety of vulnerabilities.

2. We plan to develop the storage assurance and dynamic assurance models to quantify information assurance and to evaluate the proposed S-FAS scheme. We will find principles to improve assurance levels of heterogeneous distributed storage systems. The principles will be general guidelines to help designers achieve a secure fragment allocation solution for distributed systems.

3. We plan to develop a secure allocating processing (SAP) algorithm to improve security and system performance by considering the heterogeneous feature of a large distributed system.

4. In order to conduct the performance analysis for the S-FAS scheme and SAP allocating algorithm, we will develop a prototype for the S-FAS scheme and SAP algorithm. We will also implement the prototype and conduct some experiments on the throughput of the proposed scheme and algorithm. We will do some experiments on system throughput and testing against real world traces.

5. Last, to further explore the security solution while considering system availability, we will propose Reef, which extends S-FAS by incorporating replicas of file fragments. Reef is aimed to improve the distributed storage system security and availability by integrating the fragment replication technique, secret sharing, fragment replication.

Reef will consider heterogeneity features of distributed storage systems during the replica placement phase. The Reef scheme will be an extension of the S-FAS scheme. The system model for Reef should be similar to that of S-FAS except that Reef address the system failures mode and is aimed to improve system reliability in addition to security. We plan to build a static assurance model to quantitatively evaluate the system assurance for the Reef scheme. We will also develop a replica allocation process algorithm called R-SAP to demonstrate how does the proposed Reef scheme work. In the Reef design, we plan to address the distributed system security, performance, as well as availability. To evaluate the assurance provided by Reef, we will study the impacts caused by replication degree, system size, and the number of fragments.

The architecture of the core work of this dissertation is outlined in Fig. 1.1. Our original motivation and objective for this research are security techniques for distributed heterogeneous systems. Performance and availability are another two very important desired prosperities in distributed systems. Security, performance and availability prosperities usually conflict with each others. We discover that by making use of the heterogeneous features in distributed systems we can improve not only security, but also performance and availability, so we further address the performance and availability issues based on our proposed security solution.

## 1.4 Dissertation Organization

This dissertation is organized in the following manner:

Chapter 2 reviews related work and presents the comparison of our work with existing solutions.

Chapter 3 describes S-FAS - a secure fragmentation allocation scheme by make use of the heterogeneous feature among storage nodes in distributed systems. In the S-FAS approach, we integrate file fragmentation with the secret sharing technique in a distributed storage system with heterogeneous vulnerabilities. Storage sites in a distributed systems are

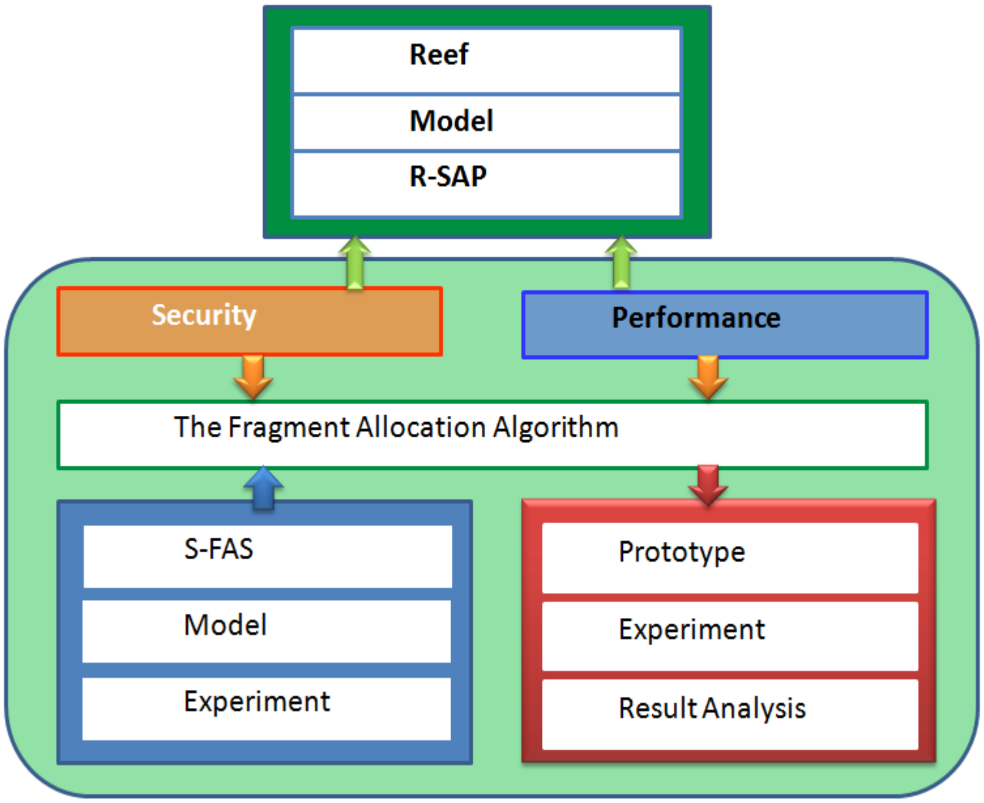

Figure 1.1: *The Architecture of The Core Work*

classified into a variety of different server types based on vulnerability characteristics. Given a file and a distributed system, S-FAS allocates fragments of the file to as many different types of nodes as possible in the system. Data confidentiality is preserved because fragments of a file are allocated to multiple storage nodes. Storage and dynamic assurance models are developed to evaluate the quality of security offered by S-FAS. Analysis results show that fragment allocations made by S-FAS lead to enhanced security because of the consideration of heterogeneous vulnerabilities in distributed storage systems.

Chapter 4 presents a Secure Allocation Processing (SAP) algorithm for the S-FAS scheme to improve the security level and consider its performance using the heterogeneous features of a large distributed system. To improve the security, the design of SAP is guided by the experimental results from S-FAS; to improve performance, we not only consider the heterogeneity of the storage nodes and the whole system, but also the heterogeneous features of the requests. The SAP allocation algorithm considers load balancing, delayed effects caused by the workload variance of many consecutive requests, and the heterogeneous features (such as CPU speed and network bandwidth) of the storage nodes in the system.

Chapter 5 presents a prototype using the multi-threading technique and C language for the S-FAS scheme with the SAP algorithm to guide the file allocation. The prototype is built in the distributed cluster environment with heterogeneous storage nodes, in which the Network File System (NFS) and Linux are installed. Some experiments on system throughput and testing against real world traces are presented. The evaluation results show that the proposed solution can not only improve the security level, but also improve the throughput and performance of the distributed storage systems with heterogeneous vulnerabilities by using the multi-thread technique.

Chapter 6 proposed a solution called Reef to improve the distributed storage system security and availability by integrating the fragmentation technique, secret sharing, and fragment replication. Reef considers heterogeneity features of distributed storage systems during the replica placement phase. The Reef scheme is an extension of the S-FAS scheme.

The system model for Reef is similar to that of S-FAS except that Reef address the system failures mode and aims to improve system reliability in addition to security. We build a static assurance model to quantitatively evaluate the system assurance for the Reef scheme. We also developed a replica allocation process algorithm called R-SAP to demonstrate how does the proposed Reef scheme work. In the Reef design, we addressed the distributed system security, performance, as well as availability. To evaluate the assurance provided by Reef, we studied the impacts caused by replication degree, system size, and the number of fragments.

Chapter 7 presents the future work directions based on the ideas contained in the dissertation.

Chapter 8 summarizes the main contributions of this dissertation.

# Chapter 2

# Literature Review

In this chapter, we summarize the previous literatures that are most relevant to the research in terms of security, performance, and availability in distributed cluster computing systems. Section 2.1 introduces related work on security solutions in distributed systems; Section 2.2 presents the related work on fragmentation techniques in distributed storage systems; Section 2.3 introduces the secret sharing security solution; Section 2.4 presents the related work on load balancing for high performance distributed cluster storage systems; Section 2.5 presents the related work on replication scheme to improve performance and availability for distributed systems. Section 2.6 presents our observations from the existing solutions. Section 2.7 presents the comparison of our work with the existing solutions.

## 2.1 Security Techniques for Distributed Systems

Much research has been performed to improve security of distributed and high-performance computing systems such as Grids. For example, Pourzandi *et al.* proposed a structured security approach that incorporates both distributed authentication and distributed access control mechanisms [101].

Intrusion detection techniques have been widely used to provide basic assurance of security in distributed systems. However, most intrusion detection techniques are inadequate to protect data stored in distributed systems [63]. One of the most effective approaches to improving information assurance in distributed systems is intrusion tolerance [35] [122] [137]. To enhance security assurance, researchers have developed a range of intrusion-tolerant tools and mechanisms. The fragmentation technique summarized below is one of the intrusion tolerance methods that can be used in combination with intrusion detection techniques.

## 2.2 Fragmentation Techniques

A fragmentation technique partitions a security sensitive file into multiple fragments that are distributed across different storage servers in a distributed system. A lot of fragmentation schemes have been proven to be valuable tools for improving the security of data stored in distributed systems (see [66]). Many fragmentation approaches aim to improve availability and performance of distributed systems by applying data replication methods. For example, Dabek *et al.* developed a wide-area cooperative storage system in which a fragmentation scheme was implemented to improve availability and to facilitate load balancing [32].

Although combining a fragmentation and replication scheme can enhance performance and availability, data replications may increase security risks due to an increasing number of file fragments handled by distributed storage servers. A file is more likely to be compromised when more replications of the file are stored in distributed storage servers.

Existing file fragmentation technologies are inadequate in addressing the issue of heterogeneous vulnerabilities in large-scale distributed systems. Our preliminary results show that security can be improved in a distributed storage system when a fragmentation scheme incorporates the heterogeneous-vulnerability feature with our S-FAS scheme.

## 2.3 Secret Sharing

Secret sharing–independently invented by Shamir and Blakley–is a method of distributing a secret among a group of participants, each of which is allocated a share of the secret. The secret can be successfully reconstructed only when a sufficient number of shares are collected and combined [98][112].

Shamir proposed the $(k, n)$ secret sharing scheme that divides data $D$ into $n$ pieces in such a way that $D$ can be easily reconstructed from any $k$ pieces. If fewer than $k$ pieces are disclosed, no one can reconstruct $D$ from the revealed pieces.

The secret sharing scheme has been extended and employed in different application domains [116]. For example, Bigrigg *et al.* proposed an architecture called PASIS for secure storage systems. The PASIS architecture integrates the secret sharing scheme with information dispersal to improve security, integrity and availability [138][149]. In a storage system with PASIS, even if an attacker compromises a limited (i.e., fewer than the threshold) subsets of storage nodes, the confidentiality of data stored in the system is still preserved. The aforementioned secret-sharing solutions designed for distributed storage systems ignore the issue of heterogeneous vulnerabilities. This fact motivates us to extend the secret sharing scheme by considering heterogeneity of vulnerabilities in the context of distributed storage systems.

## 2.4 Load Balancing to Improve System Performance

We can classify existing load balancing approaches into different categories such as static and dynamic or homogenous and heterogeneous. The focus of homogeneous load balancing schemes is to improve the performance of homogeneous parallel and distributed systems. On the other hand, heterogeneous load balancing approaches attempt to boost the performance of heterogeneous clusters, which comprise a variety of nodes with different performance characteristics in computing power, memory capacity, and disk speed. A static load balancing scheme for Computational Fluid Dynamics simulations on a network of heterogeneous workstations has been studied by Chronopoulos et al [27]. Their load-balancing algorithm takes both the CPU speed and memory capacity of workstations into account. To dynamically balance computational loads in a heterogeneous cluster environment, Cap and Strumpen explored heterogeneous task partitioning and load balancing. Xiao et al [139]. have investigated an approach that considers both system heterogeneity and effective usage of memory resources, thereby minimizing both CPU idle time and the number of page faults in heterogeneous systems.

## 2.5 Replication Scheme for High Performance and Availability

Data replication is a fundamental technique to increase data availability, reliability and robustness in distributed systems. A number of factors including replication degree, replica placement and maintenance strategies have impact on the effect of data replication. There is a lot of work have been done studying the impact from replication degree [72], replica placement [45] [9] [115] [131], and replication maintenance [155]strategies and overhead [145] [46]. However, data replication brings the issue of data consistency and many strategies have been proposed to solve the data consistency issue caused by data replication [6] [8] [5] [144] [12].

High performance is always highly demand in distributed systems. Many factors influence performance of a distributed system including quality of hardware, software, network connectivity, data management and scheduling strategies, and etc.. High performance has been investigated from various aspects [25] [24] [109] [118]. Data replication is one of the approaches that not only improve availability, but also improve performance of a distributed system. Many techniques have been developed to boost the performance of distributed systems where data replication is applied [91] [34] [38] [151].

Availability is usually measured as a factor of the reliability of a distributed systems. Availability is one of the desired properties [62] and different approaches have been proposed to improve availability in distributed systems [54] [36] [42] [10] [128] [95]. Except data replication technique, sophisticated management, load balancing and recovery techniques are needed to achieve high availability amidst an abundance of failure sources that include software, hardware, network connectivity, and power issues [44]. Data placement and replication strategies are among the top list of multiple design choices while data replication is chosen to improve availability of a distributed system [44]. A variety of research has been done to study efficient data placement [97] and replication strategies [127] [61].

Data replication can improve availability, but it brings the issue of data consistency. A lot of work have been done to study the tradeoff between availability and data consistency. However, there is very little work has been done to study the tradeoff between availability and security while replication scheme is used. The more replicas of a file or fragment are stored, the bigger the risk is that the file or fragment is compromised by hackers. In this dissertation, we will address the tradeoff issue between availability and security due to data replication in heterogeneous distributed systems.

## 2.6 Observations

We observe that vulnerabilities of storage nodes in a distributed system are more heterogeneous due to the following four main reasons. First, storage nodes have different ways to protect data. Second, a security policy can be implemented in a variety of mechanisms. Third, the key length of an encryption scheme may vary across multiple storage nodes. Fourth, heterogeneities exist in computational units of storage sites. There is very little work has been done to boost security by making use of heterogeneous features in distributed systems. We believe that future security mechanisms for distributed systems must be aware of vulnerability heterogeneities.

With the previously mentioned limitations of existing techniques designed for distributed systems, we propose our investigation results and solution for distributed systems with heterogeneous systems in this dissertation.

## 2.7 Comparison of Our Work with Existing Solutions

Our proposed security schemes, assurance evaluation models, secure file allocation algorithms and prototype in this dissertation are different from existing solutions, because our approach aim to incorporate the heterogeneous vulnerabilities of distributed systems into file fragment allocation and secret sharing. Our solution captures heterogeneous features of the nodes regarding vulnerabilities to improve security. At the same time, our techniques also consider to boost performance and availability of distributed systems.

# Chapter 3

# Secure Fragment Allocation Scheme S-FAS

In previous two chapters we present the introduction of this dissertation and the related work. In this Chapter we introduce our first work of a fragment allocation scheme called S-FAS in to improve security of a distributed system where storage sites have a wide variety of vulnerabilities.

Distributed storage systems are becoming ubiquitous because of the large amount of data required for search engines, multimedia websites, and data-intensive high-performance computing [39] [48]. These distributed storage systems typically are at high risk and inefficient concerning the high confidential and performance requirements of the storage data. Security is one of the key qualities that most customers care about. Without a certain level of security a storage system is useless for a lot of applications of high confidential requirements. With more and more personal, commercial, governmental and scientific data needing to be stored in distributed systems, it is important to enhance the security level of distributed storage systems.

A handful of traditional or novel techniques developed to improve security in storage systems include authentication, authorization, fragmentation techniques, secret sharing, erasure coding. These security techniques can significantly enhance the assurance level of the distributed system.

With the increasing of the storage node number in distributed storage systems, the heterogeneity feature is becoming more common.Vulnerabilities of storage nodes in a distributed system are heterogeneous in nature due to the following four main reasons. First, storage nodes have different ways to protect data. Second, a security policy can be implemented in a variety of mechanisms. Third, the key length of an encryption scheme may vary across multiple storage nodes. Fourth, heterogeneities exist in computational units of storage sites. We believe that future security mechanisms for distributed systems must be aware of vulnerability heterogeneities. Although heterogeneity issues of distributed systems have been widely investigated, little attention has yet been paid to security solutions designed for distributed storage systems with heterogeneous vulnerabilities. Since the existing security techniques developed for distributed systems are inadequate for distributed systems with heterogeneity in vulnerabilities, the focus of this study is heterogeneous vulnerabilities in largescale distributed storage systems.

The goal of our research is to develop a Secure Fragment Allocation Scheme (S-FAS) to fully make use of the feature of heterogeneous vulnerabilities among large scale distributed storage systems where storage sites have a widevariety of vulnerabilities. In the S-FAS approach, we integrate file fragmentation with the secret sharing technique in a distributed storage system with heterogeneous vulnerabilities. Storage sites in a distributed systems are classified into a variety of different server types based on vulnerability characteristics. Given a file and a distributed system, S-FAS allocates fragments of the file to as many different types of nodes as possible in the system. Data confidentiality is preserved because fragments of a file are allocated to multiple storage nodes.

We develop storage assurance and dynamic assurance models to evaluate the quality of security offered by S-FAS. Analysis results show that fragment allocations made by S-FAS lead to enhanced security because of the consideration of heterogeneous vulnerabilities in distributed storage systems.

The rest of the chapter is organized as follows: We discuss in Section 3.1 the system and threat model. Section 3.2 describes the design of the S-FAS Scheme. In Section 3.3 we presents the static and dynamic assurance models for S-FAS.Then, Section 3.4 shows assurance evaluation results of S-FAS. Finally, Section 6.7concludes the chapter and presents our future research directions.

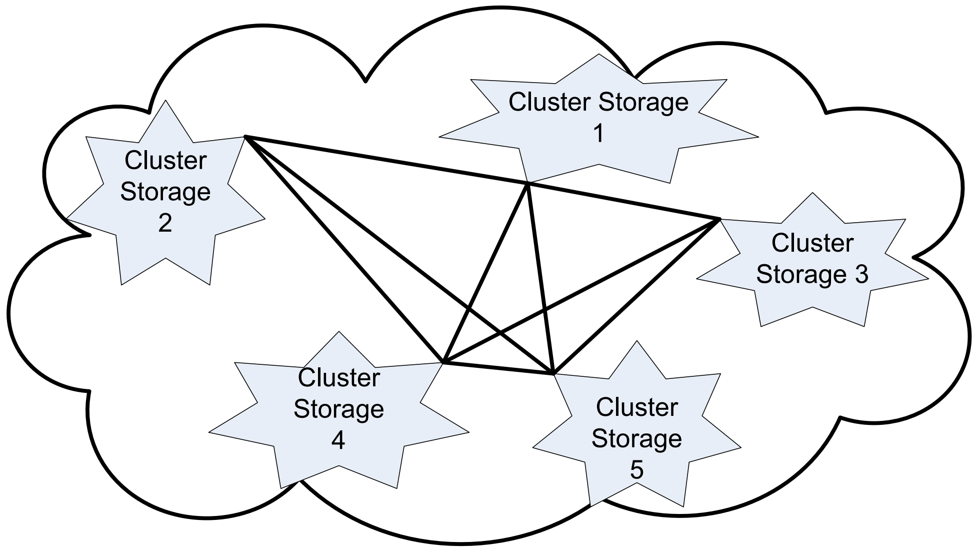

Figure 3.1: A distributed storage system is comprised of a set of cluster storage subsystems. Multiple fragments of a file can be stored either in storage nodes within a single cluster storage subsystem or in nodes across multiple cluster storage subsystems. See Fig. 3.2 for details on a cluster storage subsystem.

## 3.1 System and Threat Model

We firstly outline the system and threat models that capture main characteristics of distributed storage systems. The system model is used as a basis to design the S-FAS fragmentation allocation scheme, whereas the threat model helps us identify vulnerabilities and certain potential attacks in distributed storage systems.

### 3.1.1 System Model

The S-FAS fragmentation allocation scheme was designed for a distributed storage system (see Fig. 3.1) where each storage site is a cluster storage subsystem. Different cluster storage subsystems may be connected within some subnetworks to form a larger scale distributed storage sysytem.

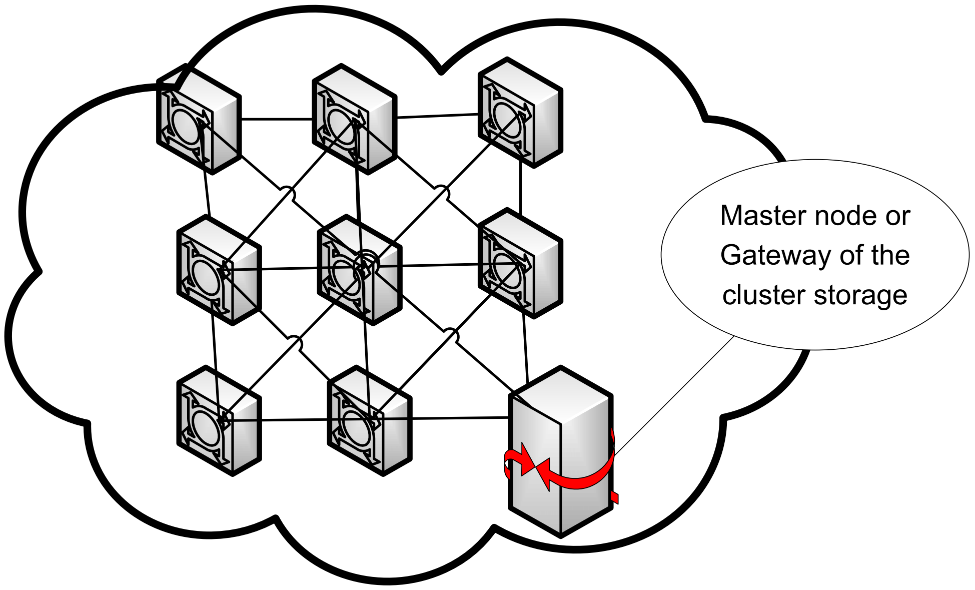

Figure 3.2: A cluster storage subsystem consists of a number of storage nodes and a gateway. Storage nodes are divided into different server-type groups, each of which represents a level of security vulnerability.

Fig. 3.2 depicts a cluster storage subsystem, which consists of a number of storage nodes and a gateway. Considering heterogeneous vulnerability in large-scale storage systems, we divide storage nodes into different server-type groups, each of which represents a level of security vulnerability .

Before presenting details on the system model, let us summarize all notations used throughout this chapter in Table 3.1.

In this study, we consider a distributed storage system containing $L$ cluster storage subsystems, i.e., $R_1, R_2, ..., R_L$. Cluster storage subsystems $R_i$ consists of $H_i$ storage nodes, i.e., $R_i = \{r_{i1}, r_{i2}, ..., r_{iH_i}\}$. All the storage nodes connected in cluster $R_i$ have heterogeneous vulnerabilities.

Since all the nodes, including a master node, are fully connected in a cluster storage subsystem, we model the topology of a cluster storage system as a general graph. Cluster

Table 3.1: Notation used in the system and models.

| Notations | Meaning |
|---|---|
| $N$ | Number of server nodes in the system |
| $U$ | The whole system considered |
| $L$ | Number of subsystems in the whole system |
| $H_i$ | Number of server nodes in the subsystem $i$ |
| $F$ | A file stored in the system |
| $F_i$ | Fragment $i$ of file $F$ |
| $T_j$ | Server type $j$ in the system |
| $K$ | The total number of server types |
| $S_j$ | The size of a certain server type in a cluster |
| $m$ | Threshold for the secret sharing scheme |
| $n$ | The total number of fragments for each file in the secret sharing scheme |
| $R_i$ | Cluster storage subsystem |
| $r_{ij}$ | Node $j$ in subsystem $i$ |
| $X$ | The event that a set of storage nodes is chosen to be attacked |
| $Y$ | The event that if $X$ occurs, at least $m$ fragments can be compromised using the same attack method. |
| $Z$ | The event of a successful attack to a certain fragment of a file |
| $V$ | The event file $F$ is compromised under one attack method |
| $P_N$ | The successful probability of an attack on a node |
| $P_f$ | The successful probability to compromise a fragment in a compromised node |
| $P(X)$ | The probability of event $X$ occurring |
| $P(Y)$ | The probability of event $Y$ occurring |
| $P(Z)$ | The probability of event $Z$ occurring |
| $P(V)$ | The probability of event $V$ occurring |
| $\alpha$ | An allocation mapping of file $F$ |
| $SA(\alpha)$ | The storage assurance of an allocation mapping $\alpha$ of file $F$ |
| $\mathrm{DA}(\alpha)$ | The dynamic assurance of an allocation mapping $\alpha$ of file $F$ |
| $q$ | Number of fragments needed to reconstruct a file transmitted from outside of the subsystem |
| $g$ | Number of fragments compromised out of the $q$ fragments transmitted from outside of the subsystem |
| $P_L$ | The probability that a fragment is intercepted during its transmission |
| $P_D$ | The probability that a file $F$ is intercepted because of the compromised transmitted fragments |

storage subsystem $R_i$ has a gateway, which hides the cluster's internal architecture from users by forwarding file requests to storage nodes.

Data in cluster storage subsystem $R_i$ can be accessed through its master node. When a read request is submitted to cluster $R_i$, the master node is responsible for reconstructing file fragments and returning the file to users. When a write request of a file is issued, the master node updates all the fragments of the file.

Legitimate users access cluster storage subsystems through master nodes; malicious users may bypass the master nodes to access storage nodes without being authorized. See Section 3.1.2 below for details on the threat model.

### 3.1.2 Threat Model

It is not reasonable to assume that if a malicious user breaks into a storage node, fragments of a file stored on the node are thereby compromised. Normally, a malicious user needs two steps to compromise fragments of a file stored on a server. First, the malicious user must successfully attack the server. Second, fragments are retrieved by the malicious user.

Let $P_N$ be the probability that a storage server is successfully attacked; let $P_f$ be the probability that authorized users retrieve fragments stored on the server, provided that the server has been compromised. We define event $Z$ as a successful attack on a fragment (i.e., unauthorized disclosure of the fragment). Since the above two consecutive attack steps are independently, the probability that event $Z$ occurs is a product of probability $P_N$ and probability $P_f$. Thus, the probability that a fragment is disclosed to an unauthorized attacker can be expressed as:

$$\mathrm{P}\left(\mathrm{Z}\right) = P_N * P_f. \tag{3.1}$$

In a dynamic allocation environment, a malicious user can use a compromised node to collect other needed fragments of the file when the fragments are passing through the compromised node.

If encryption keys are disclosed to attackers, unauthorized interceptions of encrypted files stored on the attacked node may occur. Given two storage nodes with different vulnerabilities, successful attacks of the nodes are not correlated. This statement is true for many potential threats, because compromising one storage node does not necessarily lead to the successful attack of the second one.

## 3.2 S-FAS: A Secure Fragment Allocation Scheme

In this section, we first outline the motivation for addressing the heterogeneity issues in the vulnerability of distributed storage systems. Next, we describe a security problem addressed in this study. Last, we present a secure fragment allocation scheme called S-FAS for distributed storage systems.

### 3.2.1 Heterogeneity in the Vulnerability of Data Storage

Since the existing security techniques developed for distributed systems are inadequate for distributed systems with heterogeneity in vulnerabilities, the focus of this study is heterogeneous vulnerabilities in large-scale distributed storage systems. Vulnerabilities of storage nodes in a distributed system are heterogeneous in nature due to the following four main reasons. First, storage nodes have different ways to protect data. Second, a security policy can be implemented in a variety of mechanisms. Third, the key length of an encryption scheme may vary across multiple storage nodes. Fourth, heterogeneities exist in computational units of storage sites. We believe that future security mechanisms for distributed systems must be aware of vulnerability heterogeneities.

### 3.2.2 A Motivational Example

If the above heterogeneous vulnerability features are not incorporated into fragment allocation schemes for distributed storage systems, a seemingly secure fragment allocation

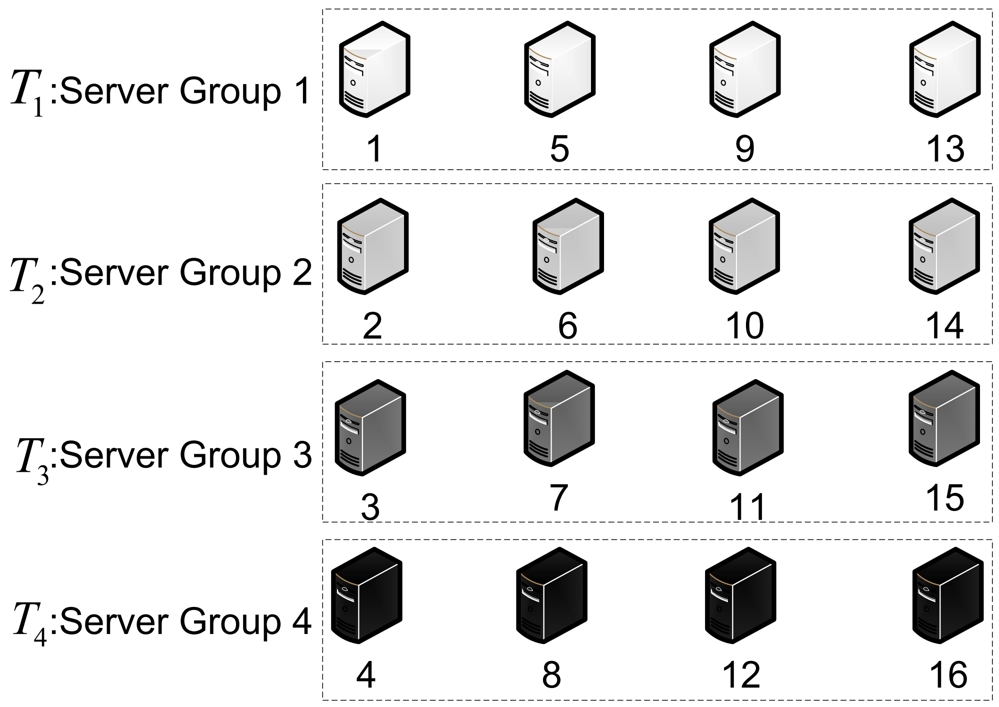

Figure 3.3: A distributed storage system contains 16 storage nodes, which are divided into 4 server-type groups (or server groups for short), i.e., $T_1$, $T_2$, $T_3$, and $T_4$. Servers in each group have the same level of security vulnerability.

decision can lead to a breach of data confidentiality. The following motivational example illustrates a security problem caused by ignoring vulnerability heterogeneities.

Let us consider a file $F$ with three partitioned fragments: $f_a$, $f_b$, and $f_c$, and a distributed storage system (see Fig. 3.3) that contains 16 storage nodes divided into 4 server-type groups (or server groups for short), i.e., $T_1$, $T_2$, $T_3$, and $T_4$. Storage nodes in each server group offer similar services with the same level of vulnerability. In this example, server group $T_1$ consists of nodes $r_1, r_5, r_9, r_{13}$, i.e., $T_1 = \{r_1, r_5, r_9, r_{13}\}$. Similarly, we define the other three server groups as: $T_2 = \{r_2, r_6, r_{10}, r_{14}\}$, $T_3 = \{r_3, r_7, r_{11}, r_{15}\}$, and $T_4 = \{r_4, r_8, r_{12}, r_{16}\}$.

Fig. 3.4 shows that it is possible to make insecure fragment allocation decisions that do not take vulnerability heterogeneity into account. The decision made using a hashing function (see Eq. 11 in [82]) randomly allocates the three fragments of file $F$ to three different nodes, each of which belongs to one of the three server sets illustrated in Fig. 3.4. For example, the three fragments $f_a$, $f_b$, and $f_c$ are stored on nodes $r_1$, $r_6$, and $r_8$, respectively.

This fragment allocation happens to be a good solution, because $r_1$, $r_6$, and $r_8$ have different vulnerabilities as the three nodes belong to different server groups (i.e., $T_1$, $T_2$, and $T_4$). A malicious user must launch three successful attacks (one for each server group) in order to compromise all three fragments.

The above fragment allocation scheme fails to address the threat described in previous described Threat Model. This is because an attacker can first retrieve one fragment of $F$ by compromising a single node, then the attacker simply waits for the other two fragments to be passed through the compromised node. To solve this security problem, Zanin *et al.* developed a static algorithm to decide whether a particular storage node is authorized to handle a file fragment of $F$ [149]. Zanin's algorithm can generate an insecure fragment allocation because heterogeneous vulnerabilities are not considered. For example, the three fragments are respectively stored on nodes $r_4$, $r_8$, and $r_{12}$, which share the same vulnerability in server group $T_4$ (see Fig. 3.4). Rather than three attacks, one successful attack against server group $T_4$ allows unauthorized users to access the three fragments of file $F$. Two other insecure fragment allocations are: (1) allocating $f_a$, $f_b$, $f_c$ to nodes $r_1$, $r_5$, and $r_9$, respectively; and (2) allocating $f_a$, $f_b$, $f_c$ to nodes $r_7$, $r_{11}$ and $r_{15}$, respectively. These three fragment allocation decisions are unacceptable, because the fragments are assigned to a group of storage nodes with the same vulnerability, meaning that an attacker who comprised one node within a group can easily compromise the other nodes in the group. The attacker can reconstruct $F$ from $f_a$, $f_b$, and $f_c$ stored on the comprised server group.

### 3.2.3 Design of the S-FAS Scheme

To solve the above security problem, we have to incorporate vulnerability heterogeneities into fragment allocation schemes. Specifically, we design a simple yet efficient approach to allocating fragments of a file to storage nodes with various vulnerabilities. Since allocating fragments of a file into different storage clusters can degrade performance, our S-FAS scheme attempts to allocate fragments to storage nodes within a cluster. If the number of nodes

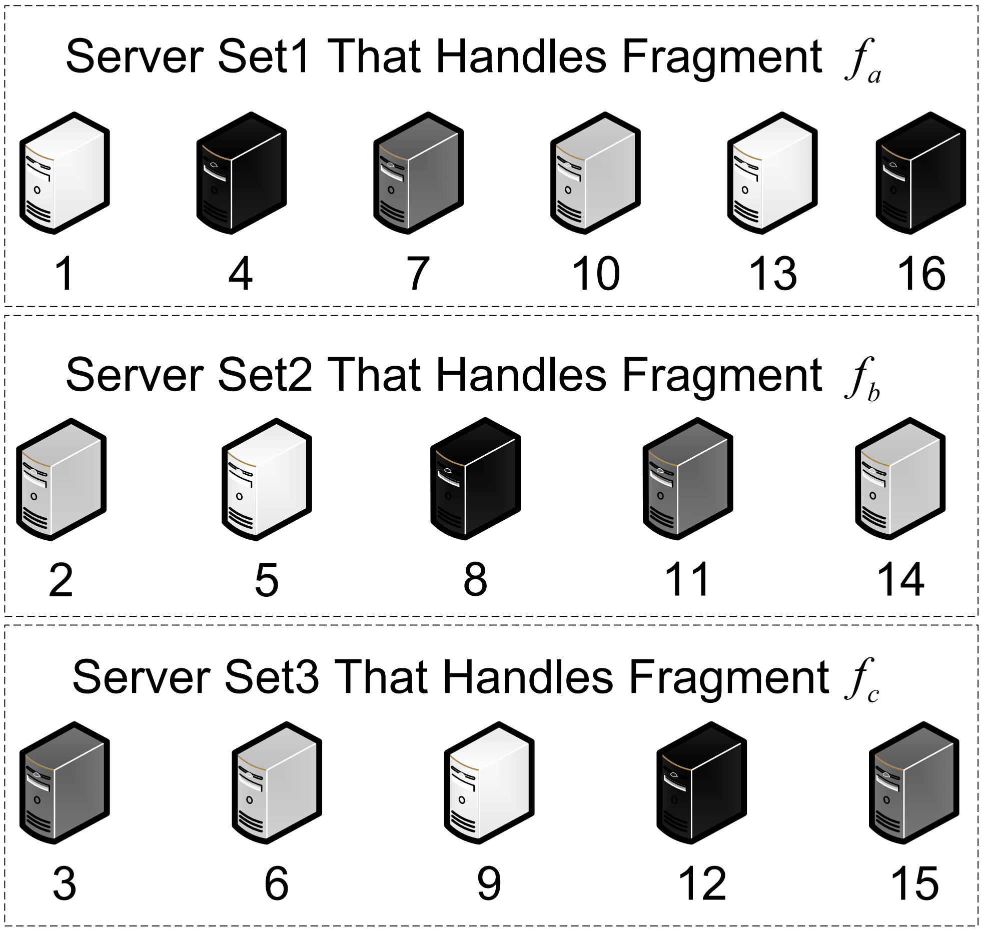
Server Set1 That Handles Fragment $f_a$
1
4
7
10
13
16
Server Set2 That Handles Fragment $f_b$
2
5
8
11
14
Server Set3 That Handles Fragment $f_c$
3
6
9
12
15

Figure 3.4: Possible insecure file fragment allocation decision made using a hashing function (see Eq. 11 in [82]): Server set 1 handles fragment $f_a$, server set 2 handles fragment $f_b$, and server set 3 handles fragment $f_c$. Server set 1 contains storage nodes $r_1$, $r_4$, $r_7$, $r_10$, $r_13$, and $r_16$; server set 2 contains storage nodes $r_2$, $r_5$, $r_8$, $r_{11}$, and $r_{14}$; and server set 3 contains storage nodes $r_3$, $r_6$, $r_9$, $r_{12}$, and $r_{15}$. It is possible that fragments $f_a$, $f_b$, and $f_c$ may be allocated to storage nodes that belong to the same server-type group. For example, the three fragments are respectively stored on nodes $r_4$, $r_8$, and $r_{12}$, which share the same vulnerability in server group $T_4$. Rather than three attacks, one successful attack against server group $T_4$ allows unauthorized users to access the three fragments of file $F$.

with different vulnerabilities cannot meet the aforementioned criterion, file fragments must be allocated across multiple clusters. To improve the assurance of a distributed storage system while maintaining high I/O performance, each cluster storage subsystem has to be built with high vulnerability heterogeneity. This causes the fragments of a file to be less likely distributed across multiple storage clusters.

Because of the following two reasons, the S-FAS scheme can significantly improve data security when fragments are stored in a large-scale distributed storage system. First, S-FAS integrates the fragmentation technique with secret sharing. Second, S-FAS addresses the issue of heterogeneous vulnerabilities when file fragments are allocated to a distributed storage system.

The S-FAS scheme makes fragment allocation decisions by following the four policies below:

- **Policy 1:** All the storage nodes in a distributed storage system are classified into multiple server-type groups (server group for short) based upon their various vulnerabilities. Each server group consists of storage nodes with the same vulnerability level.

- **Policy 2:** To improve security of a distributed storage system, S-FAS allocates fragments of a file to storage nodes belonging to as many different server groups as possible. In doing so, it is impossible to compromise the file's fragments using a single successful attack method.

- **Policy 3:** The fragments of a file are trying to be allocated to nodes with a wide range of vulnerability levels all within a single cluster storage subsystem. The goal of this policy is to improve performance of the storage system by making the fragments less likely to be distributed across multiple clusters.

- **Policy 4:** The $(m, n)$ secret sharing scheme is integrated with the S-FAS allocation mechanism.

If a file's fragment-allocation decisions are guided by the above four policies, successful attacks against less than $m$ server groups have little chance to gain unauthorized accesses of files stored in a distributed system. In other words, if the number of compromised fragments of a file is less than $m$, attackers are unable to reconstruct the file from the fragments that are accessed by the unauthorized attackers. The S-FAS scheme can improve information assurance of files stored in a distributed storage system without enhancing confidentiality services deployed in cluster storage subsystems of the distributed system, because S-FAS is orthogonal to security mechanisms that provide confidentiality for each server group in a distributed storage system. Thus, S-FAS can be seamlessly integrated with any confidentiality service employed in distributed storage systems in order to offer enhanced security services.

## 3.3 Static and Dynamic Assurance Models

We developed assurance models to quantitatively evaluate the security of a heterogeneous distributed storage system in which S-FAS handles fragment allocations.

### 3.3.1 Static Storage Assurance Model

For encrypted files, their encryption keys are partitioned and allocated using the same strategy that handle file fragments. Once a storage node in set $U$ is compromised, file fragments and encryption key fragments stored on the node are both breached. If a malicious user wants to crack a file, at least $m$ nodes within $U$ must be successfully attacked.

We first investigate the probability that a file is compromised using one attack method. Let $X$ be the event that a set of storage nodes is chosen to be attacked. Let $Y$ be the event that if $X$ occurs, at least $m$ fragments can be compromised using the same attack method. As we already defined, event $Z$ represents a successful attack to a certain fragment of a file. Applying the multiplication principle, we calculate the probability that $V$ - an event that file $F$ is compromised under one attack - occurs as:

$$\mathrm{P(V)} = \sum_{\mathrm{j=1}}^{\mathrm{k}} \mathrm{P(X)P(Y)P(Z)} \tag{3.2}$$

where $P(X)$, $P(Y)$ and $P(Z)$ are probabilities that events $X$, $Y$ and $Z$ occur when the total number of different server-type groups (server group for short) is $K$. The probability $P(V)$ is proportional to probability $P(Z)$, which largely depends on the quality of security mechanisms deployed in the storage system, as well as the attacking skills of hackers.

Note that when $k$ equals 1, there is no vulnerability difference among storage nodes. Supposing that all the fragments of a file can be compromised using one successful attack method, the probability that $Y$ occurs becomes 1. Then, we can express $P(V)$ as:

$$\mathrm{P(V)} = \sum_{\mathrm{j=1}}^{\mathrm{k}} \mathrm{P(X)P(Z)} \tag{3.3}$$

Let $S_j$ be the number of storage nodes in *server type* $T_j$ set and $N$ be the total number of nodes in a distributed system. The probability that nodes in set $T_j$ are randomly attacked can be derived as $\mathrm{P(X)} = \frac{\mathrm{S}_j}{\mathrm{N}}$.

Probability $P(Y)$ in Eq. 3.2 can be calculated as follows:

$$\mathrm{P\,(Y)} = \sum_{\mathrm{i=m}}^{\mathrm{n}} \frac{C_{S_j}^{i} C_{N-S_j}^{n-i}}{C_N^n}, (j = 2, \ldots K) \tag{3.4}$$

where $C_N^n$ is the total number of possibilities of allocating fragments of a file, and the product of $C_{S_j}^{i}$ and $C_{N-S_j}^{n-i}$ is the number of possibilities that a file is compromised using a successful attack method which means at least $m$ (It may be $m+1$, $m+2$, ..., $n$) fragments of the file are compromised.

To simplify the model, one may assume that security mechanisms and attacking skills have no significant impacts on information assurance of the entire distributed storage system. This assumption is reasonable because of two factors. First, S-FAS is independent of security mechanisms that provide confidentiality for server groups in a distributed storage system. Second, if empirical studies can provide values for probability $P(Z)$, the probability $P(V)$ can be derived from $P(Z)$ and the model (see Eq. 3.4) that calculates $P(Y)$. Since the study of the distribution of $P(Z)$ is not within the range of this work, in this paper the impact of probability $P(Z)$ on $P(V)$ is ignored by setting the value of $P(Z)$ to 1.

Now we can derive Eq. 3.2 from Eq. 3.4 as below:

$$\mathrm{P(V)} = \sum_{\mathrm{j=1}}^{\mathrm{K}} \left( \frac{\mathrm{S_j}}{\mathrm{N}} \mathrm{P(Z)} \sum_{\mathrm{i=m}}^{\mathrm{n}} \frac{\mathrm{C_{S_j}^{i} C_{N-S_j}^{n-i}}}{\mathrm{C_N^n}} \right) \tag{3.5}$$

The confidentiality of file $F$ is assured if $F$ is not compromised. Thus, we can derive the assurance $SA(\alpha)$ of the storage system from Eq. 3.5 as:

$$\begin{aligned} \mathrm{SA}(\alpha) &= 1 - \mathrm{P(V)} \\ &= 1 - \sum_{\mathrm{j=1}}^{\mathrm{K}} \left( \frac{\mathrm{S_j}}{\mathrm{N}} \mathrm{P(Z)} \sum_{\mathrm{i=m}}^{\mathrm{n}} \frac{\mathrm{C_{S_j}^{i} C_{N-S_j}^{n-i}}}{\mathrm{C_N^n}} \right) \end{aligned} \tag{3.6}$$

### 3.3.2 Dynamic Assurance Model.

During read and write operations, some fragments of a file may be transmitted among different storage clusters or subnetworks. We assume that data transmissions within a cluster are secure, while connections among clusters and subnetworks may be insecure. Let $P_L$ be the probability that a fragment is intercepted during its transmission on an insecure link. We consider a common case in which some fragments of file $F$ are allocated outside a cluster. The probability $P_D$ that a fragment of $F$ is intercepted during its transmission can be expressed as:

$$\mathrm{P_D} = \mu_1 \mu_2 \mathrm{P_L} + \mu_3 [1 - \mathrm{P_L}] \mathrm{P_L} \tag{3.7}$$

where $\mu_1 = 1$ indicates that connections among storage clusters are insecure and $\mu_1 = 0$ means the connections are secure. $\mu_2 = 1$ indicates that fragments are transferred among different clusters, otherwise $\mu_2 = 0$. Similarly, $\mu_3 = 1$ means that fragments are transmitted across different subnetworks. When $\mu_1$, $\mu_2$, and $\mu_3$ equal to 0, there is no fragment transmission risk. If $q$ fragments need to be collected outside a cluster processing read/write operations, then probability $\mathrm{P_q(g)}$ that $g$ out of $q$ fragments are intercepted can be expressed as:

$$\mathrm{P_q\left(g\right) = C_q^g P_D{}^g (1 - P_D)^{q-g}} \tag{3.8}$$

Now we model the dynamic assurance of an allocation mapping $\alpha$ of file $F$. For simplicity, let us focus on a time period during which there is only one attempt to attack storage nodes where $F$ is stored. During this time period, we assume that only one read or write operation is issued to access $F$. There are two cases where file $F$ can be compromised. First, a malicious user can reconstruct $F$ from $m$ compromised fragments using the same attack method. Second, although less than $m$ fragments are compromised, other $g$ fragments are intercepted during their transmissions. Hence, we can derive the dynamic assurance $\mathrm{DA}(\alpha)$ from the storage risk (see Eq. 3.5) and the transmission risk (see Eq. 3.8), as shown here:

$$\begin{aligned} &DA(\alpha) = 1- \\ &\left(P\left(V\right) + \left(\sum_{g=(m-i)}^{q} P_q(g)\right) \sum_{j=1}^{K} \left(\frac{S_j}{N} \times \sum_{i=0}^{m-1} \frac{C_{S_j}^{i} C_{N-S_j}^{n-i}}{C_N^n}\right)\right) \end{aligned} \tag{3.9}$$

## 3.4 Evaluation of System Assurance

The assurance models described in Section 3.3 indicate that system assurance is affected by the number $K$ of storage types, the number $N$ of storage nodes in the system, and the number $S_j$ of nodes in the $j$th storage type. In addition, threshold $m$ and the number $n$ of fragments in a file also have an impact on system assurance. Now, we quantitatively evaluate the impacts of these factors on the information assurance of distributed storage systems. We first obtain a comprehensive evaluation of S-FAS in terms of data storage assurance and

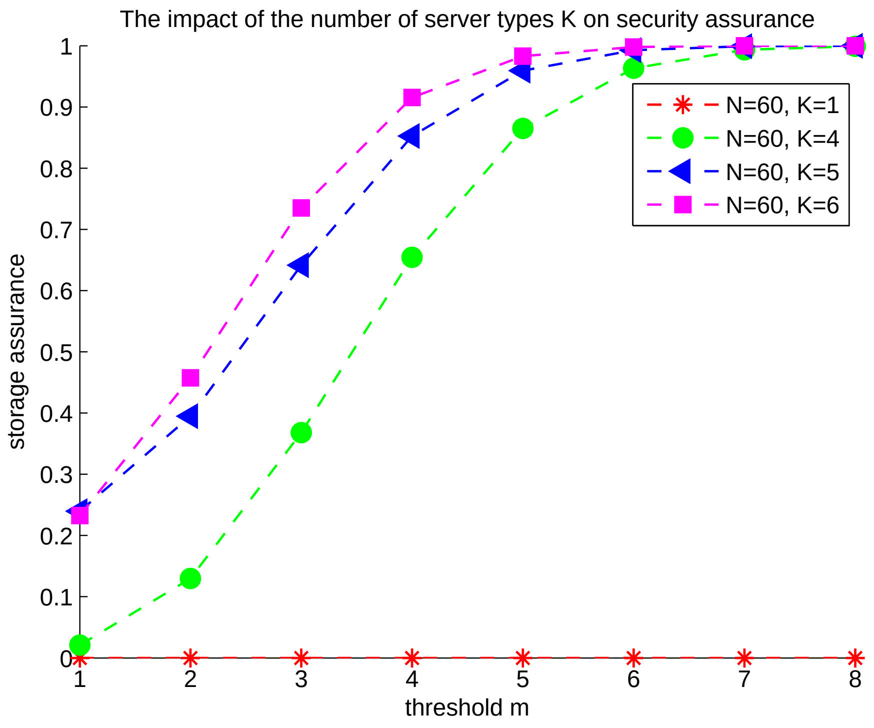

Figure 3.5: *Heterogeneous system and homogeneous system using secret sharing scheme.* In all the four test cases, $N$ is set to 60. $K$ is set to 1, 4, 5, and 6, respectively. When $K$ is 1, there is only one server group in the system.

dynamic assurance of S-FAS. We compare our approach with a traditional fragment allocation scheme that does not consider vulnerability heterogeneities. We evaluated a distributed storage system with the threshold value $m$. The default number $n$ of fragments of a file is set to 12 and $S_j = \frac{N}{K}$ for all $j$ from 1 to $K$.

### 3.4.1 Impact of Heterogeneity on Storage Assurance

If all storage nodes in the evaluated distributed system are identical in terms of vulnerability, the probability that fragments of a file can be compromised using one successful attack method is 1. Fig. 3.5 shows the impact of the number $K$ of storage types on system assurance. Results plotted in Fig. 3.5 suggest that for a distributed system with homogeneous vulnerability, threshold $m$ has no impact on system assurance. When it comes to a

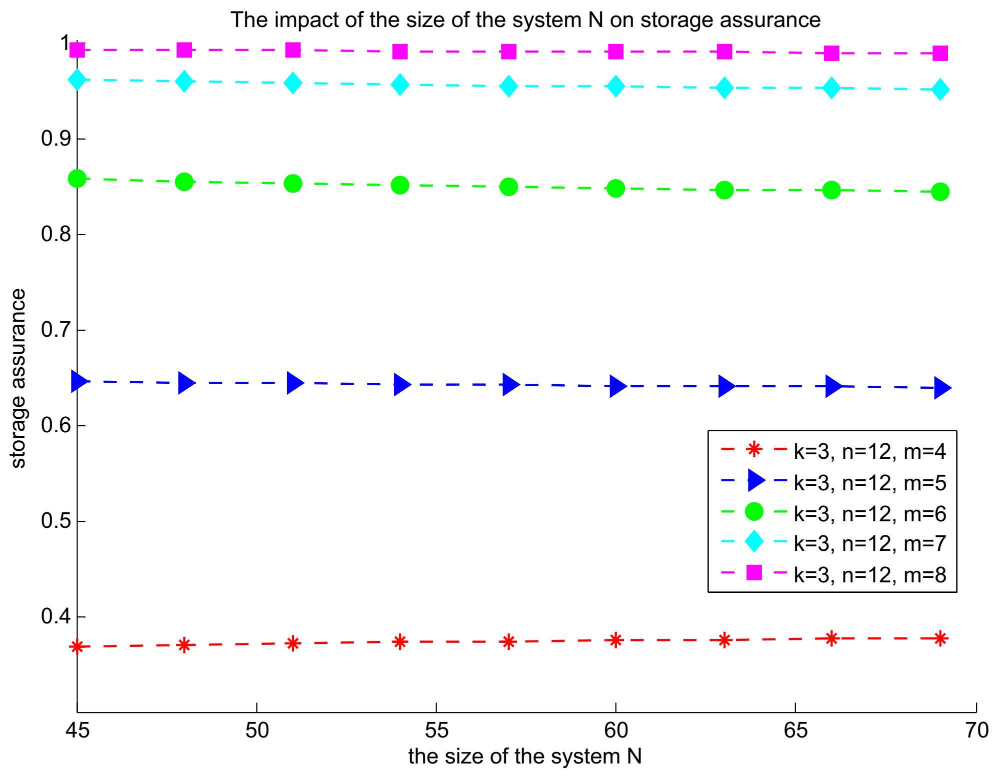

Figure 3.6: *The impact of the system size $N$ on storage assurance.*

distributed system with heterogeneous vulnerabilities, the system assurance increases significantly with the increasing values of $K$ and threshold $m$ (see Fig. 3.5). Such a trend implies that a high heterogeneity level of vulnerability gives rise to high confidentiality assurance.

### 3.4.2 Impact of System Size on Storage Assurance

To quantify the impact of system size $N$ on data assurance of a file stored in the system, we gradually increase system size from 45 to 70 by increments of 5. We keep $k$ at 3 and also vary $m$ from 4 to 8. Fig. 3.6 reveals that the storage assurance of the system is not very sensitive to the system size, indicating that storage assurance largely depends on the vulnerability heterogeneity level rather than system size. Thus, large-scale distributed storage systems with low levels of vulnerability heterogeneities may not have higher assurance than small-scale distributed systems. These results suggest that one way to improve system assurance is to increase vulnerability heterogeneity while increasing the scale of a distributed

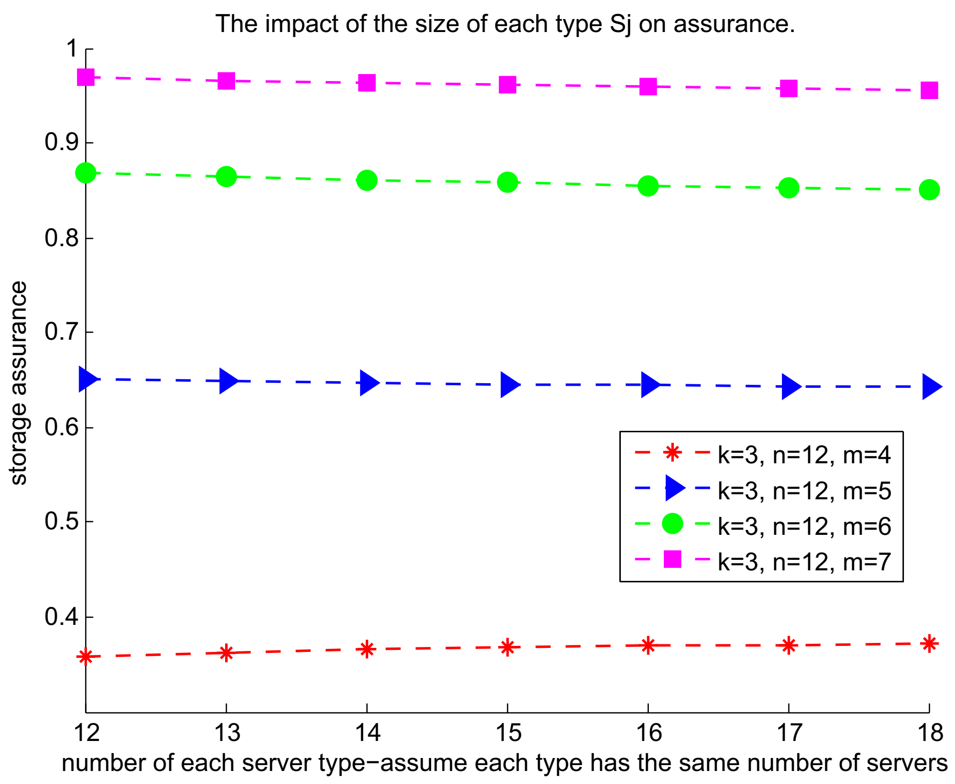

Figure 3.7: The impact of server-group size on data storage assurance. The server-group size means the number of storage nodes in a server-type group. Note that the storage nodes within a server group share the same level of vulnerability. The server-group size varies from 12 to 18 with an increment of 1.

storage system. A high heterogeneity level in vulnerability helps in increasing threshold $m$, making it harder for attackers to compromise multiple server groups and reconstruct files.

### 3.4.3 Impact of Size of Server Groups on Storage Assurance

Fig. 3.7 illustrates the impact of server-group size on data storage assurance. Note that the server-group size is the number of storage nodes in a server-type group, in which all the storage nodes share the same level of vulnerability. We vary the server-group size from 12 to 18 with an increment of 1. We observe from Fig. 3.7 that when threshold $m$ is small (e.g, $m = 4$), the assurance of systems with large server-group sizes is slightly higher than that of systems with small server-group sizes. Interestingly, the opposite is true when the threshold $m$ is large (e.g, $m > 4$). Given a fixed number of storage nodes in a distributed storage

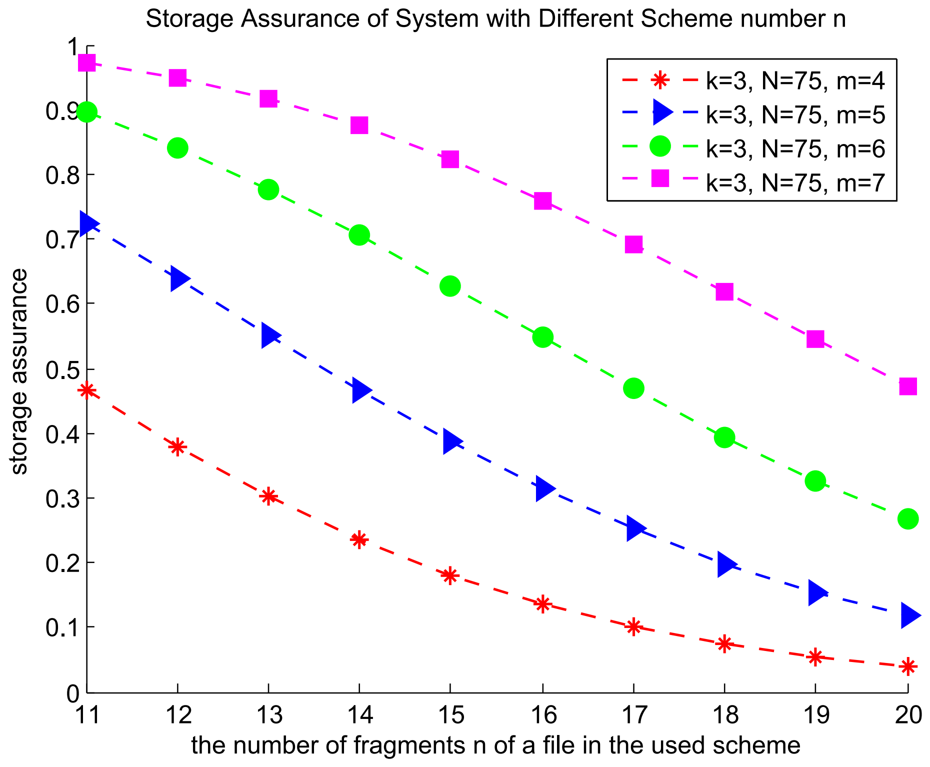

Figure 3.8: The impact of the number $n$ of fragments of a file on storage assurance. The number $n$ of fragments increases from 11 to 20. The parameters $k$ and $N$ are set to 3 and 75, respectively.

system, increasing the server-group size can decrease the number of server groups, which in turn tends to reduce vulnerability heterogeneity. The results shown in Fig. 3.7 match the results in the previous experiments in which a low level of vulnerability heterogeneity (or larger server-group sizes ) results in degraded storage assurance.

### 3.4.4 Impact of Number $n$ of File Fragments on Storage Assurance

Fig. 3.8 illustrates the impact of the number $n$ of fragments of a file on storage assurance. In this experiment, we increase the number $n$ of fragments from 11 to 20 and measured data storage assurance using our model. The parameters $k$ and $N$ are set to 3 and 75, respectively. We also vary threshold $m$ from 4 to 7. Results depicted in Fig. 3.8 confirm that the system assurance is reduced with the increasing value of fragment number $n$. The results indicate that a large number of file fragments leads to low data storage assurance of the file. This assurance trend is reasonable because more fragments are likely to be allocated to storage nodes with the same vulnerability. If one storage node is compromised by an attacker, fragments stored on nodes with the same vulnerability can also be collected by the attacker, who is more likely to be able to reconstruct the file from the disclosed fragments.

In addition, Fig. 3.8 shows that increasing the value of threshold $m$ can improve storage assurance. This pattern is consistent with the results obtained in the previous experiments.

### 3.4.5 Impact of Threshold $m$ on Storage Assurance

Figs. 3.5-3.8 clearly show the impact of threshold $m$ on storage assurance of a distributed system. More specifically, regardless of other system parameters, the storage assurance always goes up with the increasing threshold value $m$. The results indicate that the more fragments an attacker needs in order to reconstruct a file, the higher data storage assurance can be preserved for the file in distributed storage systems. These results suggest that to improve data storage assurance of a file, one needs to partition the file and allocate fragments in such a way that an attacker must compromise more server groups (the best case is m server groups) in order to reconstruct the file.

### 3.4.6 Impact of $P_L$ on Dynamic Assurance

Now we are in a position to evaluate dynamic assurance of distributed storage systems. The three parameters $\mu_1$, $\mu_2$, and $\mu_3$ in Eq. 3.7 have an important impact on dynamic assurance because these parameters indicate whether there is risk during fragment transmissions. Please refer to Sections 6.5.1 to 3.4.5 for details on the impacts of a set of parameters on data storage assurance.

$P_L$ - the probability that a fragment might be intercepted by an attacker during the fragment's transmission through an insecure link - has a noticeable impact on dynamic assurance of a distributed storage system provided that threshold $m$ is small (e.g., smaller than 9). Fig. 3.9 shows the dynamic assurance of a distributed system when $P_L$ is varied

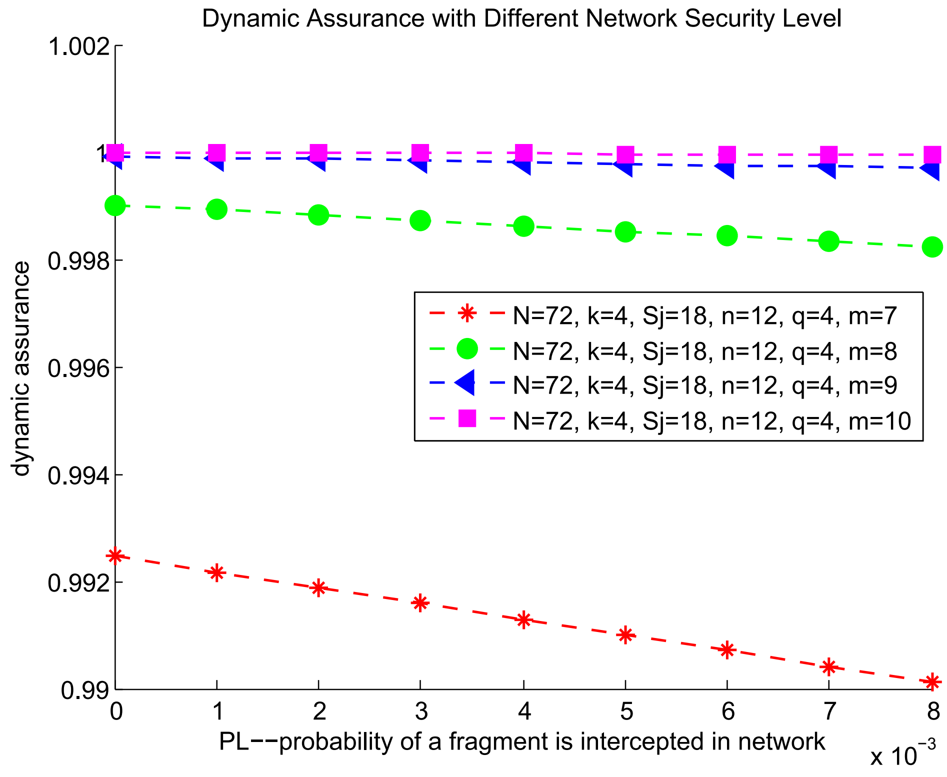

Figure 3.9: Impact of $P_L$ - the probability that a fragment might be intercepted by an attacker during the fragment's transmission through an insecure link. $P_L$ is varied from 0 to $8 * 10^{-3}$ by increments of $1 * 10^{-3}$. Threshold $m$ is varied from 7 to 10

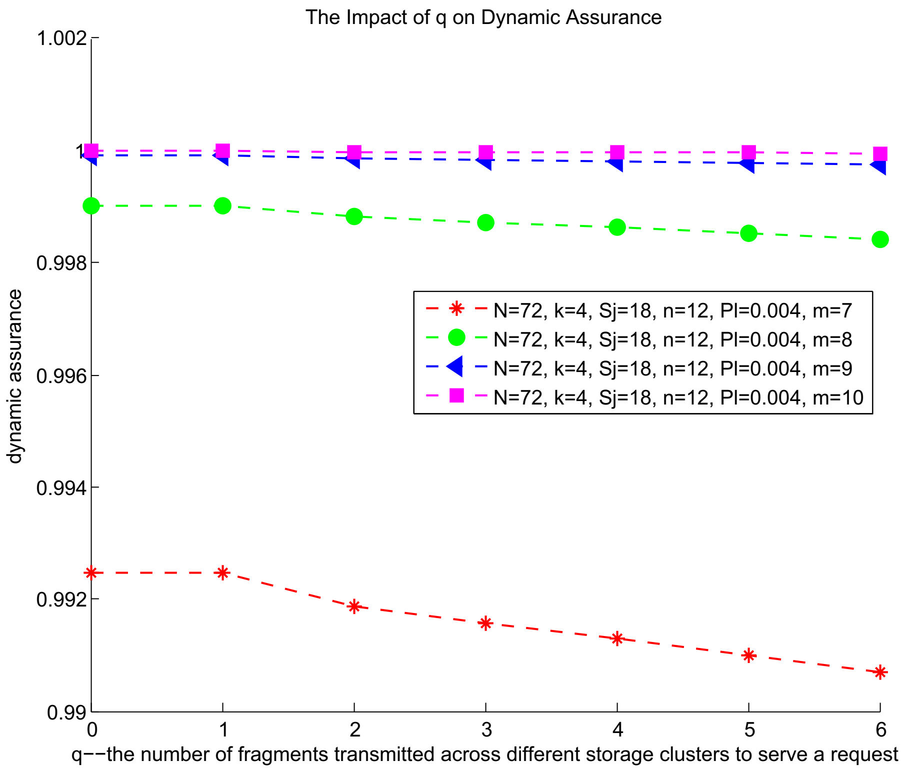

Figure 3.10: Impact of $q$ - the number $q$ of fragments transmitted to and from a storage cluster. $q$ is chosen from 0 to 6 with an increment of 1. Threshold $m$ is set from 7 to 10)

from 0 to $8 * 10^{-3}$ by increments of $1 * 10^{-3}$. We also vary threshold $m$ (i.e., $m$ is varied from 7 to 10) to evaluate the sensitivity of dynamic assurance on parameter $P_L$ under different threshold $m$.

Fig. 3.9 demonstratively confirms that when threshold $m$ is equal to or smaller than 8, a large value of $P_L$ results in low dynamic assurance of the system. The results are expected since a high value of $P_L$ means that the transmitted fragments are likely to be intercepted by an attacker. Once the attacker has collected enough fragments of a security-sensitive file, the file could be reconstructed. When threshold $m$ is larger than 8, the dynamic assurance is not noticeably sensitive to the probability $P_L$ that a fragment is compromised during its network transfer.

### 3.4.7 Impact of $q$ on Dynamic Assurance

Like parameter $P_L$, the number $q$ of fragments transmitted to and from a storage cluster also has an impact on the dynamic assurance of a distributed storage system. Intuitively, Fig. 3.10 shows that when the number of fragments of a file that must be transmitted through insecure links is increasing, the dynamic assurance of the file drops. Interestingly, when threshold $m$ is larger than 8, the dynamic assurance becomes very insensitive to the number $q$ of fragments. This observation suggests that when the threshold is small, the S-FAS fragment allocation scheme must pay particular attention to lower the value of $q$ in order to maintain a high dynamic assurance level.

In addition, we observe from Fig. 3.10 that dynamic assurance is always lower than the corresponding storage assurance (where q=0 in Fig. 3.10). This trend is always true because in a dynamic environment, file fragments have to be transmitted through insecure network links where malicious users may intercept the fragments in order to reconstruct files.

## 3.5 Chapter Summary

It is critical to maintain the confidentiality of files stored in a distributed storage system, even when some storage nodes in the system are compromised by attackers. In recognizing that storage nodes in a distributed system have heterogeneous vulnerabilities, we investigated a secure fragment allocation scheme by incorporating secret sharing and heterogeneous vulnerability to improve security of distributed storage systems.

We addressed the security heterogeneity issue by categorizing storage servers into different server-type groups (or server group for short), each of which represents a level of security vulnerability. With heterogeneous vulnerabilities in place, we developed a fragment allocation scheme called S-FAS to improve security of a heterogeneous distributed system. S-FAS allocates fragments of a file in such a way that even if attackers compromised a number of server groups but fewer than k fragments are disclosed, the file cannot be reconstructed from the compromised fragments.

To evaluate the S-FAS scheme, we built the static and dynamic assurance models in order to quantify the assurance of a heterogeneous distributed storage system processing file fragments. We developed a SAP file allocation algorithm based on the analysis of the assurance model as well as the proposed S-FAS scheme. In order to measure the performance of the S-FAS scheme and the algorithm, we built a prototype in a real-world distributed storage system.

We demonstrated how S-FAS incorporates the vulnerability heterogeneity feature into file fragment allocation for distributed storage systems. Experimental results show that increasing heterogeneity levels can improve file assurance in a distributed storage system. The experimental results of our prototype implementation offer us inspiration on how to use S-FAS to efficiently improve security and performance in distributed storage systems with heterogeneous vulnerabilities.

There are three future research directions of this study. First, we will make an effort to improve the performance of the SFAS fragment allocation scheme in a heterogeneous distributed system. Second, we will integrate the data replication technique with S-FAS to enhance reliability and performance of the fragment allocation scheme for distributed systems. Third, we will implement a distributed storage system prototype where S-FAS is deployed. In this prototype, we will evaluate performance of S-FAS in a real-world system.

# Chapter 4

# Secure Allocation Processing (SAP) Algorithm for S-FAS

In previous chapter we present a fragment allocation scheme called S-FAS to improve security of a distributed system where storage sites have a wide variety of vulnerabilities. However the S-FAS scheme mainly focus on security considerations, the heterogeneous features can also be leveraged to improve performance.

In this Chapter we develop a secure allocating processing (SAP) algorithm for the S-FAS scheme to improve the security level and consider its performance using the heterogeneous feature of a large distributed system. The SAP allocation algorithm considers load balancing, delayed effects caused by the workload variance of many consecutive requests, and the heterogeneous feature of the storage nodes in the system.

The rest of the chapter is organized as follows: We discuss in Section 4.1 the motivation to develop the SAP algorithm. Section 4.2 describes the factors that affect performance and security in distributed systems. Section 4.3 describes a static allocation algorithm integrated with the S-FAS scheme. In Section 4.4, a sample allocating process is illustrated. Section 5.4 summarizes this chapter and outlines some future work directions.

## 4.1 Motivation for the Secure Allocation Processing (SAP) Algorithm

There are tradeoffs among desired features(e.g., security and performance) of distributed storage systems. Most existing solutions improve the security of the systems at the cost of system performance. However, both high security and performance are among the top client desired features in many widely deployed data centers operated by Google, Amazon and Yahoo, etc.. The exploration and development of solutions that can improve not only system security, but also system performance is in high demand.

In our previous research [124] we proposed a scheme, S-FAS, to address security heterogeneity issues by dividing storage servers into different server groups. We focus on the development of a file fragmentation and allocation approach to improving the assurance and scalability of a heterogeneous distributed system. If one or more fragments of a file have been compromised, it is still very hard for a malicious user to reconstruct the file from the compromised fragments. Our solution is different from previous approaches, because ours captures heterogeneous features regarding to vulnerabilities among servers. In a server group, storage servers with the same vulnerability share the same weakness allowing attackers to reduce the servers' information assurance. Although it may be impractical to classify all servers in a system into a large number of groups, a reasonable way of identifying server types is to organize servers with similar vulnerabilities into one group. We built static and dynamic assurance models to evaluate the security level of our S-FAS scheme. Analysis results show that the system assurance level can be improved by the S-FAS scheme compared with the traditional methods without the consideration of heterogeneous features.

The time cost to reconstruct a file from its fragments is obviously greater than that of the non-fragmentation storage method; and performance degradation becomes inevitable [129]. In this research we investigate impact of secure fragmentation scheme on system performance. Similar research work on file assignment can be found in the literature [71] [30]. These studies showed that not only the queuing cost, and file heat of the file (represents the product of file access rate and the access service time) [30] [108], but also the variance of service time can influence the performance of homogeneous systems where each site has the same performance characteristics. In our investigated heterogeneous distributed systems, the variance in service time caused by heterogeneous servers with different performance characteristics has a large influence on the overall system performance.

We investigate the possible parameters that influence both the system performance and the proposed S-FAS scheme. There are three aspects to a distributed storage system that influence its security and performance, workload, storage nodes and network interconnects.

There are different elements of each aspect of the system. We extracted the possible key elements of each aspect by analyzing of the proposed S-FAS scheme.

Based on the both performance and security analysis, we developed a secure allocating processing (SAP) algorithm for the S-FAS scheme to both improve the security level and consider system performance by using the heterogeneous feature of large distributed systems. The SAP allocation algorithm considers load balancing, delays caused by the workload variance of many consecutive requests, and the heterogeneous nature of storage nodes in a system. We developed a prototype of S-FAS using the multi-threading technique to implement the SAP algorithm to guide file allocations. The experiment results show that the proposed security solution can not only improve security, but also improve the throughput of a distributed storage system with heterogeneous vulnerabilities by virtue of the multi-threading.

In this study, we made the following contributions:

- First, we develop a secure allocating processing (SAP) algorithm to improve security and system performance by considering the heterogeneous feature of a large distributed system.
- Second, in order to conduct the performance analysis for the S-FAS scheme and SAP allocating algorithm, we developed a prototype for our S-FAS scheme.
- Third, we implemented the prototype and conducted some experiments on the throughput of the proposed scheme and algorithm.
- Fourth, we evaluated our solution by implementing some real world traces.

## 4.2 Factors Affecting Performance and Security

There are three main sources affecting the processing delay of a request from a client: workload, storage nodes, and network interconnects.

Before presenting details on the SAP algorithm, let us summarize all notations used throughout this section in Table 4.1.

Table 4.1: Notation used in the SAP Algorithm.

| Notations | Meaning |
|---|---|
| $S$ | the size of file $F$ |
| $u$ | server type in the system |
| $S_i$ | the size of server type $i$ in a cluster |
| $\|F\|$ | the total number of files |
| $\|F_i\|$ | the total number of fragments in file $\|F_i\|$ |
| $Fij$ | the $j$th fragment of the $i$th file |
| $Burden_{ij}$ | the work load that the $j$th fragment of the $i$th file brings to the node where it is stored |
| $DataSize_{ij}$ | the $j$th fragment size of the $i$th file |
| $b_N$ | Bandwidth of the connected network of the Nth node |
| $\lambda_{ij}$ | the access rate of the $j$th fragment of $i$th file |
| $N_{uv}$ | the $v$th node of server type $u$ u,v= 0,1... |
| $I_{ij}$ | Decreasing sorted list of fragments to be allocated (Fragments of the same file are consecutive and belong to the same row) |
| $TLoad\,(N_{uv})$ | total available load of node $N_{uv}$ |
| $CLoad\,(N_{uv})$ | current load of node $N_{uv}$ |
| $RLoad\,(N_{uv})$ | $N$ number of element vector recording the total available storage capacity for all nodes |
| CurrentN[u] | the current available node in the $u$th type of servers |
| LoadBN$_{uv}$ | the most load that shoud be assigned to node N$_{uv}$ |

Concerning the workload, the service time, $S_i$, for each file $F_i$, and the access rate of this file, $A_i$, are the two main characteristics that affect the performance on the file. These two characteristics are usually combined in a joint metric called "heat" to evaluate the active rate of a file [68]. The service time is directly influenced by file size and two parameters-m and n-in the threshold (m, n). Since we consider the performance of heterogeneous systems, service time $S_i$ for file $F_i$ is not fixed, because different allocation nodes are chosen to store the fragments of the file.

In order to fully benefit from the performance capabilities of a homogeneous distributed or parallel system, one has to insure that the load must be uniformly distributed among all storage nodes. Otherwise, some disks may become performance bottleneck, severely increasing the response time of requests, and reducing the overall system throughput [68]. As such reducing the time cost on network and slave nodes may be achieved by minimizing the utilization of each node and by minimizing the variance of service times among all the nodes. Most of the current published work on this topic concentrated on minimizing the nodes utilization by balancing load across all disks while ignoring the minimization of the variance of service time in the context of heterogeneous distributed systems. When fragments of a wide variety of sizes are intermixed on each node, small fragment requests are likely to wait for large fragment requests accesses that were queued ahead of the smaller ones. This is inefficient, especially when the load is heavy and the queuing delays dominate the response time. Thus, in addition to load balancing, the performance of a distributed system can be improved by reducing the variance of service times among fragment storage nodes. System network interconnects also affect performance of heterogeneous distributed systems. Thus, our file fragment allocation algorithm has to take the network delays into account.

Experimental results described in section show that diversity of a system with heterogeneous features can possibly increase the system security. To make use of the heterogeneous nature of vulnerabilities in storage nodes, the SAP algorithm should allocate the fragments of a file to as many types of storage nodes as possible to improve the system security.

While considering the impact of different bandwidths of networks, we define the coefficient $w_N$ (See weight in Eq. 4.1). Equation 4.1 is used to determine load to be distributed to the $N$th node concerning the network speed variance of different storage node. This equation helps to guarantee that storage nodes connected by fast networks handle more I/O loads.

$$w_N = \frac{b_N}{\sum_{i=1}^{N} b_N} \tag{4.1}$$

Based on workload analysis, we use *burden*(see Eq. 6.7 below) to evaluate the workload that a fragment brings to the storage node that stores the fragment.

$$Workload_{ij} = \lambda_{ij} * DataSize_{ij} \tag{4.2}$$

In order to balance the loads imposed on different nodes in the system, we distribute the workload expressed by Eq. 6.8 where a number of fragments of multiple files are stored in node N.

$$Workload\_N = w_N * \sum_{i=1}^{i=d} \sum_{j=1}^{j=|F_i|} \lambda_{ij} * DataSize_{ij} \tag{4.3}$$

## 4.3 SAP Allocation Algorithm

We developed a static allocation algorithm integrated with the S-FAS scheme. we focus on the static algorithm rather than its dynamic counterpart, because we plan to consider the movement of data in our future work.

Initially, all nodes within the same server type are listed in a decreasing order of processing speed under a certain workload. All files are sorted in list $I$ in a descending order of fragment sizes. Then, the nodes are allocated to the file fragments in order to optimize both performance and security. Fig. 4.2) shows the data flow of the SAP algorithm. The algorithm includes the partitioning and sorting part(see Fig. 4.1), and fragment allocation steps(see Algorithm 1).

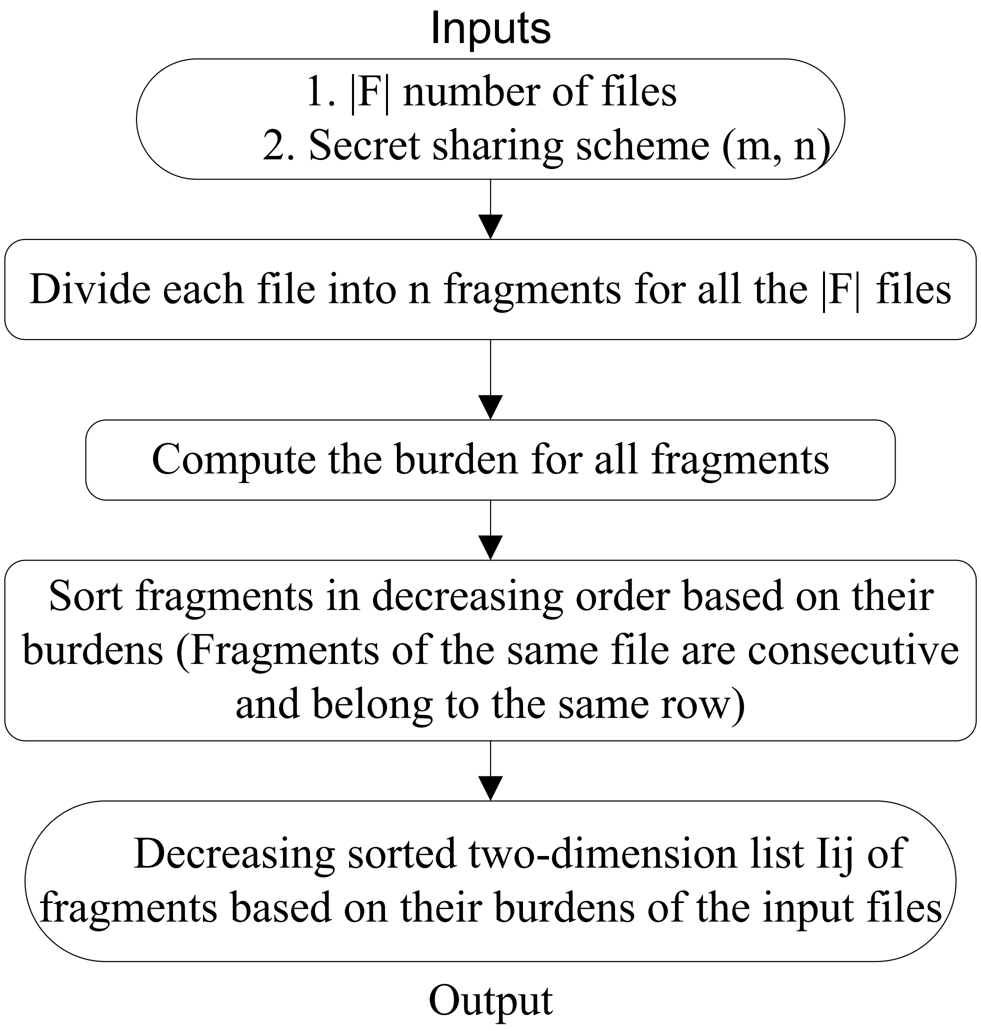
Inputs
1. |F| number of files
2. Secret sharing scheme (m, n)
Divide each file into n fragments for all the |F| files
Compute the burden for all fragments
Sort fragments in decreasing order based on their burdens (Fragments of the same file are consecutive and belong to the same row)
Decreasing sorted two-dimension list Iij of fragments based on their burdens of the input files
Output

Figure 4.1: *The SAP fragment Partition and Sorting Step. The chosen secret sharing scheme (m, n) is employed.*

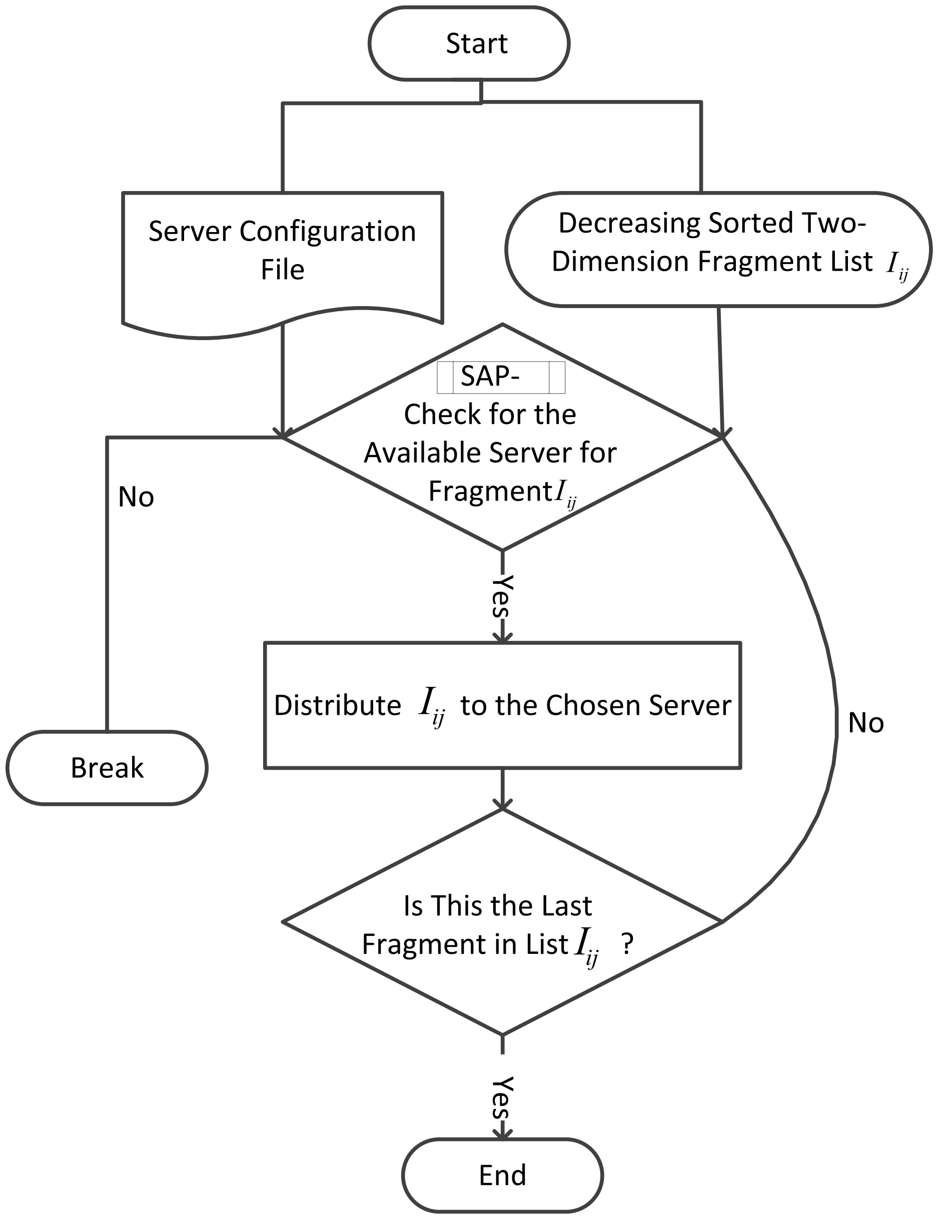
Start
Server Configuration File
Decreasing Sorted Two-Dimension Fragment List $I_{ij}$
SAP-
Check for the Available Server for Fragment $I_{ij}$
No
Yes
Distribute $I_{ij}$ to the Chosen Server
Break
No
Is This the Last Fragment in List $I_{ij}$ ?
Yes
End

Figure 4.2: *The SAP Data Flow.*

**Algorithm 1** SAP Allocating Process Step:

```
Input:
$I_{ij}$
$CLoad\ (N_{uv})$
$TLoad\ (N_{uv})$

Step1. Sort all nodes in a two-dimension list in decreasing order based on their vulnera-
bility type and processing speed.

Step2. Allocate each fragment in list I to servers
$i$=1
for $u = 0$ and $u < K$ do
  CurrentN[u] = 0
end for
$u = 0$
$v$ = CurrentN[u]
CurrentN [u] = $u$
use the equations for $w_N$ and $NBurden_{ij}$ to calculate LoadB N$_{ij}$
for $i = 1$ and $i < |F|$ do
  //Allocate all fragments of file i
  for $j = 1$ and $j < |F_i|$ do
    if $CLoad\ (N_{uv}) \geq$ LoadBN$_{uv}$ then
      CurrentN [u] + +
    end if
    $CLoad\,(N_{uv})$= $CLoad\,(N_{uv})$+$Burden_{ij}$
    $u + +$
    v = CurrentN[u]
    if $u = K$ then
      $u = 0$
    end if
  end for
end for

Output:
The updated fragment list for multi-thread fragment writing
```

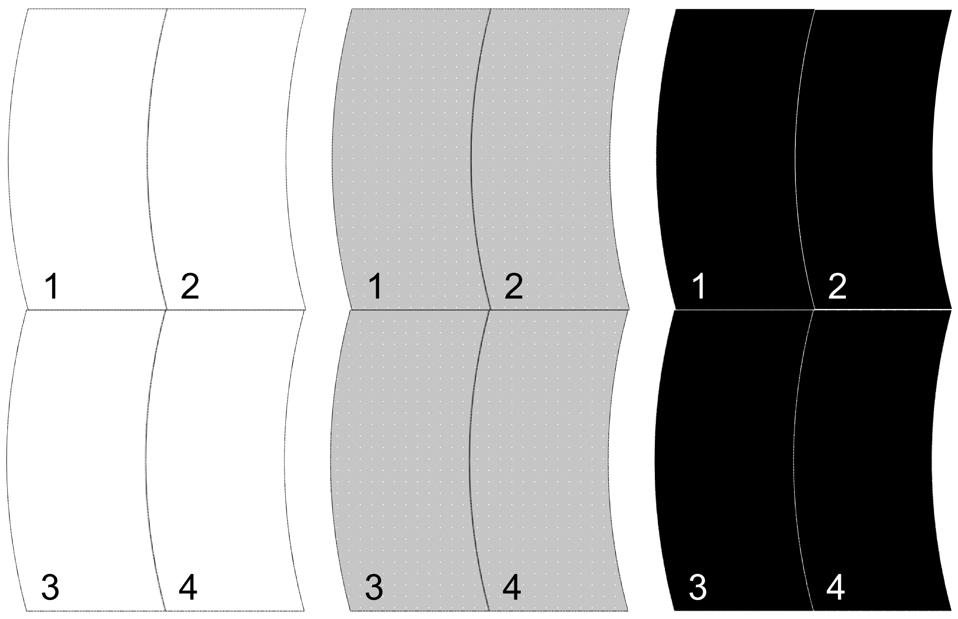

Figure 4.3: Allocation Load Sample

## 4.4 An Allocation Example for SAP

Let us use an example to demonstrate the process of assigning storage nodes to file fragments. There are three files and each file has four fragments to be allocated(see Fig. 4.3). The data size, fragment size, access rate, and file heat are listed in Table 4.2. Here we assume the size unit of fragment or file is megabyte(MB). The access rate is measured by how many times' requests to a file per second. Heat is defined as in [68] to evaluate the active rate of a file. Here we assume the heat value of the hottest file in a system to be 100, and the heat values of other files are comparably set to values from 0 to 100. Because our analysis is independent of the unit of each item as previously described, we do not list the units in Tables 4.2, 4.3 and 4.4.

We assume that there are three server types, each of which contains three servers. The network bandwidth of each server is listed in the second column in Table 4.3. According the

Table 4.2: Features of Example Files

| Aspects of the File | $F1$ | $F2$ | $F3$ |
|---|---|---|---|
| FileSize | 80 | 60 | 40 |
| Fragment Size | 20 | 15 | 10 |
| AccessRate | 3 | 6 | 5 |
| Heat | 60 | 90 | 50 |

Equations 4.1, 6.7 and 6.8, we can calculate each node's upper bound of affordable burden (see the third column in Table 4.3) for these files to be allocated.

Table 4.3: Features of a Heterogeneous Storage System Example

| Nodes | Bandwidths Weight | BurdenAfford–Upbound |
|---|---|---|
| $N_{11}$ | 1 | 40 |
| $N_{12}$ | 2 | 80 |
| $N_{13}$ | 3 | 120 |
| $N_{21}$ | 4 | 160 |
| $N_{22}$ | 1 | 40 |
| $N_{23}$ | 2 | 80 |
| $N_{31}$ | 3 | 120 |
| $N_{32}$ | 3 | 120 |
| $N_{33}$ | 1 | 40 |

Fig. 4.4 shows the allocation results made by the SAP algorithm described in Fig. 4.1 and Algorithm 1.

Table 4.4 summarizes the allocated burden and the overloaded load imposed on each node in the system. The negative and positive symbols mean that final allocated burdens exceed or below expected bounds. The nodes that afford a little bit of more load for these bunches of files, it may be assigned less amount of load while allocating other bunches of files. The allocation result deviation will be neutralized during the whole system's working process.

## 4.5 Chapter Summary

It is critical to maintain the confidentiality of files stored in a distributed storage system without significantly degrading performance. We developed a SAP file allocation algorithm

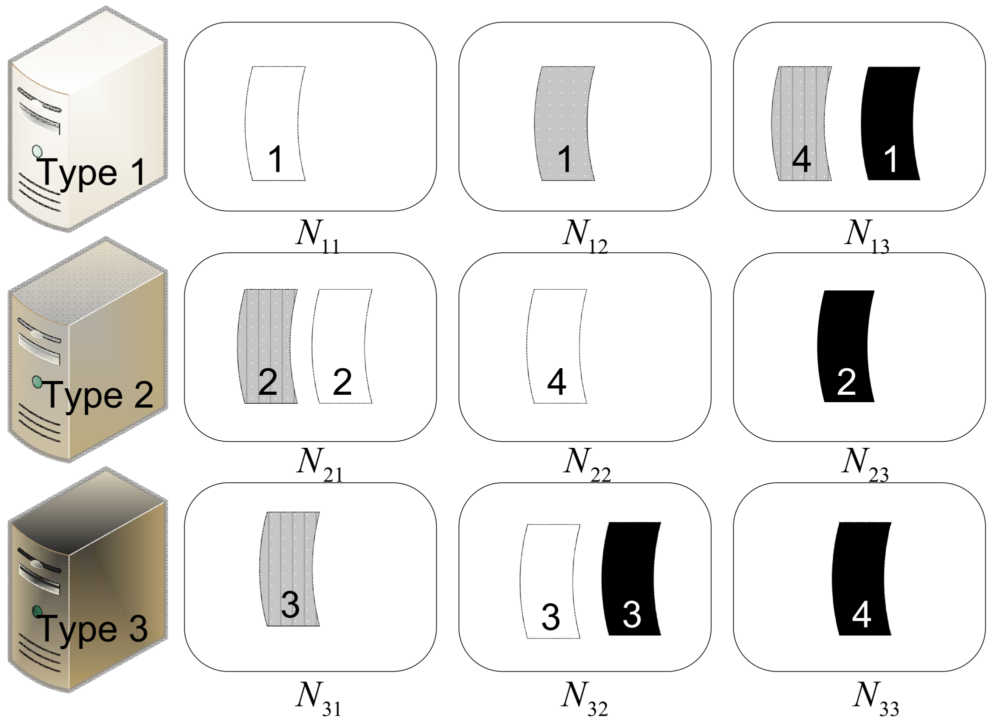

Figure 4.4: Alocation Result

based on the analysis of the assurance model and the proposed S-FAS scheme proposed in our previous research. In order to evaluate the performance of the S-FAS scheme and the algorithm we built a prototype. We implemented the prototype using C programming language and conducted some experiments on the throughput of the proposed scheme and algorithm. At last we evaluated our solution by implementing some real world traces.

There are a few future research directions of this study. First, we will integrate a data replication technique to be integrated with S-FAS to enhance reliability and performance of the fragment allocation scheme for distributed systems. Second, we will develop a dynamic file allocation algorithm for a distributed system where data replicas are achieved and maintained. Third, we will make an effort to extend the SAP algorithm to address security issues into a cloud storage system. Last but not least, we will continue to upgrade our prototype for future scalability testing.

Table 4.4: Allocation Result by the SAP Algorithm

| Nodes | BurdenAllocated | BurdenOverBound |
|---|---|---|
| $N_{11}$ | 60 | 20 |
| $N_{12}$ | 90 | -10 |
| $N_{13}$ | 140 | -20 |
| $N_{21}$ | 150 | +10 |
| $N_{22}$ | 60 | +20 |
| $N_{23}$ | 50 | +30 |
| $N_{31}$ | 90 | +30 |
| $N_{32}$ | 110 | +10 |
| $N_{33}$ | 50 | -10 |

# Chapter 5
# S-FAS Prototype

In previous chapter we present a secure allocating processing (SAP) algorithm for the S-FAS scheme to improve the security level and consider its performance using the heterogeneous feature of a large distributed system.

In order to use practical implementations to demonstrate the ideas on actual systems with real-world applications, we developed a prototype using the multi-threading technique and C language for the S-FAS scheme with the SAP algorithm to guide the file allocation. The prototype is built in the distributed cluster environment with heterogeneous storage nodes, in which the Network File System (NFS) and Linux are installed. We did some experiments on system throughput and testing against real world traces. The evaluation results show that the proposed solution can not only improve the security level, but also improve the throughput and performance of the distributed storage systems with heterogeneous vulnerabilities by using the multi-thread technique.

The rest of the chapter is organized as follows: We present in Section 5.1 the detailed design of the prototype. Section 5.2 describes the evaluation of the prototype on system throughput in distributed systems. In Section 5.3 we introduce the performance evaluation by testing against real world traces. Section 5.4 concludes this chapter and presents our future research directions.

## 5.1 Prototype Design

Fig. 5.3 illustrates system architecture of a heterogeneous distributed system, where nodes are classified into three main groups - computing nodes (clients), storage nodes, and a storage server. The storage server is responsible for allocating file fragments using our

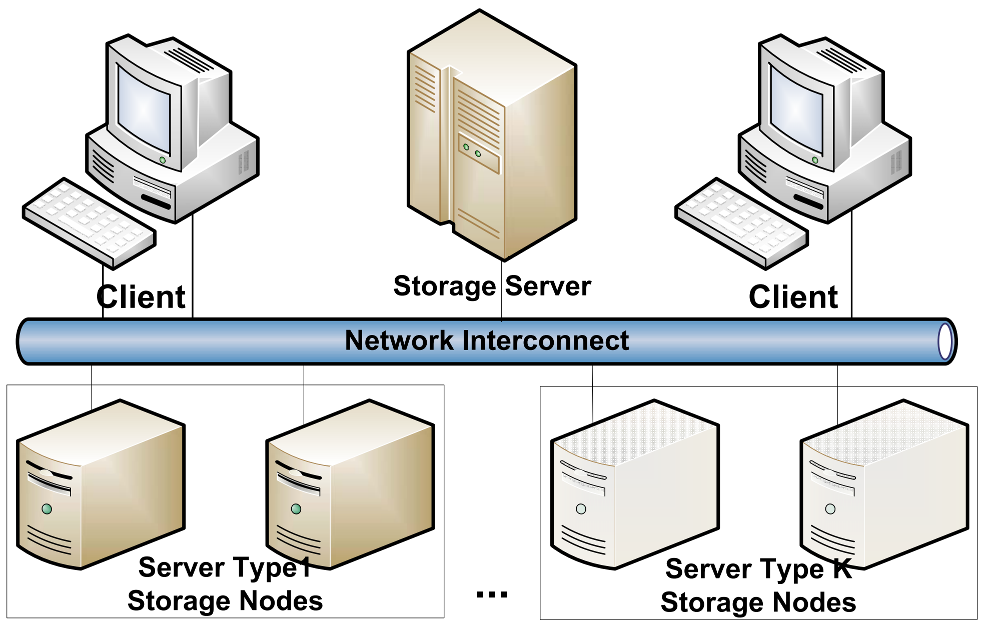

Figure 5.1: *System architecture.*The storage server manages metadata. The storage nodes are logically grouped into different server types. Client nodes can directly access storage servers through the network connection.

SAP algorithm, managing metadata, and dealing with incoming file requests(e.g., reads or writes) from clients. The storage nodes are divided into several groups; nodes in each group share similar heterogeneous vulnerability. The client nodes can directly access storage servers through the network interconnections.

Fig. 5.4 shows the modules implemented in the prototype, which is comprised of the following components.

- The first part is a set of configuration files for servers, files, and all fragments of all file fragments stored in storage nodes.
- The second component is the active lists for servers (see the data structure of the example server node, which forms the above server list) , files, and all fragments in the memory of storage node.

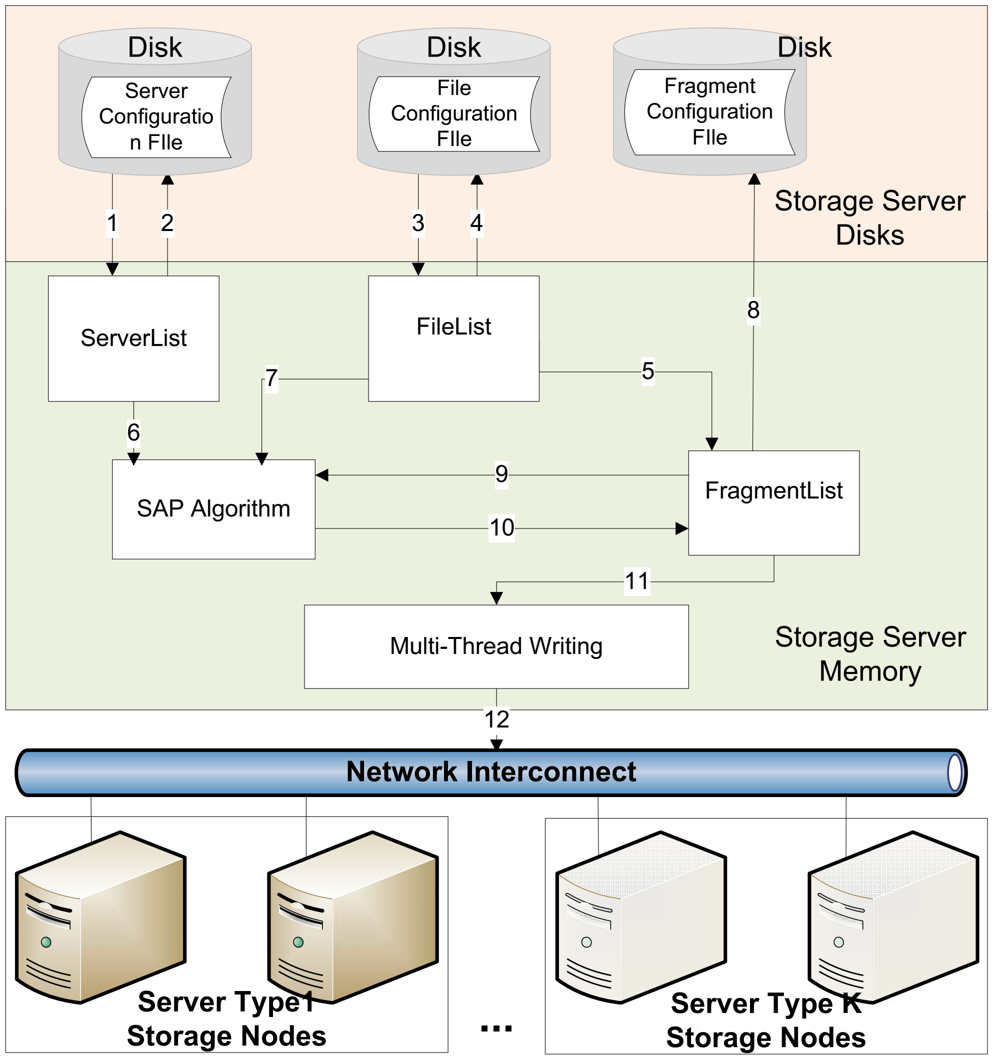

Figure 5.2: *The Prototype design.*The upper part of this figure represents the disks of a storage server where the storage node configuration file, file configuration file, and fragment configuration file are stored. The middle part is the memory part of the storage server node, where the key allocating module parts such as the SAP algorithm, multi-thread writing, and the three active link lists are stored. The three active lists records the most updated information for storage nodes, stored files and fragments in the system. The lower layer in the figure represent the client nodes in the system.

- The third part is the SAP algorithm and multiple threads located in the storage server.
- The last component is composed of distributed storage nodes which are used to store file fragments.

The prototype runs on the storage server node and directly communicates with the client nodes. The upper layer in Fig. 5.4 represents the disks of the storage node, where the server configuration file, the file configuration file, and the fragment configuration file are stored. The middle layer is residing in the main memory part of the server node where the key allocation modules such as the SAP algorithm, multiple writing threads, and the three active link lists are stored. The three active lists record the most recent information of the server, stored files and fragments in the system. The bottom layer in the figure represents the client nodes in the system.

### 5.1.1 Architecture and Modules

This prototype was designed to improve the performance using the S-FAS scheme and SAP algorithm applied in distributed storage system.

#### Architecture

Fig. 5.3 illustrates the architecture of Security Improved Distributed Storage System (SIDSS) , where nodes are divided into three main groups - compute nodes (clients), storage nodes, and a storage server. The storage server is responsible for handling file allocating using our SAP algorithm, managing the metadata and dealing with the incoming file requests for data reads or writes from the clients. The storage nodes are divided into a few groups based on their heterogeneous vulnerability features. Client nodes can directly access storage servers through the network interconnect.

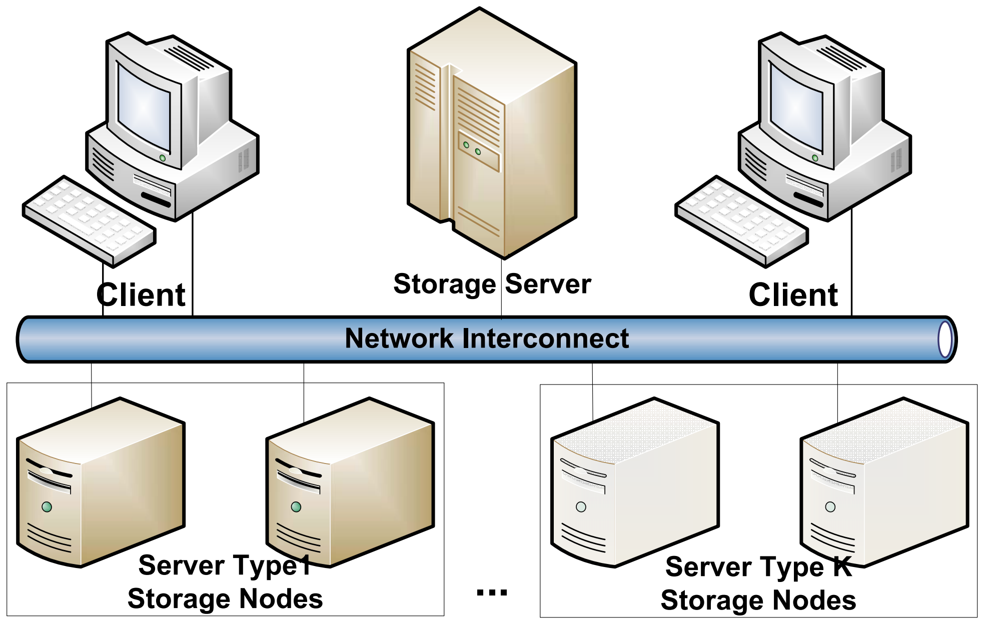

Figure 5.3: *The architecture of the system.*The storage server manages metadata (e.g., configuration files for server nodes, stored files and fragments. The Storage node also runs the NFS).The storage nodes are logically grouped into different server types (server nodes of the same server type share the similar vulnerabilities.) Client nodes can directly access storage servers through the network interconnect.

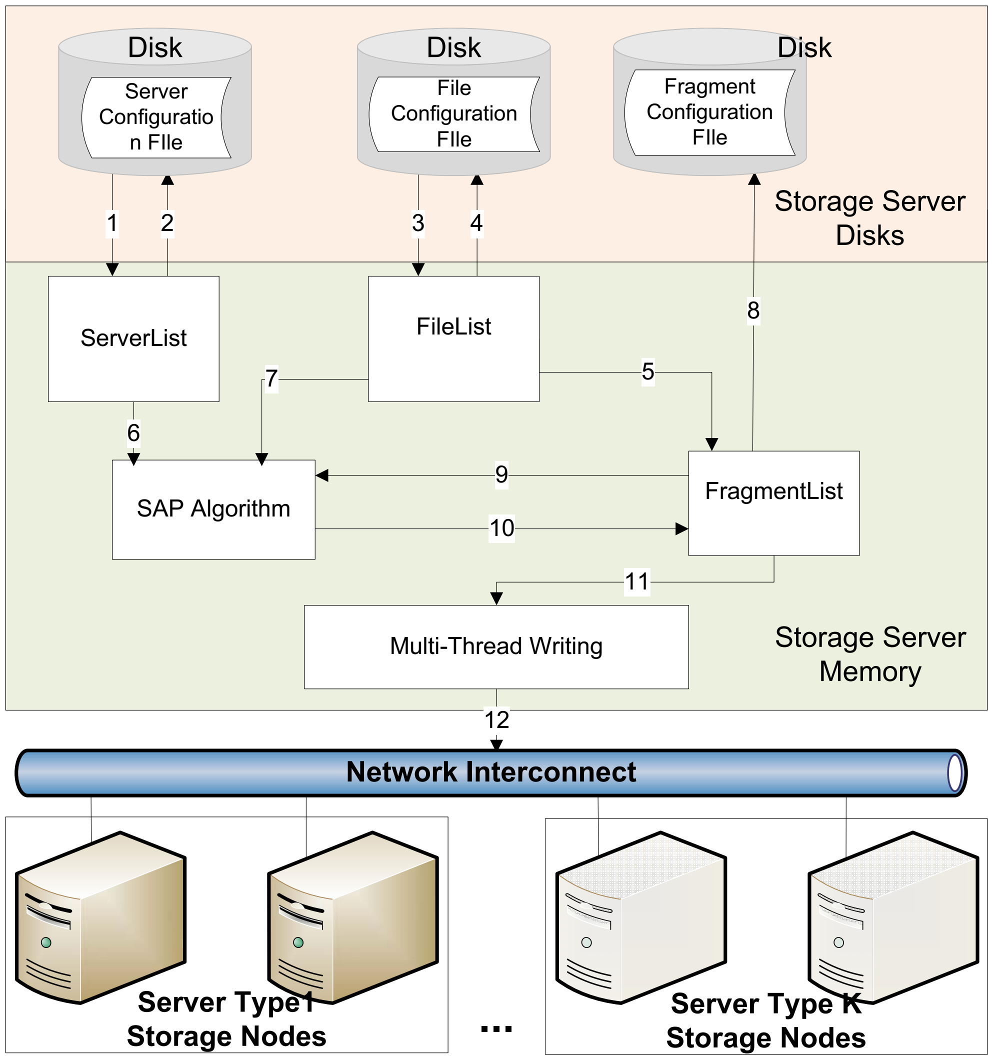

Figure 5.4: *The allocating module.* The allocating module runs on the storage server node and directly communicate with the client nodes. The upper part of this figure represents the disks of the storage node where the server configuration file, file configuration file and the fragment configuration file are stored. The middle part is the memory part of the storage server node, where the key allocating module parts such as the SAP algorithm, multi-thread writing and the three active link lists are stored. The three active lists records the most updated information for server, stored file and fragments in the system. The lower layer in the figure represent the client nodes in the system.

### Allocating Module

Fig. 5.4 illustrates the architecture of the file allocating module, which is comprised of the following four parts.

- The first part is the configuration files for servers, files and all fragments of all files on the disk of storage server.
- The second part is the active lists for servers(see the data structure of the example server node which forms the server list above) , files and all fragments in the memory of storage node.
- The third part is the SAP algorithm and multi-thread writing part, which is also active in the memory of the storage server.
- Distributed storage nodes: the storage node where the fragments of files is finally stored.

### Reads and Writes Processing Module

Fig. 5.5 illustrates the architecture of the reads and writes or trace processing module, which is mainly comprised of the following four parts.

- The first part is the configuration files for servers, files and all fragments of all files on the disk of storage server.
- The second part is the active list of all fragments in the distributed system.
- The third part is the SAPreading and SAPwriting part, which is also active in the memory of the storage server. Distributed storage nodes: the storage node where the fragments of files are finally stored

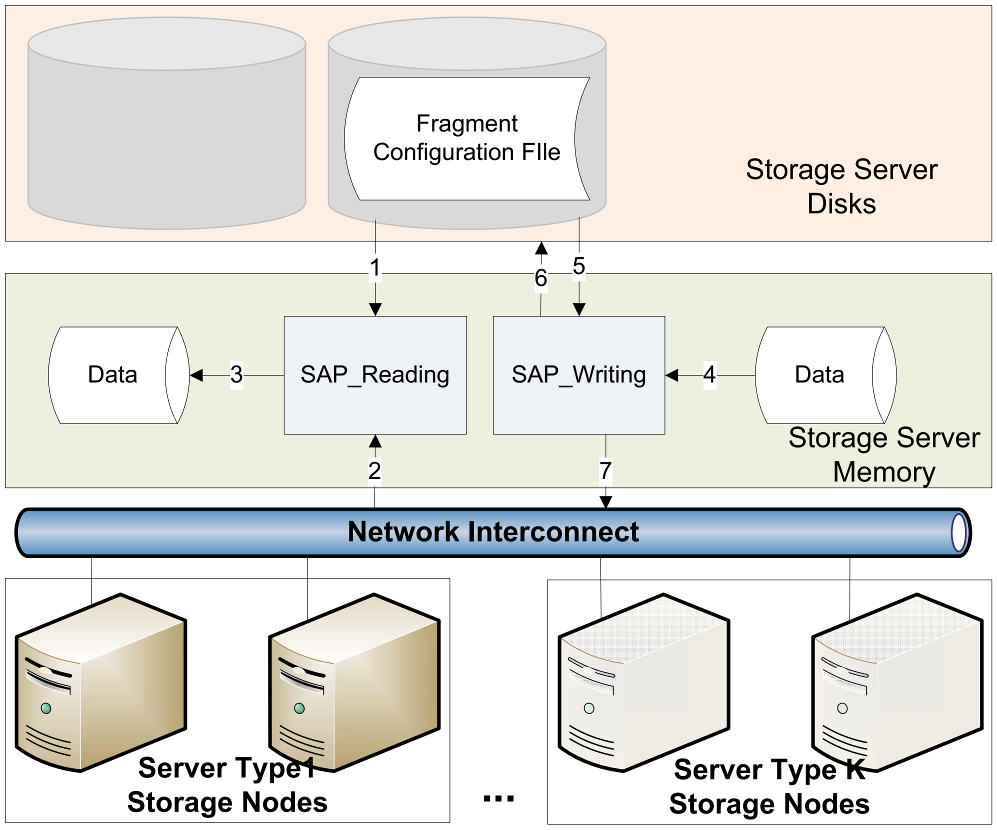

Figure 5.5: *The reads and writes/tracing module.*The reads and writes/tracing module runs at the storage server node. The SAP reading and/or writing part deal with the requests from clients or trace records.

### 5.1.2 Data Flow

Before we describe the data flow for our prototype we would like to give some assumption that may help understand this project better. The main object of this project is to evaluate the performance of our proposed S-FAS security improving scheme. The allocating means we need to put some new data into the system and store there. The read processing happens when the client assume the data they request is already stored in the system. We consider the write operation here in our current prototype only as update.

For the reads/writes processing, when clients request an operation on a file that is not exist in the system, a reminder telling that the file is not exist yet will return to clients. In future work, we may consider adding the function of dealing with files that are not exist in the system for write operation.

#### Allocating Module Data Flow

The allocating module works at the condition we need to put more files(data) into the system. This module mainly runs at the storage server node. After the storage server node receives an allocating for new storage request, there are the following few steps to be carried out to fullfill the new storage request.

- Step A(includes marked item 1, 2, 3, 4 in the Fig. 5.4 ). The allocating module initializes the server list according the configuration file information on the disk of the storage node, and file list according the allocating request from the client;
- Step B(includes marked item 5, 6, 7, 8, 9, 10 in the Fig. 5.4). . Then the allocating module runs the SAP allocating algorithm to get the arrangement plan of the new data, at the same time the fragment list and fragment configuration file are updated.
- Step C(includes marked item 11, 12 in the Fig. 5.4). Then the multi-thread writing function is run to write the fragmented new data to the arranged storage server according the arrange information in the fragment list.

**Reads and writes Processing Module Data Flow**

The reads and writes/tracing module runs at the storage server node. The SAP reading and/or writing part deal with the requests from clients or trace records. Firstly this module identify whether the request is read or write. Then it will check the active fragmentation list to locate the position this operation need to visit. Then it will finish the operation of read or write following the request.

For a normal read operation, this module needs 5 steps to finish the read task:

- Step A(includes marked item 1 in the Fig. 5.4). The SAP reading get the metadata of the request data(it may be a whole file or some part of a file) from the fragment list.
- Step B(includes marked item 2 in the Fig. 5.4). The SAP reading then creates multi-threads to read the fragments of the requested data into the storage server.
- Step C(includes marked item 2 in the Fig. 5.4). Then the SAP-Reading merges the fragments following the original order to get the requested data reconstructed and ready to reply to the client.
- Step D(includes marked item 3 in the Fig. 5.4). The last step for a read operation is just like the non-fragmented traditional read operation–send the requested data back to the client.

The SAP-writing deals with the write operation similarly with the normal write operation except the extra work that need to do because of the fragmentation of the files.

## 5.2 Experimental Evaluation on System Throughput

### 5.2.1 Evaluation Methodology

To evaluate the performance of our SAP algorithm, we implement a prototype (see Section VI) and analyze the performance of the prototype. We pay attention to two system parameters, namely, data size and the number of fragments for each file. We conducted an experiment to test the file allocation process by varying workload conditions and fragment number in the prototype.

- File size: In our experiments, the test file size varied from 950MB(996147200 bytes) to 2300MB(2411724800 bytes). The details can be found in the corresponding figures(file size unit: byte, throughput: byte/millisecond).
- Fragment number of a file: We varied the fragment number from 3 to 15. All the upper bounds are smaller than 16, because there are 16 storage nodes in our testbed. The prototype performs better when each storage node stores at most one fragment from each file.
- Number of Storage Nodes Effect on Performance: In order to analyze the number of storage nodes effect on performance for different sizes of data, we evaluate our prototype by experiments with scalable data sizes and storage numbers.

### 5.2.2 Experiment Results

Fig. 5.6 shows the impact of file size on system throughput measured in Byte/Sec. From Fig. 5.6 we observe that compared with the traditional scheme, our approach significantly improves the throughput. The performance improvement is attributed to the multi-threading method used to handle multiple fragments of a file in prototype. The results show evidence that multiple threading is an efficient way of implementing the SAP algorithm. A second observation drawn from Fig. 6 is that increasing file size can reduce the system throughput. This trend is observed for the two tested schemes. The results obtained from this experiment indicate that file partitioning and multiple threading can significantly improve the performance of a heterogeneous distributed system.

Fig. 5.7 shows the impact of the fragment number on the throughput. From Fig. 5.7 we can see that the throughput increases when the number of fragments goes up. A large

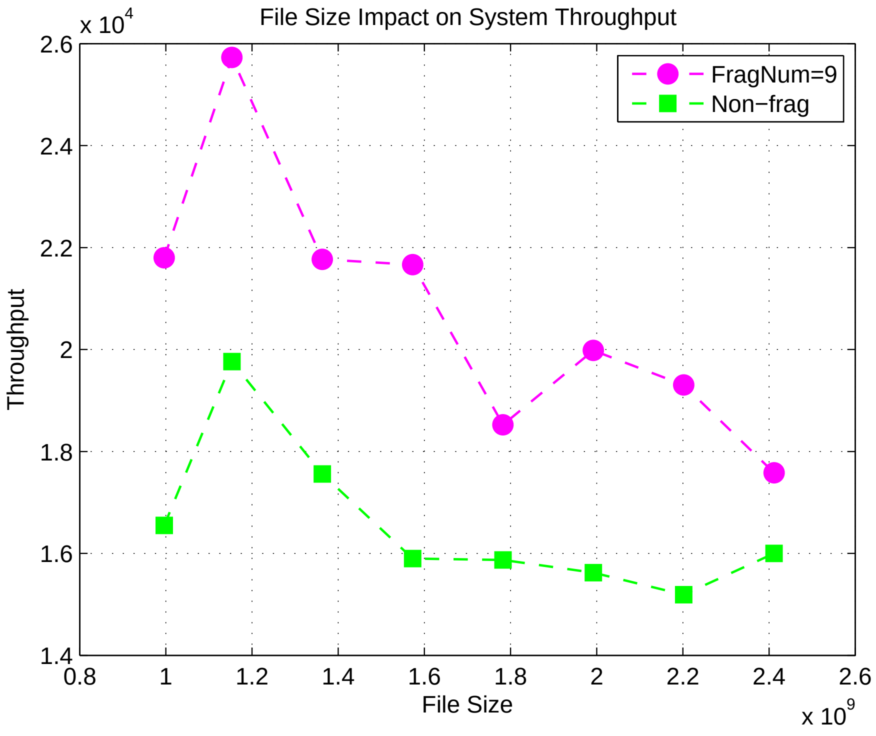

Figure 5.6: *Impact of file size on system throughput.*

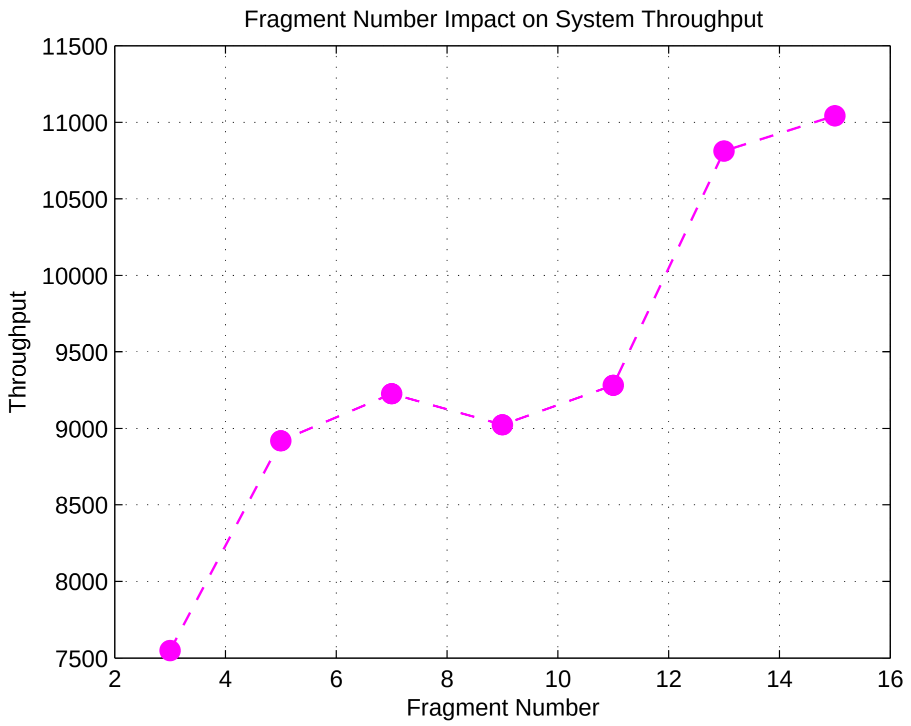

Figure 5.7: *Impact of fragment number on system throughput.*

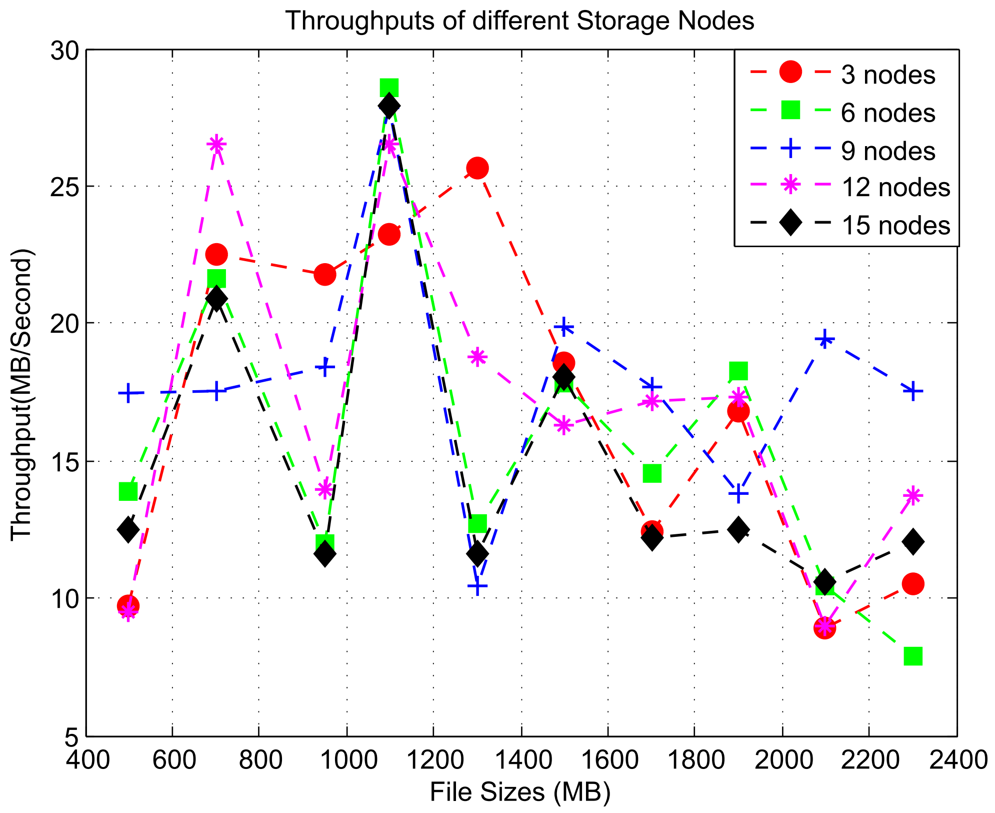

Figure 5.8: File Size Impact on Throughput of Systems Including Different Number of Storage Nodes

number of fragments in a file can improve the read and write performance of the system with respect of the file, because many storage nodes can access fragments in parallel. With multiple threads in place, multiple storage nodes are able to provide a high aggregated bandwidth.

Figs. 5.6 and 5.7 illustrate that file fragmentation and multiple threading are two key techniques to improve performance of the SAP file allocation algorithm designed for heterogeneous distributed systems.

Fig. 5.8 shows that system throughput decreases with increasing of file size. For different sizes of distributed storage systems, the performance decrement coming from the increasing of file size on system throughput are different, indicating that an optimal system size can be determined for certain file sizes.

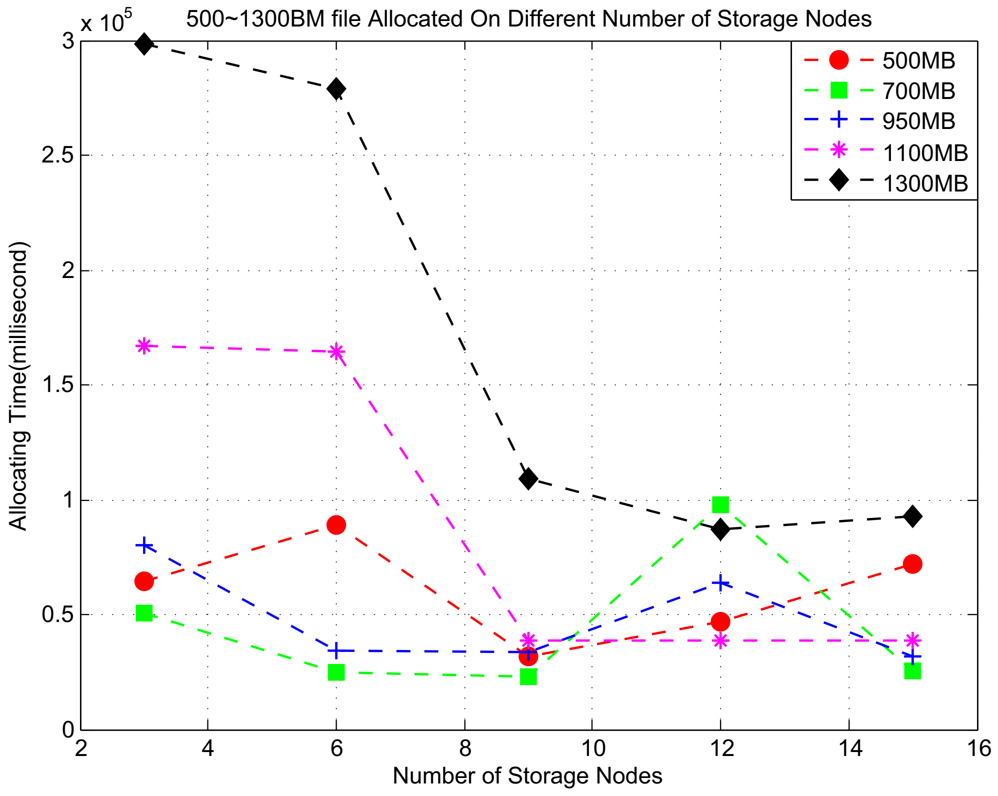

Figure 5.9: File Size and Number of Storage Nodes Impact on System Throughput(1).

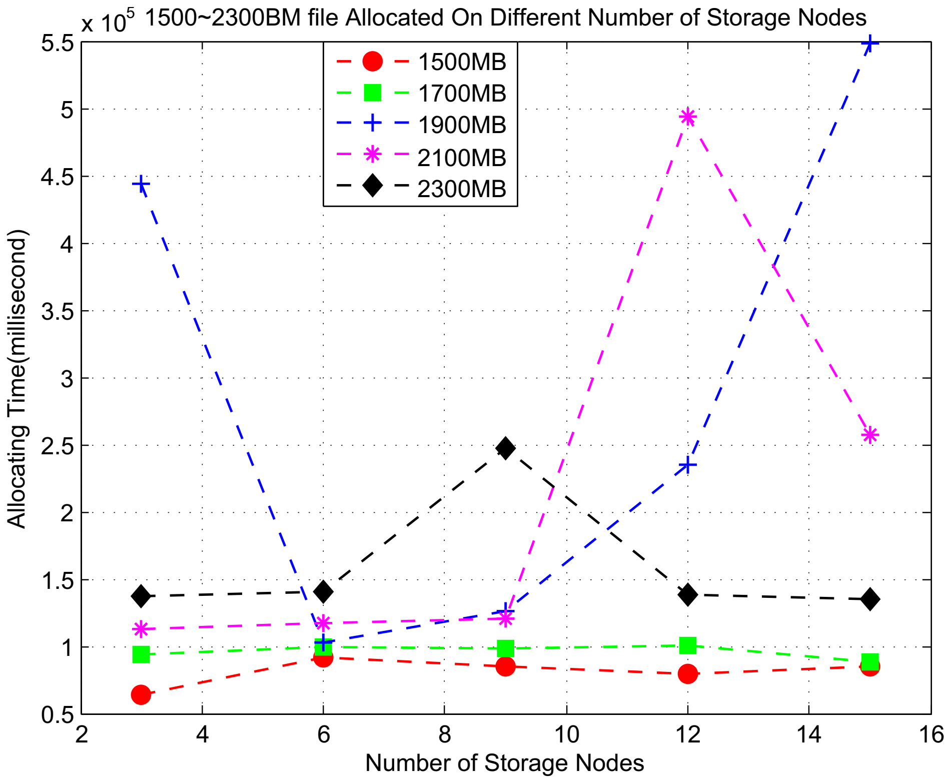

Figure 5.10: File Size and Number of Storage Nodes Impact on System Throughput(2)

Fig. 5.9 and Fig. 5.10 reveal the impact of file size and number of storage nodes on system throughput. We observe that when the data size is small, the system throughput increases with the increasing of system size and the number of storage nodes in the system. When the data size reaches to a certain level, the storage assurance decreases with the increasing value of the system size. One of the potential reasons might be the overhead of data partitioning.

## 5.3 Read and Write Performance

To evaluate system performance of S-FAS and SAP, we replay a few real-world traces, which include parallel web-server, parallel scientific database for remote-sensing data, and data mining.

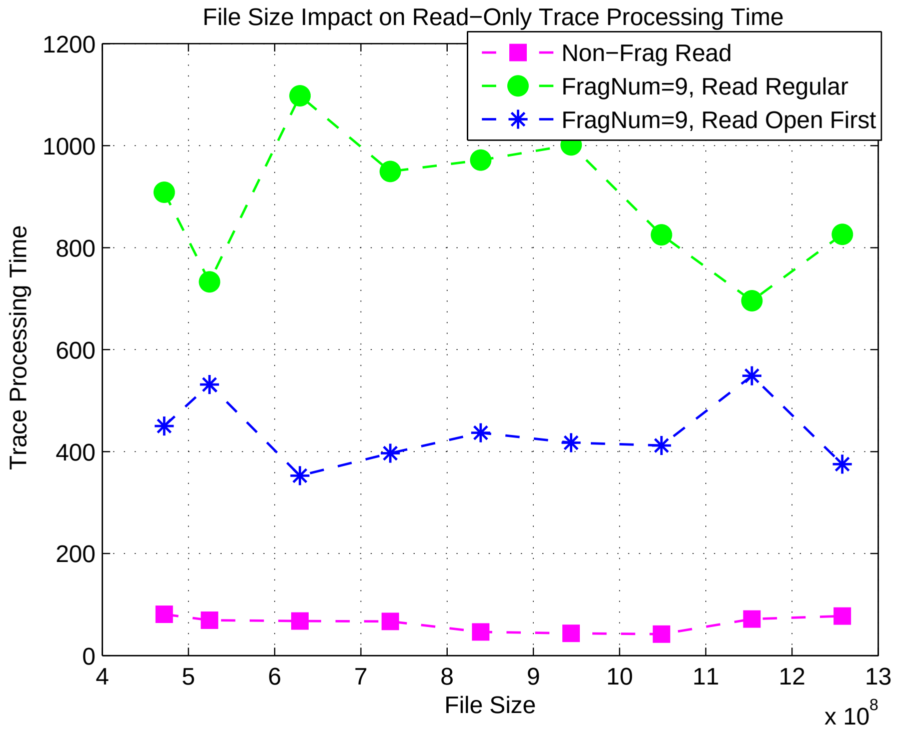

Figure 5.11: *File Size Impact on Read-Only Trace Processing Time.*

We evaluate three scenarios: the read only, write only, and the read-write cases using the Cholesky trace representing I/O intensive applications. We measure the time used for each I/O operation in the trace and assess trace-replay time presented in the y-axis.

The following six tests can be grouped into two sets: the first three tests have different file size, and the last three tests are focused on fragment number.

From Fig. 5.11 shows that the S-FAS scheme spends more time to process the read-only applications, due to the extra time cost introduced by fragment reconstruction. However, SAP significantly improves the performance using the read-open-first strategy. In the read-open-first strategy, regardless of a partial file access or entire file access, all the fragments of the file on the server storage node are accessed to speed up future reads.

For write-only applications, Fig. 5.12 indicates that the S-FAS scheme maintains the same performance as the traditional method. This is because the extra time cost for fragment

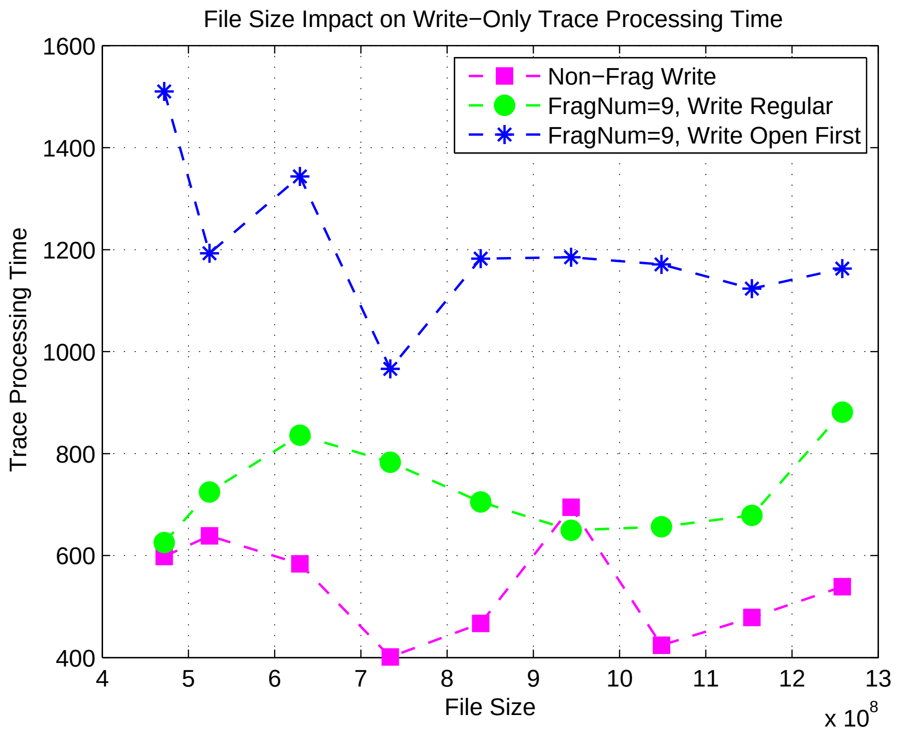

Figure 5.12: *File Size Impact on Write-Only Trace Processing Time.*

is very little when it comes to writes. Thus, S-FAS benefits write-only applications by their security level without significantly degrading performance.

The last three tests are focused on the impact of fragment number on the S-FAS performance. we compare the results between the regular read-write method and the open-first method for the read-only, write-only, and read-write applications.

We vary the fragment number from 3 to 15 while keeping the file size unchanged in the three tests. Fig. 5.14 shows that with the increasing of fragment number, the time spent to process the request for the same file size remains a constant. This is mainly because the requested data size in the trace is small compared with the entire size file. Most of the requested data size is even smaller than a fragment of the files. In most cases to serve a request, there is no need to access multiple file fragments. In our future study, we will investigate a way to improve system performance when multiple file fragments are concurrently accessed.

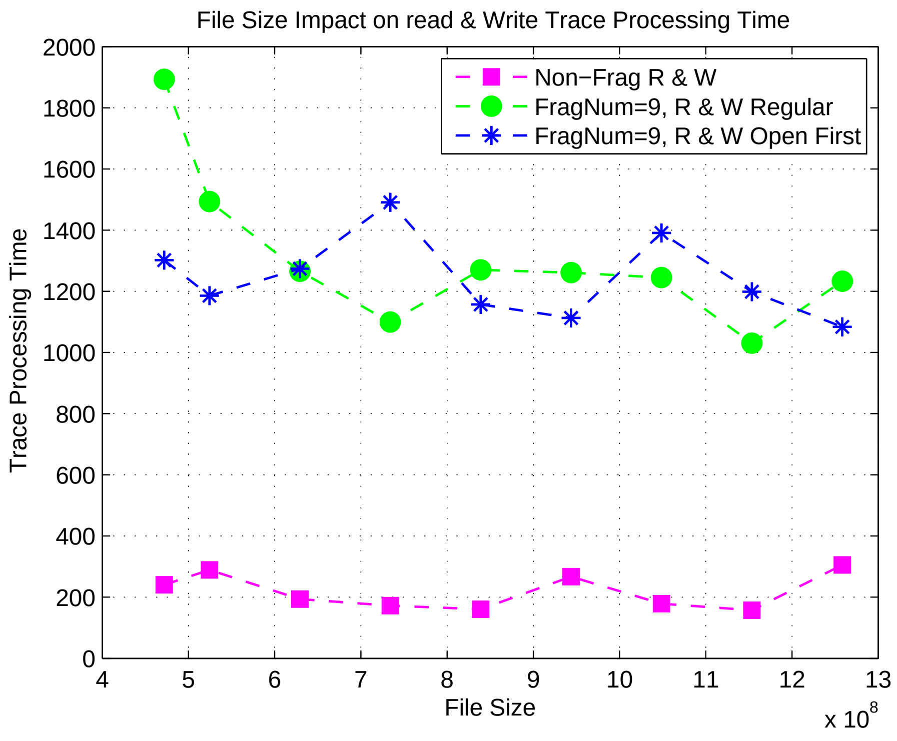

Figure 5.13: *File Size Impact on read and Write Trace Processing Time.*

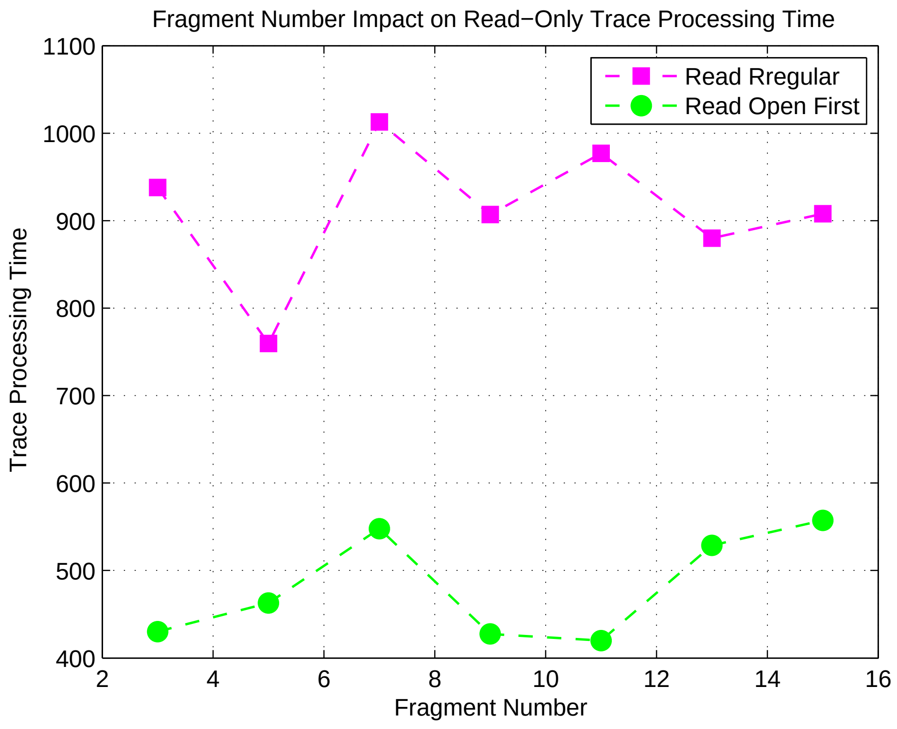

Figure 5.14: *Fragment Number Impact on Read-Only Trace Processing Time.*

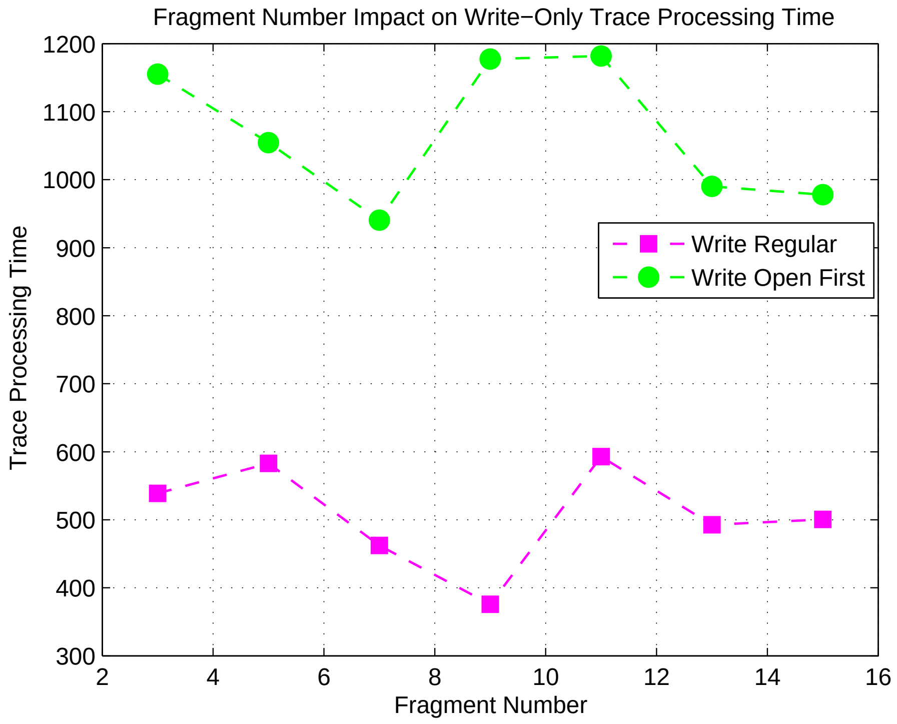

Figure 5.15: *Fragment Number Impact on Write-Only Trace Processing Time.*

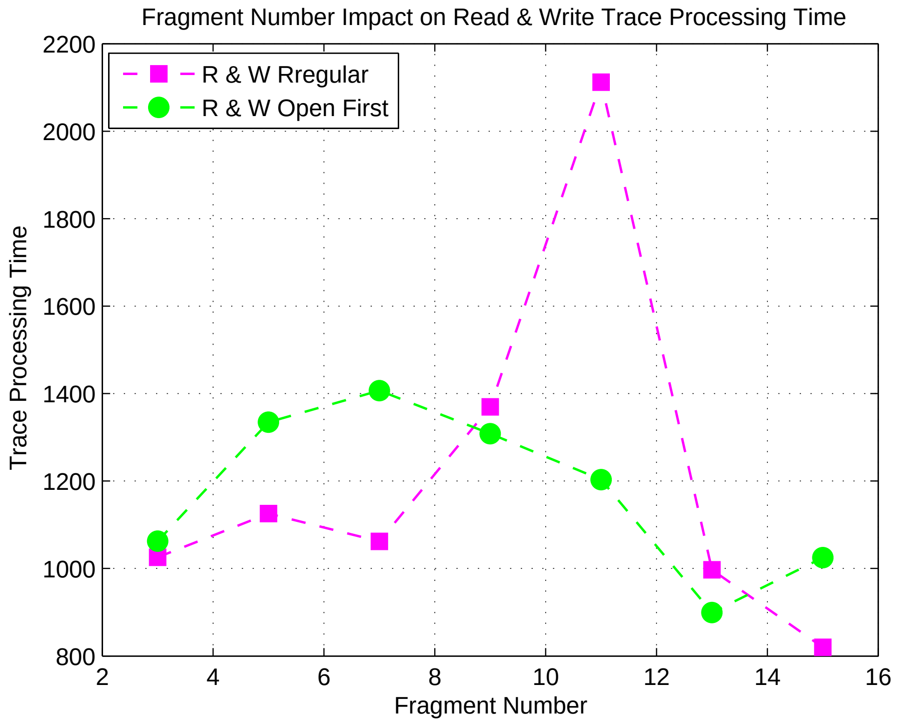

Figure 5.16: *Fragment Number Impact on Read and Write Trace Processing Time.*

## 5.4 Chapter Summary

It is critical to maintain the confidentiality of files stored in a distributed storage system without significantly degrading performance. We developed the SAP file allocation algorithm, the performance of which is evaluated by an assurance model. We also proposed the S-FAS scheme, whose prototype was implemented using C programming language. In addition, we conducted experiments to measure the throughput of the proposed scheme and algorithm. Finally, we replay a few real world traces on our prototype to evaluate system performance.

# Chapter 6

# Reef-A Replication Solution to Improve Availability

In the previous Chapters, we proposed the S-FAS scheme and SAP allocation solution, which can improve distributed storage systems' security and performance. In this Chapter let us illustrate a technique called Reef to improve the availability, security, and performance of distributed systems by integrating fragment replication into our S-FAS and SAP solution.

Recall that S-FAS does not maintain replica fragment. Replication of persistent data is crucial to achieving high availability in storage systems. Reliability can be improved through multiple redundant sites, which makes well-designed cloud computing suitable for business continuity and disaster recovery. Replication degree is important for system performance and resource efficiency. In this chapter we describe a way of integrating replication with S-FAS that addresses heterogeneity issue of distributed systems.

If a distributed system only stores one copy for each fragment, it will be very vulnerable especially in a risky network environment. Redundancy is a common method to improve system availability; however, increasing the number of replicas for each file/fragment makes the file/fragment more likely to be compromised. Evidence shows that heterogeneous features can be well used to improve storage security. The goal of this part of our dissertation is to find a solution to improve both availability and security for distributed systems with heterogeneous features.

The Reef scheme is an extension of the S-FAS scheme. The system model of Reef is similar to that of S-FAS, except that Reef considers system failures caused by hardware diversity when categorizing storage nodes into different groups in the storage node deployment phase. The static storage assurance model described in this chapter aims to provide the analysis of the Reef scheme. Some useful principles to help improve system assurance are draw from the analysis of the Reef scheme. The evaluation and analysis results provide the guidance for the design of a secure allocation process algorithm called R-SAP for a scheme that integrates fragment replication. The design logic of R-SAP is very close to the design logic of SAP except that the R-SAP needs to consider one further step of distributing different replicas of each file's fragments.

The rest of the chapter is organized as follows: We discuss in Section 6.1 the importance of availability for distributed systems and how to make heterogeneity valuable to availability. Section 6.2 describes the system model for the replication scheme and a motivation example for the design of the Reef Scheme. In Section 6.3 we introduce the desired prosperities and design of Reef. In Section 6.4, we propose Reef's static assurance model. Section 6.5 shows the evaluation and analysis results by varying important parameters in the Reef scheme and provides hints on how to apply the proposed Reef scheme efficiently. Then, in Section 6.6 we present a Replication Secure Allocation Process algorithm called R-SAP for Reef. Finally, Section 6.7concludes the chapter and presents our future research directions.

## 6.1 Improve Availability in S-FAS Solution

In this section, we present the importance of availability in distributed systems and the current high availability solutions for distributed systems. Like tradeoff between security and performance (see Chapter 3), tradeoff also must be made between security and availability for high availability techniques for distributed systems. We illustrate how to make use of heterogeneous features in distributed systems to make good tradeoff between security and availability.

### 6.1.1 Availability in Distributed System

Availability is a storage system property that is both highly desired and yet minimally implemented. When some storage nodes in a distributed storage system are down, the entire system might grind to a halt [141] [93]. Downtime is clearly very expensive; for example, in the on-line business world, millions of dollars per hour are lost when systems become unavailable. Replication techniques are commonly employed to enhance the availability of data-intensive services [69]. There are static and dynamic data placement strategies to build data replication services [17] [70] [59] [74].

Replication degree indicates the number of replicas maintained for each file/fragment in a replication mechanism. Files may have different access rates; the locations of their guests may very different [96]. If there are too many replicas existing in a distributed storage system, the storage space will be wasted [94]. The storage servers need extra time to search the corresponding replicas of a file. Thus storage system availability is increased at the cost of degraded system performance. To achieve a better availability and system performance, different files should have different replication degrees. There are also some scenarios exist that most of the files have similar access rates and their users' geographic locations are similar. In such cases, there is no need to vary the replication degree of the stored files.

### 6.1.2 Make Heterogeneity valuable to Availability

In Chapter 3 we already discussed that with the increasing number of the storage node in distributed storage systems, the heterogeneity level tend to increase. Recall that our proposed S-FAS scheme and SAP allocation algorithm make use of the heterogeneous feature to improve security and performance. The risk of a file to be hacked or accessed by unauthorized users is increasing when the replication degree goes up. We address the issue of whether it is possible to improve availability and security by making use of heterogeneous features among storage nodes in distributed systems? Hardware failure is one of the factors that influence availability. Heterogeneous prosperities in hardware lead to various hardware failure rates, which further affect availability. When a replication mechanism is used to improve availability, one has to distribute the replicas to the storage nodes that give rise to high availability, system performance, and security. Intuitively, the availability can be improved if the replicas of a file or fragment are distributed to storage nodes with diverse hardware configurations.

The security risk of different replicas also have impact on availability. Our proposed S-FAS scheme categorize storage nodes into different groups based on their various vulnerabilities. The storage security is improved by distributing the fragments of a file into various storage groups. With one set of successful attack techniques to a certain storage group, a hacker might compromise all the replicas stored on the compromised nodes. The file is still safe as long as the number of unique replica is less than the secret sharing threshold. Ideally the replicas of a fragment should be distributed to the same storage group or few storage groups.

In order to further investigate the above basic idea aiming to improve availability and security for distributed heterogeneous systems, we describe our design, model, evaluation, and an allocation algorithm in the rest of this chapter.

## 6.2 System Model and Architecture

The system model in this chapter is the same as that for S-FAS (see in chapter 3, 3.1.1); the system architecture is the same as that of SAP described in Figure 5.1. As we present in previous section, the availability is improved if the replicas of the same file or fragment are distributed to storage nodes with diverse hardware configurations. In the Reef system model, when we logically categorize storage nodes into different groups depending on the diversity of hardware and vulnerability. Nodes sharing the similar security vulnerabilities are placed into one group; within one group, choose the nodes that have diverse hardware. Thus, to consider security, we firstly apply the same strategy as that in S-FAS and distribute replicas of multiple fragments into different storage groups. Then, we place the replicas of the same fragment into storage nodes within one storage group where nodes are sharing similar security vulnerabilities and have diverse hardware. The availability can be improved, because it is very rare that all of the storage nodes with different hardware fail at the same time.

Before we describe the Reef design, we summarize all notation used in this chapter in Table 6.1.

Table 6.1: Notation used in the system and models.

| Notations | Meaning |
|---|---|
| $N$ | Number of server nodes in the system |
| $U$ | The whole system considered |
| $L$ | Number of subsystems in the whole system |
| $H_i$ | Number of server nodes in the subsystem $i$ |
| $F$ | A file stored in the system |
| $F_i$ | Fragment $i$ of file $F$ |
| $F_{ix}$ | The $x$th replica of fragment $i$ of file $F$ |
| $t$ | The number of replicas for each fragment of file $F$ |
| $t_x$ | The number of replicas for the $x$th fragment of file $F$ |
| $T_j$ | Server type $j$ in the system |
| $K$ | The total number of server types |
| $S_j$ | The size of a certain server type in a cluster |
| $m$ | Threshold for the secret sharing scheme |
| $n$ | The total number of fragments for each file in the secret sharing scheme |
| $R_i$ | Cluster storage subsystem |
| $r_{ij}$ | Node $j$ in subsystem $i$ |
| $X$ | The event that a set of storage nodes is chosen to be attacked |
| $Y$ | The event that if $X$ occurs, at least $m$ different fragments' replicas can be compromised using the same attack method. |
| $CM(Y)$ | The total combinatorial number of $Y$ event. |
| $CM(n*t)$ | The total combinatorial number of different distribution of allocating the $n*t$ replicas into the system which has $N$ nodes. |
| $Z$ | The event of a successful attack to a certain fragment of a file |
| $V$ | The event file $F$ is compromised under one attack method |
| $P_N$ | The successful probability of an attack on a node |
| $P_f$ | The successful probability to compromise a fragment in a compromised node |
| $P(X)$ | The probability of event $X$ occurring |
| $P(Y)$ | The rate of $CM(Y)$ dividing $CM(n*t)$ |
| $P(Z)$ | The probability of event $Z$ occurring |
| $P(V)$ | The probability of event $V$ occurring |
| $\alpha$ | An allocation mapping of file $F$ |
| $SA(\alpha)$ | The storage assurance of an allocation mapping $\alpha$ of file $F$ |

### 6.2.1 Replication Scheme for Heterogeneous Distributed Systems

Replication scheme is commonly employed to enhance the availability of distributed systems. A large number of stories have been carried out to investigate the relationship between availability and a number of factors including replication degree, replica placement, and replication maintenance overhead. In this project we focus on studying the replication degree and replica placement impact. We aim to find a solution to improve availability and security for heterogeneous distributed systems.

In this part of the study we consider two cases, where the replicas of file fragments are duplicated. We provide the analysis of the first case when each file fragment has the same number of replicas.

- Case 1: Each file fragment has the same number $t$ of replicas.
- Case 2: File fragments have various number (i.e., $t_1$, $t_2$, ..., $t_x$) copies of replicas.

The appropriate replica placements are of growing importance to improve availability and efficiency. Appropriate replicas can be used to balance resource consumption for the underlying infrastructure. Both static and dynamic replica placement strategies have been investigated in the past [121] [79] and dynamic placement of replicas.

Many factors (e.g., system components, system topology, application types, clients' distribution and access patterns) have impacts on the efficiency of replica placements [47] [80]. We mainly study the impact on availability and security caused by heterogeneous components of the distributed systems including storage nodes, networks, and application types and access rates.

The heterogeneity of storage nodes includes hardware, software and applied security strategies. Different storage hardware have different lifetimes, potential weaknesses, and capabilities; hence the heterogeneity of hardware has impact on system failures and availability. Considering the security and protection strategies, we discuss a few reasons that may cause different vulnerabilities of the storage nodes. First, storage nodes have different ways to protect data. Second, a security policy can be implemented in a variety of mechanisms. Third, the key length of an encryption scheme may vary across multiple storage nodes. Fourth, heterogeneities exist in computational units of storage sites. We believe that future security mechanisms for distributed systems must be aware of heterogeneous vulnerabilities. The main object for this study is to improve system security and availability in a heterogeneous distributed system, so we pay attention to the heterogeneity factors affecting security and availability.

### 6.2.2 A Motivation Example

Recall that the security of the stored files can be improved by making use of the heterogeneous strategies for security on different storage nodes. In this study we analyze how a fragment replication scheme may increase storage risk by using random or hashing method replica placement in distributed heterogeneous systems. The following motivation example lays out the foundation for us to develop the secure fragment replication scheme.

We consider a heterogeneous distributed storage system (see Fig. 6.1) that contains 25 storage nodes categorized into 5 server-type groups (or server groups for short), i.e., $T_1$, $T_2$, $T_3$, $T_4$, and $T_5$. Storage nodes in each server group offer similar services with the same set of vulnerabilities. In this example, server group $T_1$ consists of nodes $r_1, r_6, r_{11}, r_{16}, r_{21}$, i.e., $T_1 = \{r_1, r_6, r_{11}, r_{16}, r_{21}\}$. Similarly, we define the other four server groups as: $T_2 = \{r_2, r_7, r_{12}, r_{17}, r_{22}\}$, $T_3 = \{r_3, r_8, r_{13}, r_{18}, r_{23}\}$, $T_4 = \{r_4, r_9, r_{14}, r_{19}, r_{24}\}$, and $T_5 = \{r_5, r_{10}, r_{15}, r_{20}, r_{25}\}$.

We assume that a file F is divided into three fragments and each fragment has two copies (i.e., $f_{a1}$, $f_{a2}$, $f_{b1}$, $f_{b2}$, $f_{c1}$, $f_{c2}$). Fig. 6.2 shows the replica placement decisions that do not take heterogeneous vulnerabilities into account. The decision made using a hashing function (see Eq. 11 in [82]) randomly allocates the six fragments of file $F$ to six different nodes, each of which belongs to one of the six server sets illustrated in Fig. 6.2. For example, the six fragment replicas $f_{a1}$, $f_{a2}$, $f_{b1}$, $f_{b2}$, $f_{c1}$ and $f_{c2}$ are stored on nodes $r_2$, $r_7$, $r_4$, $r_9$, $r_{18}$, and $r_{23}$, respectively. Here $r_2$ and $r_7$ belong to storage node group 2; $r_4$ and $r_9$ belong to storage node group 4; and $r_{18}$ and $r_{23}$ belong to storage node group 3. This replica placement happens to be a good decision, because the replicas of a certain fragment are distributed to the storage nodes within the same storage node group. At the same time, different fragments' replicas are distributed to different storage node groups which have various vulnerabilities. A malicious user must launch three successful attacks (one for each server group) in order to compromise all the three fragments.

On the other hand, let us consider a case where the six fragment replicas $f_{a1}$, $f_{a2}$, $f_{b1}$, $f_{b2}$, $f_{c1}$ and $f_{c2}$ are stored on nodes $r_{19}$, $r_{14}$, $r_9$, $r_4$, $r_5$, and $r_{24}$ respectively. Once a malicious user successfully launch server group 4, the replicas $f_{a1}$, $f_{a2}$, $f_{b1}$, $f_{b2}$, and $f_{c2}$ can be compromised and thus the file $F$ is accessible by the hacker. This is because $r_{19}$, $r_{14}$, $r_9$, $r_4$, and $r_{24}$ belong to the same storage node group 4.

Except diverse security vulnerabilities, storage nodes with different hardware configurations have different limitations, lifetimes, and environmental challenge tolerance. Distributing the replicas of a fragment into heterogeneous storage nodes tends to increase availability, because this approach reduces the possibility that the heterogeneous nodes collapse at the same time due to some environmental causes.

Fig. 6.2 shows the possible insecure file fragment allocation decision made by a hashing function(see Eq. 11 in [12]). In this placement decisions, replica $f_{a1}$ could be distributed into Server set 1, replica $f_{a2}$ could be distributed into Server set 2, Replica $f_{b1}$ could be distributed into Server set 3, Replica $f_{b2}$ could be distributed into Server set 4, Replica $f_{c1}$ could be distributed into Server set 5, Replica $f_{c2}$ could be distributed into Server set 6. Server set 1 contains storage nodes $r_1$, $r_6$, $r_{11}$, $r_{16}$, and $r_{21}$; server set 2 consists storage nodes $r_2$, $r_7$, $r_{12}$, $r_{17}$, and $r_{22}$; and server set 3 contains storage nodes $r_3$, $r_8$, $r_{13}$, $r_{18}$, and $r_{23}$; server set 4 is compromised of storage nodes $r_4$, $r_9$, $r_{14}$, $r_{19}$, and $r_{24}$; server set 5 contains storage nodes $r_5$, $r_{10}$, $r_{15}$, $r_{20}$, and $r_{25}$. It may occurred that at least three different replicas of the six fragment replicas $f_{a1}$, $f_{a2}$, $f_{b1}$, $f_{b2}$, $f_{c1}$ and $f_{c2}$ are allocated to storage nodes that belong to the same server-type group. For example, $f_{a1}$, $f_{a2}$, $f_{b1}$, $f_{b2}$, $f_{c1}$ and $f_{c2}$ are stored on nodes $r_{19}$, $r_{14}$, $r_9$, $r_4$, $r_5$, and $r_{24}$ respectively. Nodes $r_{19}$, $r_{14}$, $r_9$, $r_4$, and $r_{24}$ are the member of server group 4. If a malicious user successfully attacks server group 4, replicas $f_{a1}$, $f_{a2}$, $f_{b1}$, $f_{b2}$, and $f_{c2}$ will be compromised, making the file $F$ is reconstructed by the hacker. In

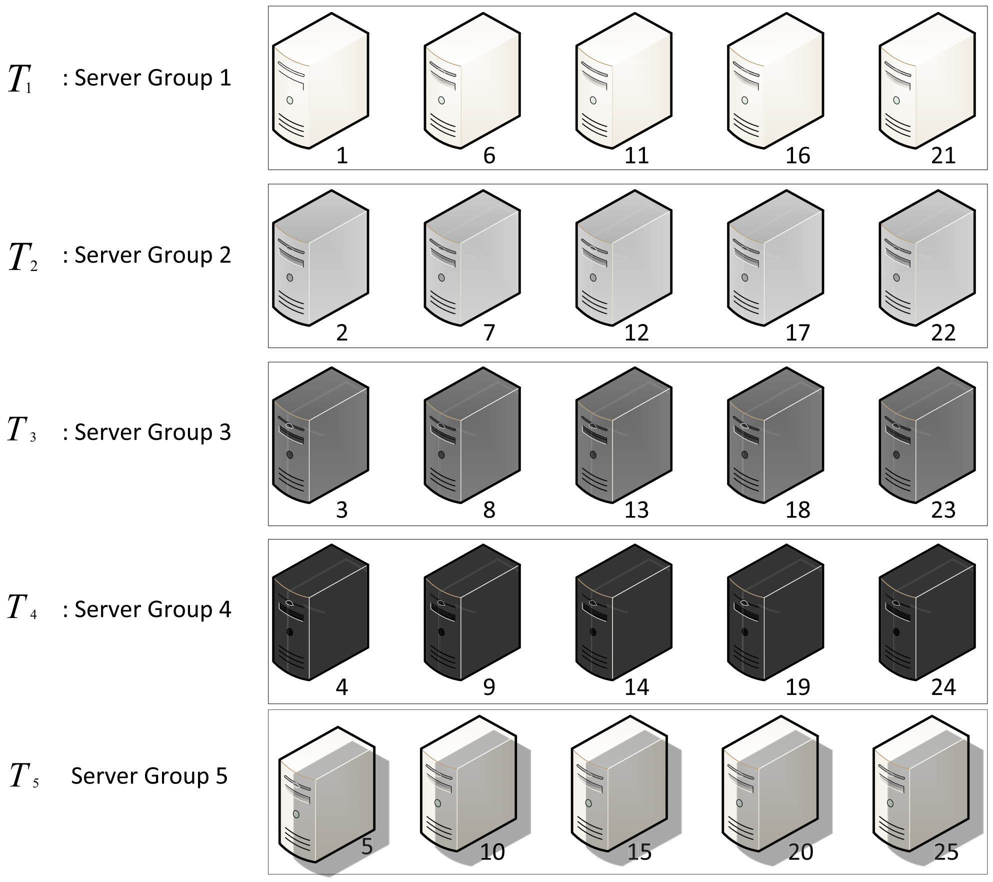

Figure 6.1: *A distributed storage system 2* contains 25 storage nodes, which are categorized into 5 server-type groups (or server groups for short), i.e., $T_1, T_2, T_3, T_4$, and $T_5$. Servers in each group have the same level of security vulnerability.

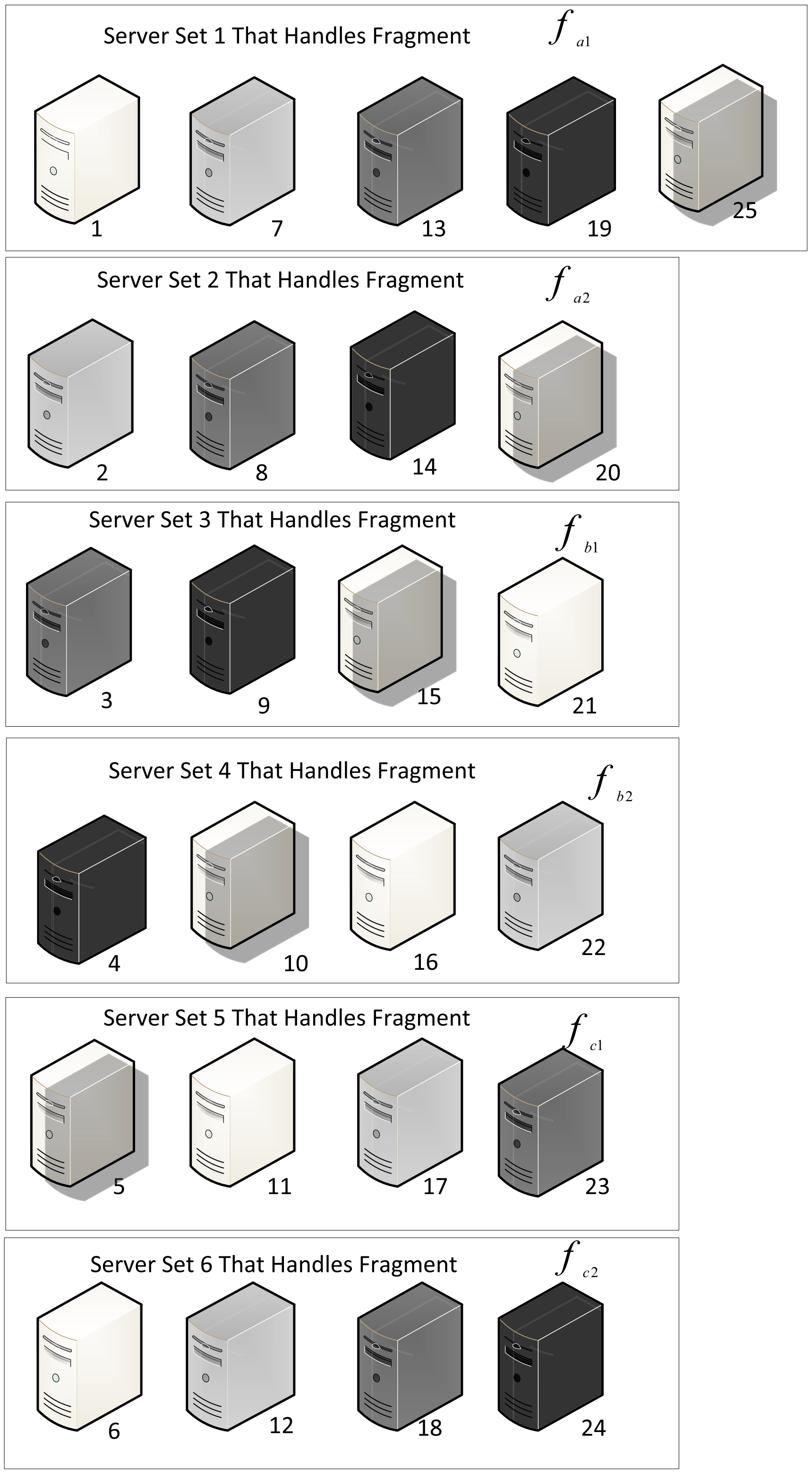
Server Set 1 That Handles Fragment $f_{a1}$
1
7
13
19
25
Server Set 2 That Handles Fragment $f_{a2}$
2
8
14
20
Server Set 3 That Handles Fragment $f_{b1}$
3
9
15
21
Server Set 4 That Handles Fragment $f_{b2}$
4
10
16
22
Server Set 5 That Handles Fragment $f_{c1}$
5
11
17
23
Server Set 6 That Handles Fragment $f_{c2}$
6
12
18
24

Figure 6.2: *Possible insecure file fragment allocation*

such a case, one successful attack against server group $T4$ will break the confidentiality of file $F$.

In this example, the system availability is improved at the cost of system security. Increasing the number of fragment replicas leads to high risk of having the fragment compromised by hackers. Both security and availability are among the top desired prosperities of the distributed system service, so fragment assignment schemes that consider both availability and security are of great demand. In the rest of this chapter we propose a fragment assignment solution that improves both security and availability. Our solution is designed for heterogeneous distributed systems. Our solution is not applicable for the distributed systems where heterogeneity is not a feature.

## 6.3 Reef: A Fragment Replication Scheme for S-FAS

In this section, we first outline the desired properties in our solution including security, availability and performance. Next, we describe the corresponding policies applied in the design of our solution to improve system security, availability, and performance.

### 6.3.1 Design Goal of Reef

The design goal of Reef is to improve security, availability, and performance through file fragment assignments in heterogeneous distributed systems. In Chapter 3 we show that system security can be improved by our S-FAS scheme. Chapter 4 and 5 demonstrate that the S-FAS and SAP solution can improve both system security and performance. The file assignment techniques proposed in the previous chapters largely rely on heterogeneous features of large-scale distributed systems running modern applications on them. In the Reef scheme, we address three desired properties including security, availability, and performance. Based on the above analysis, we summarize the design objectives of Reef below:

- **Security:** Our motivation example indicates that the replication scheme decreases system security. This is because the risk that a file is compromised increases due to the increased number of replicas stored in risky distributed systems. In many circumstances, the risk that the storage nodes are successfully attacked is out of control. Thus, our Reef aims at boosting security even under conditions that some of the storage nodes are compromised.

One way to improve security is to release as little information as possible through compromised storage nodes. In a fragment replication scheme, an ideal file assignment is to store all replicas of a fragment into one group of storage nodes. Different groups of storage nodes take charge of the replica storage for different fragments. Thus, with one set of successful attacks to any storage group, only one fragment of a file is compromised, meaning that hackers are unable to construct the file.

- **Availability:**

  To improve availability of a distributed storage system, we incorporate the fragment replication in Reef. Each fragment has a few duplicated replicas. The replication degree (a.k.a., number of replicas for each fragments) may be different among fragments, because fragments may have different access rates [117] [134].

  To improve availability, the Reef scheme attempts to make the hardware as diverse as possible within a server group. Since the scheme make an effort to distribute the replicas of a fragment to one server group, the replicas of the fragment are assigned to heterogeneous storage nodes in the group. Thus, Reef can increase availability by decreasing the possibility that the homogeneous storage nodes fail at the same time.

- **Performance:**

  A handful of studies focused on the performance improvement of data replication strategies [56] [37] [11] [22]. Our storage node group categorization is a logical classification in a distributed system. The storage nodes within the same group might physically belong to different subsystems. When fragments located in different subsystems are transferred through network, it is risky and takes more time than the case that all needed fragments are stored in one subsystem. Thus considering system performance, firstly all replicas should be distributed to the storage nodes that are close to the clients; secondly the replicas of a file should be distributed to less subsystems to reduce the risk of network transferring and time cost.

### 6.3.2 Design of the Reef Scheme

The design object for Reef is a simple yet efficient approach to distributing replicas of fragments for a file into storage nodes with various vulnerabilities and at same time to keep the file more secure, with higher availability and performance.

The storage node deployment of a distributed system directly and fundamentally influences the system prosperities including performance, security, availability and scalability etc. To improve the assurance of a distributed storage system while maintaining high I/O performance, each cluster storage subsystem has to be built with high heterogeneity in vulnerability. This provides the possibility that the fragments of a file are less likely distributed across multiple storage subclusters.

Based on the study results of our proposed security solution of S-FAS for non-replication scheme: fragmentation is one of the techniques that improve storage security; secret sharing is also an efficient technique to improve storage security; the special fragment distribution strategies is another key method for security improvement for the S-FAS scheme. The difference between non-replication and replication schemes is that there are multiple replicas for every fragment of a file. In addition to the above mentioned security techniques for non-replication scheme, we distribute the multiple replicas of each fragment of a file during the replica distribution phase. We illustrate our security solution for replica distributions in policies 3 to 6 in the Reef scheme.

The strategies to improve performance for non-replication scheme and replication scheme are very similar from the following perspectives: distributing replicas or fragments into the storage nodes that are close to clients; reducing replicas or fragments transferring across different subsystems; and distributing replicas or fragments to storage nodes with high CPU process speed and reliability.

One goal in our design is to reduce the replica or fragment transfer time across different subsystems for the replication scheme. There are different approaches for the non-replication and replication schemes. For our non-replication scheme (see the proposed solution in S-FAF), we attempt to allocate fragments of a file to storage nodes within a subcluster. If the number of nodes with different vulnerabilities cannot meet the aforementioned criterion, some fragments of the file must be allocated across multiple clusters. To illustrate the useful high performance strategy for the replication scheme, We define a file's complete-replica set, which includes at least one replica copy for every fragment of the file. In order to reduce replica transferring, Reef distributes at least one complete replica set of a file to a subsystem that close to clients. Otherwise, at least one fragment replica must be transferred between two subsystems when users issue any read or write requests for the stored files in the distributed systems.

The replication scheme is integrated into a distributed system to improve system availability. Each file is divided to multiple fragments and each fragment has multiple replicas. Disasters, unexpected downtime, and normal system maintenance can make some of storage nodes out of service. Increasing the number of replicas stored in a distributed system can improve the system availability.

The Reef scheme makes storage node deployment for distributed systems and replica placement decisions by following the seven policies in the four categories below:

**Storage Node Deployment** (The policies in this category can be ignored, if Reef is used in exist heterogeneous distributed systems. Reef can be fully used by following some policies below during the storage node deployment phase.).

- Policy 1: All the storage nodes in a distributed storage system are classified into multiple server-type groups (server group for short) based upon their various vulnerabilities. Each server group consists of storage nodes with the same set of vulnerabilities. We place servers with diversified hardware configurations into one server group to make the hardware as diverse as possible to improve the availability while the downtime is caused by hardware.

**Availability**

- Policy 2: The replication scheme is integrated with the fragment assignment module to improve system availability. Every file is divided to multiple fragments and each of which has multiple replicas.

**Security**

- Policy 3(Please ignore this policy if policy 1 is implemented): All the storage nodes in a distributed storage system are classified into multiple server-type groups (server group for short) based upon their various vulnerabilities. Each server group consists of storage nodes with the same set of vulnerabilities.
- Policy 4: To improve security of a distributed storage system, S-FAS allocates fragments of a file to storage nodes, which are members of many different server groups.
- Policy 5: The replicas of a fragment are assigned to the same group of storage nodes. The goal of this policy is to improve the storage assurance of a file by making it less likely happen to compromise enough unique fragments of a file through one set of successful attack techniques for a certain group of storage nodes.
- Policy 6: The $(m, n)$ secret sharing scheme is incorporated into the S-FAS allocation mechanism.

**Performance**

- Policy 7: In order to reduce replica transfer overhead, Reef assigns at least one complete replica set of a file to a subsystem that is close to clients.

If the allocation decisions for the fragment replicas of a file are guided by the above seven policies, successful attacks against less than $m$ server groups have little chance to gain unauthorized accesses of files stored in a distributed system. In other words, if the number of compromised unique replicas of a file's fragments is less than $m$, attackers are unable to reconstruct the file from the replicas fragments accessible to hackers. The Reef scheme can improve information assurance of files stored in a distributed storage system without enhancing confidentiality services deployed in distributed storage systems, because Reef is orthogonal to security mechanisms offering confidentiality for each server group in distributed storage systems. Thus, Reef can be seamlessly integrated with any confidentiality service employed in distributed storage systems in order to offer enhanced security services. At the same time, the integrated fragment replication scheme, the corresponding deployment of storage nodes, and the distribution of replicas help to improve availability, fault tolerance, performance, as well as scalability .

## 6.4 Static Assurance Model for Reef

In this section, we developed a static assurance model to quantitatively evaluate the security of a heterogeneous distributed storage system, in which Reef handles fragment allocations. To simplify the model, we develop the static storage assurance model for the case where all fragments share the same number $t$ of replicas.

The modeling approaches to building the static assurance models for Reef and S-FAS are the same. The models include multiplication principle, probability theory, and combinatorics. We analyze the steps in the entire process of an successful attack to a distributed system. Here we use the multiplication principle to describe the attacking process. To describe the success rate of each step within the attacking process, we apply the probability theory and combinatorics to evaluate the process.

As we describe in Table 6.1, most of the definitions we used in Reef are exactly the same as the definitions in S-FAS (Please refer to Chapter 3 for details). Nevertheless, the definition of $Y$ and $P(Y)$ in Reef is different from those used in S-FAS. In the Reef model, $Y$ means the event that if $X$ occurs, at least $m$ different fragments' replicas(fragments in S-FAS ) can be compromised using the same attack method. Specifically in the storage assurance model for Reef, we define $P(Y)$ as the ratio of the combinatorial number of $Y$ event to the combinatorial number of distributing the $n * t$ replicas to the $N$ nodes in a distributed system.

Since attackers might apply a similar approach to compromise both non-replica-based and replica-based distributed storage system, the calculation of $P(V)$ in the Reef model is the same as that in S-FAS in Chapter 3 (Please refer to Eq. 3.2 and Eq. 3.3). The definitions and calculations, of $P(X)$ and $P(Z)$ in the static assurance model for Reef are also the same as those presented in S-FAS in Chapter 3.

The main difference between the S-FAS model and the Reef model is that each file fragment has multiple replicas in Reef. The placement of multiple replicas for all the fragments of a file is a more complicated than the placement of all the fragments of a file. Recall that $Y$ is the event that if $X$ occurs, at least $m$ different fragments' replicas can be compromised using the same attack method. The number of compromised replicas ranges from $m$ to $n * t$(in S-FAS, it ranges from $m$ to $n$), because every fragment has $t$ replica copies. The compromised fragments may have different (e.g., 1, ..., $t$) copies of replicas that are stored in the compromised nodes. In the following part, we derive probability $P(Y)$.

To calculate the combinatorial number of event $Y$, we consider a special case $W$, where if one picks $y$ replicas from the $x$ fragments' replicas, the $y$ pieces are the replicas of the $x$ different fragments.

We denote $T(x, y)$ as the combinatorial number of case $W$. In the chosen $j$th storage server group that includes $S_j$ storage nodes, if event $Y$ occurs, the number of nodes storing replicas of file $F$ may range from $m$ to $S_j$ ($m <= S_j$), because there might be a portion of nodes in the $jth$ group storing replicas of file $F$.

Now we can draw the combinatorial number ($CM(Y)$) of $Y$ from the above expression as below:

$$CM(Y) = \sum_{y=m}^{S_j} [T(m, y) + T(m+1, y) + T(m+2, y) + ... + T(n, y)] \tag{6.1}$$

When $t >= y$, we have the following further reasoning and expression of $T(x, y)$ (we do not give the general formula when $t < y$ and it is not that hard to calculate $T(x, y)$ if we have the concrete values for all the related parameters).

$$\begin{aligned}
&T(1, y) = C_t^y \\
&T(2, y) = C_{2t}^y - C_2^1 * T(1, y) = C_2^2 * C_{2t}^y - C_2^1 * C_t^y \\
&\ T(3, y) = C_{3t}^y - C_3^2 * T(2, y) - C_3^1 * T_1^y \\
&\ = C_3^3 * C_{3t}^y - C_3^2 * C_{2t}^y + C_3^1 * C_t^y \\
&... \\
&T(x, y) = C_x^x * C_{xt}^y - C_x^{x-1} * C_{(x-1)t}^y + ... \\
&\ = \sum_{i=0}^{x-1} (-1)^i * C_x^{x-i} * C_{(x-i)t}^y
\end{aligned} \tag{6.2}$$

The combinatorial number $CM(n * t)$ of distributing the $n * t$ replicas of file $F$ to the $N$ nodes in a distributed system can be expressed as:

$$CM(n * t) = C_N^t * C_{N-t}^t * C_{N-2t}^t * ... * C_{N-(n-1)t}^t \tag{6.3}$$

Based on the definition of $P(Y)$–the rate of the combinatorial number of event $Y$ dividing the combinatorial number of distributing the $n * t$ replicas to the $N$ nodes, we can derive $P(Y)$ from both $CM(Y)$ and $CM(n * t)$ as:

$$P(Y) = \frac{CM(Y)}{CM(n*t)} = \frac{\sum_{y=m}^{S_j} [T(m,y) + T(m+1,y) + T(m+2,y) + ... + T(n,y)]}{C_N^t * C_{N-t}^t * C_{N-2t}^t * C_{N-3t}^t * ... * C_{N-(n-1)t}^t} \tag{6.4}$$

Recall that $P(V)$ is the probability that file $F$ is compromised under a successful attack. Now we can derive $P(V)$ from Eq. 6.4 as below:

$$P(V) = \sum_{j=1}^{K} \{\frac{S_j}{N} * P(Z) * \frac{\sum_{y=m}^{S_j} [T(m,y) + T(m+1,y) + ... + T(n,y)]}{C_N^t * C_{N-t}^t * C_{N-2t}^t * ... * C_{N-(n-1)t}^t}\} \tag{6.5}$$

The confidentiality of file $F$ is assured if file $F$ is unable to be reconstructed by an attacker. Given a fragment assignment decision $\alpha$, we can derive the storage assurance $SA(\alpha)$ for file $F$ stored in a distributed storage system as:

$$SA(\alpha) = 1 - P(V) = 1 - \sum_{j=1}^{K} \{\frac{S_j}{N} * P(Z) * \frac{\sum_{y=m}^{S_j} [T(m,y) + T(m+1,y) + ... + T(n,y)]}{C_N^t * C_{N-t}^t * C_{N-2t}^t * ... * C_{N-(n-1)t}^t}\} \tag{6.6}$$

## 6.5 Evaluation of System Assurance

In addition to the influential factors (i.e., $K$, $N$, $n$ and $m$) in the S-FAS scheme, replication degree $t$ affects the assurance model for the Reef scheme described in Section 6.4. The rest of this sub-section presents our quantitative evaluation on the impacts of these factors on the information assurance of distributed storage systems.

### 6.5.1 Impact of Replication Degree on Storage Assurance

The first affecting factor we considered is the replication degree $t$ of the scheme. Obviously, increasing the number of replicas improves the availability of a distributed system. With more replicas stored across multiple nodes in the system, the system provides good

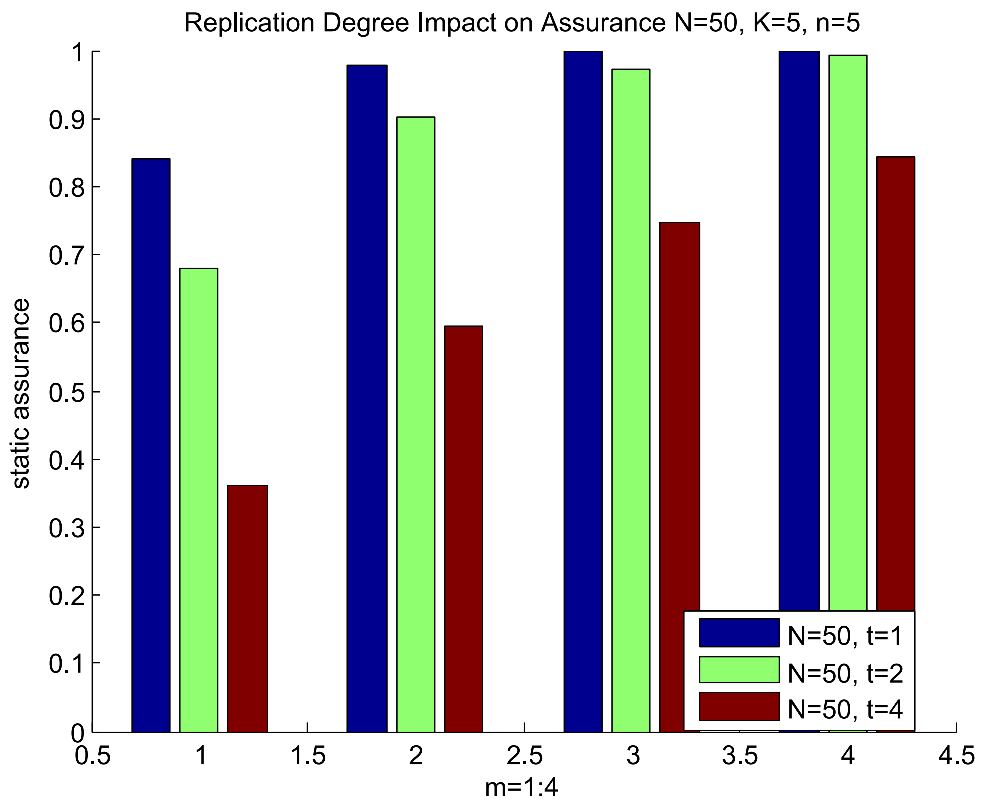

Figure 6.3: *Replication Degree Impact on Assurance*

I/O throughput. Fig. 6.3 shows the impacts of replication degree on the static assurance of a distributed storage system.

In this experiment, the secret-sharing threshold is increased from 1 to 4. We evaluate the static assurance of a 50-node distributed system, where there are five difference server types. The number of replicas is set to 1, 2, and 4 respectively. Fig. 6.3 confirms that with the increasing value of the replication degree, the assurance of the distributed system is decreased. This observation suggests that we should consider limiting the boundary of replication degree to control the storage assurance at an acceptable level.

### 6.5.2 Impact of System Size on Storage Assurance

System size or the number of storage nodes ($N$) in a distributed system is one of the very important parameters for the system. We further study the system size impact on storage assurance. In the second experiment, we keep the number of replicas to 2. The value of $N$ is set to 30, 40, and 50. The other parameters are kept the same as those in the first experiment. Fig. 6.4 presents the preliminary results of the impact of system size in the Reef replication scheme.

Fig. 6.4 shows that the storage assurance increases with the increasing of system size. This result indicates that for our proposed Reef scheme, even the diversity level represented by the $K$ is kept unchanged, the storage assurance can be enhanced by increasing the storage nodes within the set of same storage types.

This security trend of the duplicated scheme is significantly different from that of the non-replicas-based scheme in S-FAS, where increasing the system size has no impact on the storage assurance if the diversity level ( $K$ ) is a constant. This observation demonstrates that Reef is very useful for large-scale distributed systems, where replicas are deployed to improve reliability. We can increase the storage assurance by adding extra nodes to a distributed system even if the diversity level of the system does not increase.

Interestingly, when the secret-sharing threshold is set to 4, the static assurance of the distributed system becomes less sensitive to the system size. We conclude that when the secret-sharing threshold is very low, static assurance can be improved by increasing the number of nodes in the system.

### 6.5.3 Impact of Number $n$ of File Fragments on Storage Assurance

The number of fragments for each file in our integrated secret sharing scheme is another very important parameters affecting storage assurance. Now we study the impact of the fragment number $n$ on storage assurance.

In the design of the experiment, the number $n$ of fragments is increased from 5 to 7. The parameters $k$ and $N$ are set to 5 and 50, respectively. We also vary threshold $m$ from 1 to 4.

Fig. 6.5 shows that the system assurance is reduced with the increasing value of fragment number $n$. The trend of the Reef scheme is the same as the trend of S-FAS. This is because

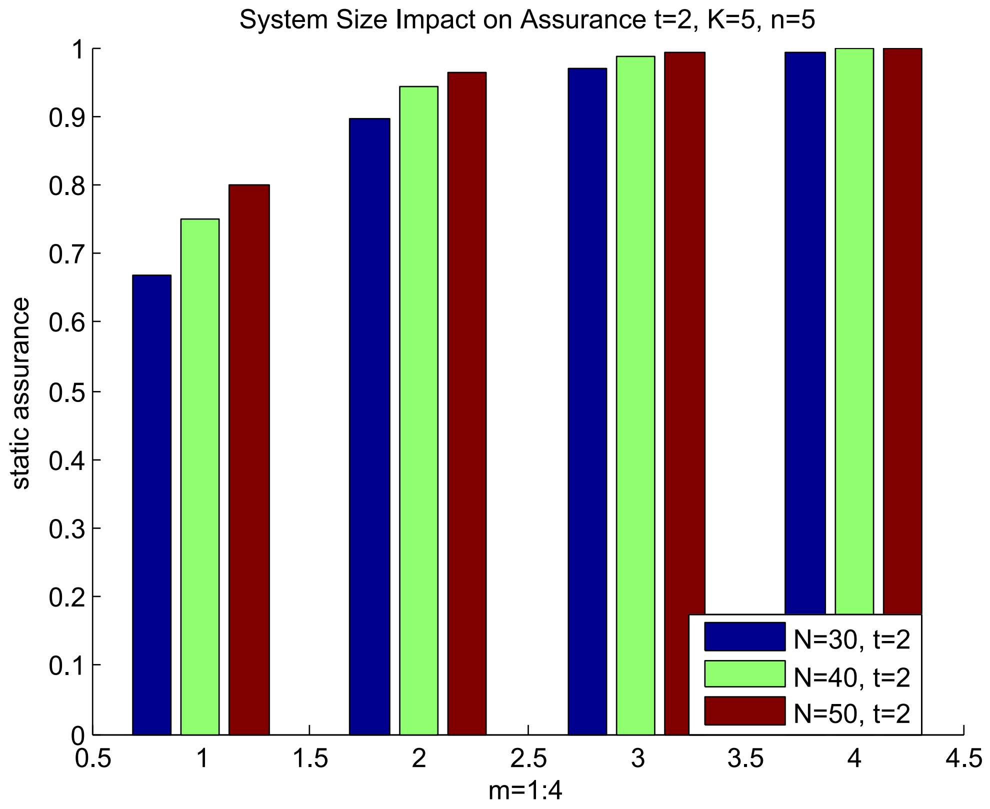

Figure 6.4: *System Size Impact on Assurance*

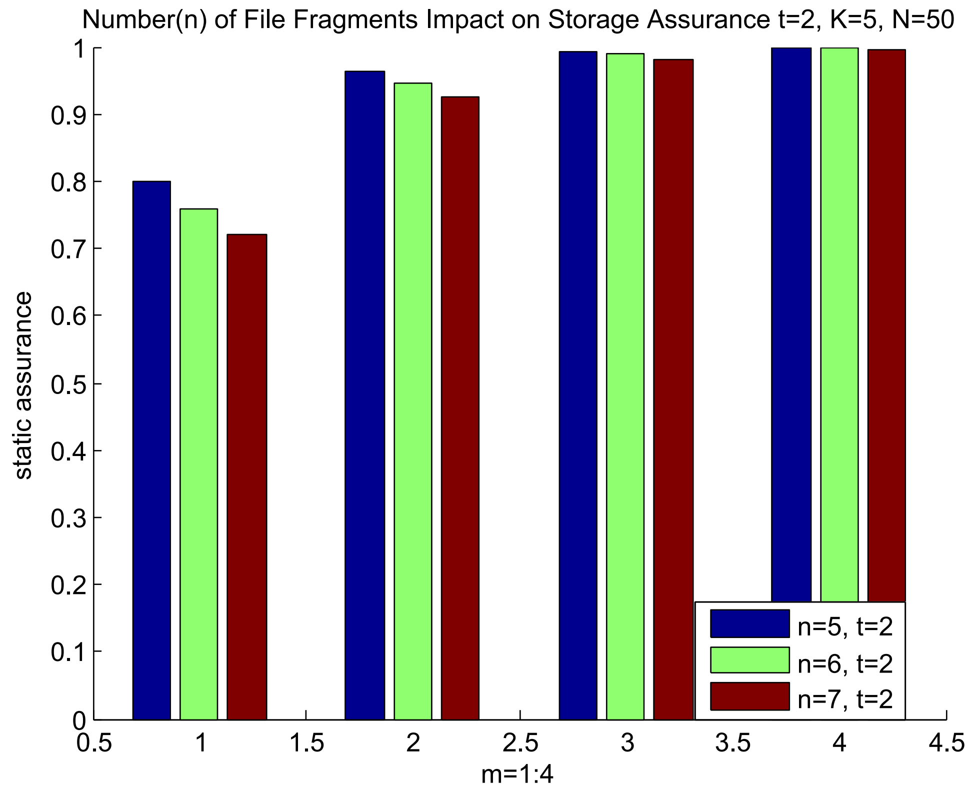

Figure 6.5: *Number of Fragments Impact on Assurance*

with the increasing of fragment number $n$, the possibility that more fragments of a file are allocated to storage nodes of the same group is increased. Compared to S-FAS, Reef is more sensitive to the number $n$ of file fragment. The reason is that each replica of a fragment in Reef has the same level of influence on storage assurance as a single fragment in the S-FAS scheme.

Thus, we can reach a conclusion that increasing the number of fragments for files stored in a distributed system reduces the static assurance of the system.

## 6.6 R-SAP: The Allocation Algorithm for Reef

In this section, we present a replication allocation process algorithm called R-SAP designed for the Reef scheme. We first describe the design goals related to system security, availability, and performance. Next, we propose the R-SAP algorithm.

### 6.6.1 Factors Affecting Security, Availability and Performance

Three design goals of a distributed storage systems are system security, availability and performance. System parameters that have significant performance impacts include: storage nodes and their hardware, software, process speed, storage capability, security vulnerability and connected network, intranet and internet connection quality. In addition to system parameters, other affecting factors are workload and data stored in distributed systems and its data features such as data size, access rate, fragment number, replication degree and the locations of the replicas. Table 6.2 lists all notations used throughout this section in the R-SAP algorithm.

All the related definitions (such as $w_N$, please refer to Eq. 4.1), analysis of the impacts on performance from the above factors in R-SAP are the same as those in SAP(Please refer to Section 4.2 in Chapter 4 for the details).

Since Reef incorporates replicas, we extend SAP's workload model to capture the workload conditions in the Reef case. We use $Workload$(see Eq. 6.7 below) to evaluate the workload that a replica of fragment impose to the storage node serving the fragment.

$$Workload_{ihx} = \lambda_{ihx} * DataSize_{ihx} \tag{6.7}$$

To address the load balancing issue, the R-SAP algorithm evenly distributes the load expressed by Eq. 6.8 across to multiple nodes $(1, ..., N)$.

$$Workload\_N = w_N * \sum_{i=1}^{d} \sum_{h=1}^{|F_i|} \sum_{x=1}^{t} \lambda_{ihx} * DataSize_{ihx} \tag{6.8}$$

### 6.6.2 The Design of R-SAP

In this section we describe the design of R-SAP that is integrated with the Reef scheme. We focus on the static algorithm rather than its dynamic counterpart, because we plan to extend R-SAP to address the dynamic data placement issue in our future study.

Table 6.2: Notation used in the SAP Algorithm.

| Notations | Meaning |
|---|---|
| $S$ | the size of file F |
| $u$ | server type in the system |
| $S_i$ | the size of server type $i$ in a cluster |
| $\|F\|$ | the total number of files |
| $\|F_i\|$ | the total number of fragments in file $\|F_i\|$ |
| $F_{ihx}$ | the $x$th replica of the $h$th fragment of the $i$th file |
| $Workload_{ihx}$ | the work load that the $x$ replica of the $h$th fragment of the $i$th file brings to the node where it is stored |
| $DataSize_{ihx}$ | the $x$th replica of the $j$th fragment size of the $i$th file |
| $b_N$ | Bandwidth of the connected network of the Nth node |
| $\lambda_{ihx}$ | the access rate of the $x$th replica of $j$th fragment of $i$th file |
| $N_{uv}$ | the v th node of server type u u,v= 0,1... |
| $I_{ihx}$ | Decreasing sorted list of fragments to be allocated (Fragments of the same file are consecutive and belong to the same row) |
| $TLoad\,(N_{uv})$ | total available load of node $N_{uv}$ |
| $CLoad\,(N_{uv})$ | current load of node $N_{uv}$ |
| $RLoad\,(N_{uv})$ | N number of element vector recording the total available storage capacity for all nodes |
| CurrentN[u] | the current available node in the $u$th type of servers |
| LoadBN$_{uv}$ | the most load that shoud be assigned to node N$_{uv}$ |

R-SAP consists of two steps: sorting the replicas in a decreasing order in a three-dimensional list $I_{ijx}$ based on the workload of the replicas(Fig. 6.6)); and dispatching all the replicas in the list $I_{ijx}$ to distributed nodes ( Fig. 6.7).

The detailed steps of R-SAP are outlined in Algorithm 2.

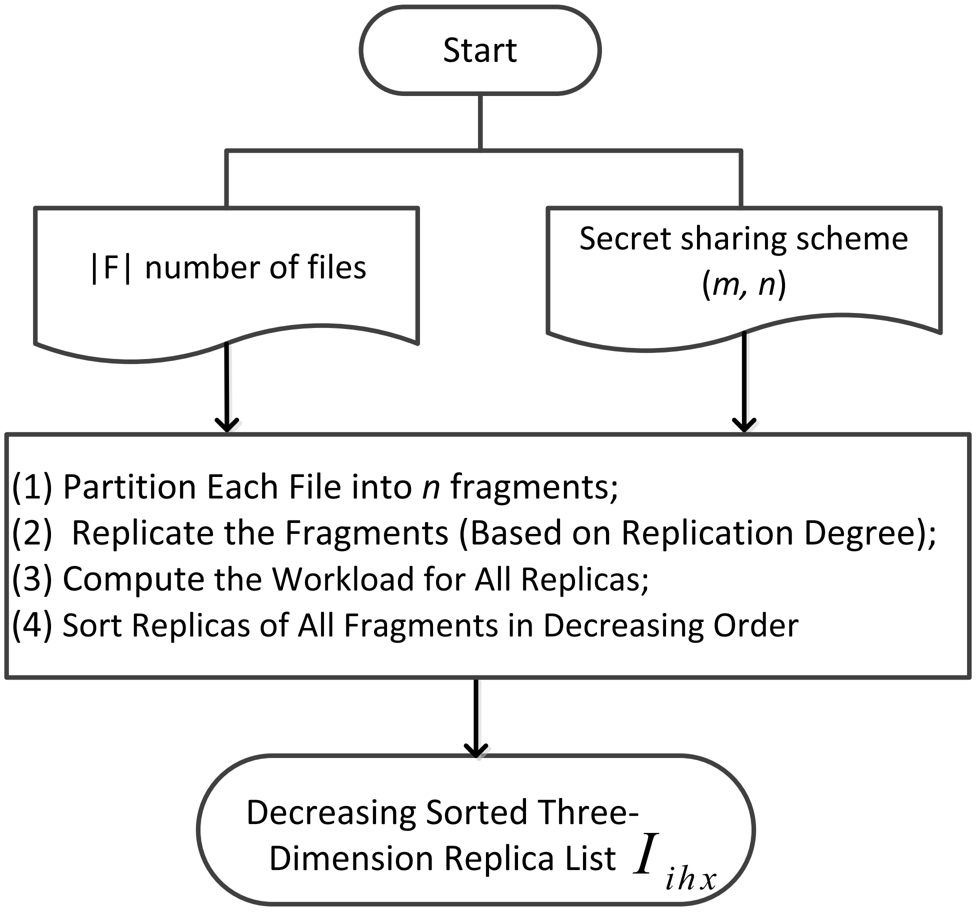
Start
|F| number of files
Secret sharing scheme (*m, n*)
(1) Partition Each File into *n* fragments;
(2) Replicate the Fragments (Based on Replication Degree);
(3) Compute the Workload for All Replicas;
(4) Sort Replicas of All Fragments in Decreasing Order
Decreasing Sorted Three-Dimension Replica List $I_{ihx}$

Figure 6.6: *Creating a Three-dimension Decreasing Ordered Replica List*

**Algorithm 2** Replication Secure Allocation Process(R-SAP):

```
Input:
$I_{ihx}$
Storage Nodes in a Two-dimension List
for $i = 1$ and $i < |F|$ do
  //Allocate all File $i$
  for $h = 1$ and $h < |F_i|$ do
    //Allocate all Fragment $h$ of File $i$;
    //Try to Distribute Fragment into Diverse Server Groups.
    for $x = 1$ and $x < t$ do
      //Allocate all Replica $x$ of Fragment $h$ for File $i$;
      //Try to Distribute Replicas of a Fragment into the Same or Less Server Groups.
    end for
  end for
end for
Output:
The Updated Replica List for Multi-thread Replica Writing
```

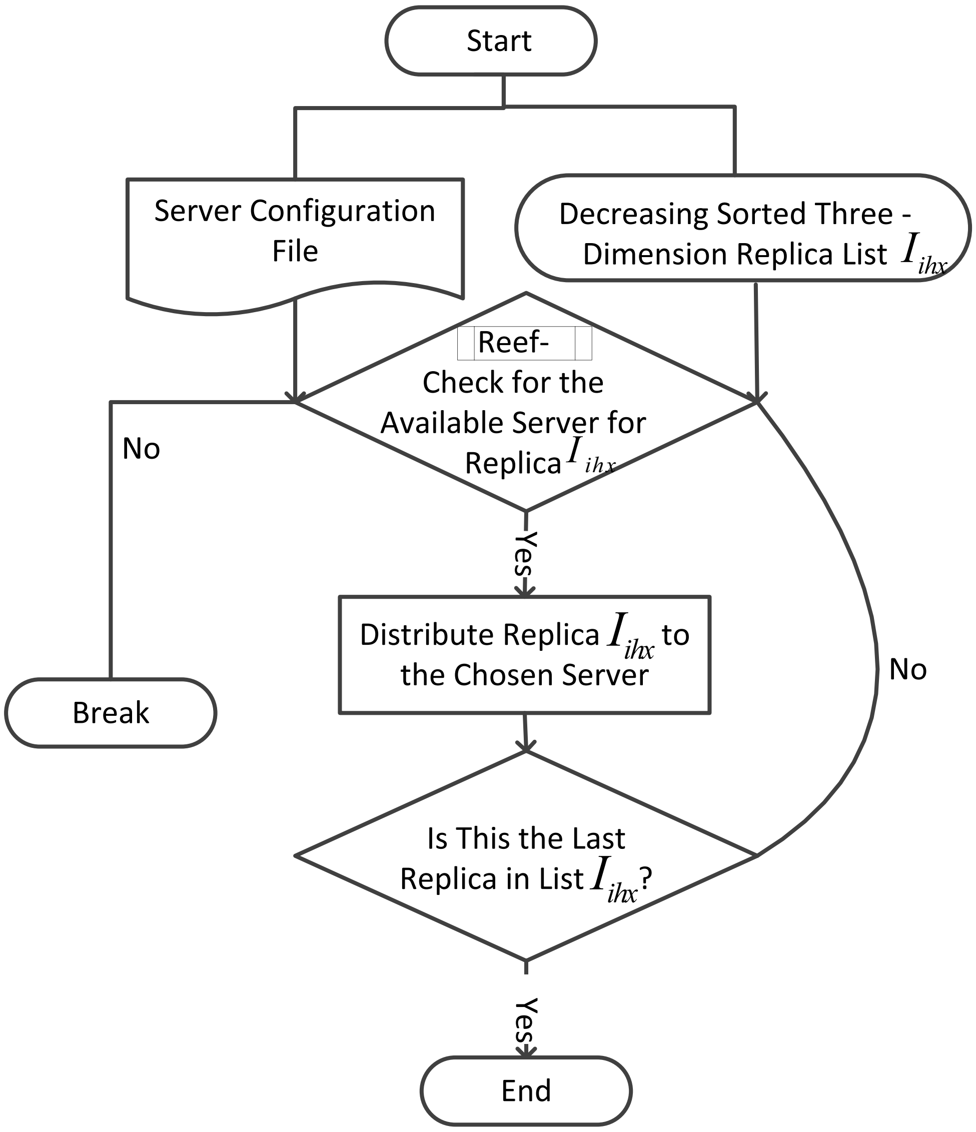
Start
Server Configuration File
Decreasing Sorted Three - Dimension Replica List $I_{ihx}$
Reef-
Check for the Available Server for Replica $I_{ihx}$
No
Yes
Distribute Replica $I_{ihx}$ to the Chosen Server
No
Break
Is This the Last Replica in List $I_{ihx}$?
Yes
End

Figure 6.7: *Data Flow of the R-SAP Algorithm*

## 6.7 Chapter Summary

It is critical to maintain a high availability of files stored in a distributed storage system, even when some storage nodes in the system are out of service.

In this Chapter we proposed a solution called Reef to improve the distributed storage system security and availability by integrating the fragment replication technique, secret sharing, fragment replication. Reef considers heterogeneity features of distributed storage systems during the replica placement phase. The Reef scheme is an extension of the S-FAS scheme. The system model for Reef is similar to that of S-FAS except that Reef address the system failures mode and aims to improve system reliability in addition to security. We build a static assurance model to quantitatively evaluate the system assurance for the Reef scheme. We also developed a replica allocation process algorithm called R-SAP to demonstrate how does the proposed Reef scheme work. In the Reef design, we addressed the distributed system security, performance, as well as availability. To evaluate the assurance provided by Reef, we studied the impacts caused by replication degree, system size, and the number of fragments.

In this dissertation study, we only focus on the static replica placement solution. Dynamic replica reallocation schemes are essential to achieve high performance and availability of distributed systems, especially for internet based applications and services. In a dynamic wide-area environment, client access patterns, network conditions, and service characteristics are constantly changing. We plan to study and propose a dynamic replica reallocation approach for heterogeneous distributed system by extending Reef to address the heterogeneous vulnerabilities in the large scale distributed systems.

# Chapter 7

# Future Research Plan

In the previous research, we addressed the heterogeneous vulnerability issue by dividing storage nodes of a distributed system into different server-type groups based on their vulnerabilities. Each server-type group - representing a level of vulnerability - contains storage nodes with the same security vulnerability. We proposed a secure fragmentation allocation scheme called S-FAS to improve security of a distributed system where storage nodes have a wide variety of vulnerabilities.

To quantify information assurance and to evaluate the proposed S-FAS scheme we developed storage assurance and dynamic assurance models. We discovered some principles to improve assurance levels of heterogeneous distributed storage systems. The principles are general guidelines to help designers achieve a secure fragment allocation solution for distributed systems.

In order to consider the system performance for S-FAS applied system, we developed a secure allocating processing (SAP) algorithm to improve security and system performance by considering the heterogeneous feature of a large distributed system.

In order to conduct the performance analysis for the S-FAS scheme and SAP allocating algorithm, we developed a prototype for our S-FAS scheme. We implemented the prototype and conducted some experiments on the throughput of the proposed scheme and algorithm.

To further study the influential factors on different desired metrics in distributed systems using the S-FAS scheme and SAP allocation algorithm, we have done experiments and analysis against different popular and important real world traces. All the experiment and analyzing results have provided guidance to upgrade the current S-FAS scheme with higher performance and availability using fragment replication and energy efficiency strategies for different kinds of applications in our future work. Based on the work we have done, we split our future research work into the following tasks.

## 7.1 Task1: Dynamic Replica Reallocation in Heterogeneous Distributed Systems Based on Reef Scheme

Currently the design of Reef is static replica placement solution. Dynamic replica reallocation is essential for high performance and availability, especially for internet based applications and services, because of constantly changing client access patterns, network conditions, and service characteristics in the wide-area environment. There is already some research work done on replication and dynamic reallocation without considering the heterogeneity of the nodes in distributed systems. Heterogeneous systems have their own characteristics such as non-uniform/correlated machine failures, so that schemes designed for distributed systems with high diversity should be different from the homogeneous systems. I plan to study and propose a dynamic replica reallocation protocol for heterogeneous distributed system based on S-FAS.

## 7.2 Task2: Considering Energy as a System Resource to Design Optimal Dynamic Replica Reallocation Adaptive Schemes

As the size and computing power of large server clusters continue to increase, the energy consumption and heat emission of those systems are becoming very important concerns. In a different yet related front, embedded or mobile devices powered by batteries heavily rely on energy conservation to provide meaningful services. For those systems, computing cycles and network/disk bandwidth, traditionally considered as system resources, are more like expenses. They all consume the fundamental system resource: energy. This calls for studies on issues like energy-aware load distribution. It also opens the door for power and energy differentiation in addition to the traditional service differentiation. Service accesses are to be differentiated based on both their service quality specifications and their respective energy consumptions. Different applications have their own most desirable service qualities at different energy costs. I will tune the fragment replica combined S-FAS scheme to different applications to achieve the optimal energy efficiency under different service quality requirements.

## 7.3 Task3: Resource Scheduling and Management Considering Heterogeneity Nature in Cloud Computing

As a new emerged and fast developing application of distributed computing, cloud computing is the delivery of computing as a service rather than a product, whereby shared resources, software, and information are provided to computers and other devices as a utility over a network. In such complex and rich resource distributed systems, heterogeneity is more obvious and common than the traditional applications. Adaptive resource scheduling is required to efficiently utilize vast system resources in response to dynamically changing user requests. Resource scheduling and management will be an interesting and important research area in order to design optimal architectures for different applications in cloud computing. I plan to explore solutions not only considering heterogeneity nature in cloud, but also the service patterns for applications and different customer requirements.

# Chapter 8
# Conclusion

This chapter concludes this dissertation by summarizing our main work and contributions.

## 8.1 Dissertation Summary

In this dissertation study, we have investigated techniques to improve security, performance, and availability of heterogeneous distributed storage systems by focusing on the heterogeneous vulnerabilities among storage nodes.

### 8.1.1 S-FAS: A Secure Fragment Allocation Scheme

In the first part of this study, we proposed S-FAS, which is a Secure Fragment Allocation Scheme. We built the static and dynamic assurance models to evaluate the S-FAS scheme. To analysis results show that fragment allocations made by S-FAS lead to enhanced security thanks to the consideration of heterogeneous vulnerabilities in distributed storage systems.

The S-FAS scheme makes fragment allocation decisions by following the four policies below:

- **Policy 1:** All the storage nodes in a distributed storage system are classified into multiple server-type groups (server group for short) based upon their various vulnerabilities. Each server group consists of storage nodes with the same vulnerability level.
- **Policy 2:** To improve security of a distributed storage system, S-FAS allocates fragments of a file to storage nodes belonging to as many different server groups as possible.

In doing so, it is impossible to compromise the file's fragments using a single successful attack method.

- **Policy 3:** The fragments of a file are allocated to nodes with a wide range of vulnerability levels all within a single cluster storage subsystem. The goal of this policy is to improve system performance by making the fragments less likely to be distributed across multiple clusters.

- **Policy 4:** The $(m, n)$ secret sharing scheme is integrated with the S-FAS allocation mechanism.

### 8.1.2 SAP: An Secure Fragment Allocation Process Module

We investigated a Secure Allocation Process (SAP) module to improve security and performance using the heterogeneous features of a distributed system. SAP considers load balancing, delayed effects caused by the workload variance of many consecutive requests, and the heterogeneities of the storage nodes in a distributed system.

### 8.1.3 A Prototype: The Implementation of S-FAS scheme and SAP Algorithm

We developed a prototype using the multi-threading technique for the S-FAS scheme with the SAP module to guide file fragment allocations. We conducted some experiments to evaluate SAP's performance. The results show that our solution can not only improve security, but also enhance the I/O throughput of heterogeneous distributed storage systems. We also replayed real traces on our prototype to evaluate SAP's performance impacts on data-intensive applications.

### 8.1.4 Reef: An Replication Scheme to Improve Availability

It is critical to maintain a high availability of files stored in a distributed storage system, even when some storage nodes in the system are out of service.

In this Chapter we proposed a solution called Reef to improve the distributed storage system security and availability by integrating the fragment replication technique, secret sharing, fragment replication. Reef considers heterogeneity features of distributed storage systems during the replica placement phase. The Reef scheme is an extension of the S-FAS scheme. The system model for Reef is similar to that of S-FAS except that Reef address the system failures mode and aims to improve system reliability in addition to security. We build a static assurance model to quantitatively evaluate the system assurance for the Reef scheme. We also developed a replica allocation process algorithm called R-SAP to demonstrate how does the proposed Reef scheme work. In the Reef design, we addressed the distributed system security, performance, as well as availability. To evaluate the assurance provided by Reef, we studied the impacts caused by replication degree, system size, and the number of fragments.

In this dissertation study, we only focus on the static replica placement solution. Dynamic replica reallocation schemes are essential to achieve high performance and availability of distributed systems, especially for internet based applications and services. In a dynamic wide-area environment, client access patterns, network conditions, and service characteristics are constantly changing. We plan to study and propose a dynamic replica reallocation approach for heterogeneous distributed system by extending Reef to address the heterogeneous vulnerabilities in the large scale distributed systems.

## 8.2 Contributions

In this dissertation study we made five main contributions. We aim to achieve security, performance, and availability in heterogeneous distributed storage systems.

1. We addressed the heterogeneous vulnerability issue by dividing storage nodes of a distributed system into different server-type groups based on their vulnerabilities. Each server-type group - representing a level of vulnerability - contains storage nodes with the same security vulnerability. We proposed a secure fragmentation allocation scheme

called S-FAS to improve security of a distributed system where storage nodes have a wide variety of vulnerabilities.

2. We developed storage assurance and dynamic assurance models to quantify information assurance and to evaluate the proposed S-FAS scheme. We discovered principles to improve assurance levels of heterogeneous distributed storage systems. The principles are general guidelines to help designers achieve a secure fragment allocation solution for distributed systems.

3. We developed a secure allocating processing module or SAP to improve security and system performance by considering the heterogeneous features of distributed systems.

4. In order to conduct the performance analysis for the S-FAS scheme and SAP allocating algorithm, we developed a prototype for our S-FAS scheme. We implemented the prototype and conducted some experiments on the throughput of the proposed scheme and algorithm.

5. To improve the distributed storage system availability, we proposed the Reef scheme by integrating fragment replicas into our S-FAS and SAP for heterogeneous distributed systems. We developed a secure fragment allocation process module – R-SAP – to demonstrate the effectiveness of the proposed Reef scheme. The evaluation results show that the proposed Reef scheme and R-SAP can improve both availability and security of heterogeneous distributed systems.

# Bibliography

[1] Computer clusters.

[2] Distributed computing systems.

[3] F. Abdoli and M. Kahani. Ontology-based distributed intrusion detection system. In *Computer Conference, 2009. CSICC 2009. 14th International CSI*, pages 65–70, 2009.

[4] Marcos K. Aguilera, Arif Merchant, Mehul Shah, Alistair Veitch, and Christos Karamanolis. Sinfonia: a new paradigm for building scalable distributed systems. *SIGOPS Oper. Syst. Rev.*, 41(6):159–174, October 2007.

[5] A. Ahmed, A. Abdullah, and P. D D Dominic. A multi-agent based replication strategy for improving availability and maintaining consistency of data in large scale mobile traffic control environments. In *High Capacity Optical Networks and Enabling Technologies, 2008. HONET 2008. International Symposium on*, pages 163–168, 2008.

[6] A. Ahmed, P. D D Dominic, and H. Ibrahim. A binary hybrid replication strategy for improving availability and maintaining consistency of data in large scale mobile environments. In *Information Technology, 2008. ITSim 2008. International Symposium on*, volume 3, pages 1–9, 2008.

[7] G. Ahrens, A. Chandra, M. Kanthanathan, and D.P. Cox. Evaluating hacmp/6000: a clustering solution for high availability distributed systems. In *Fault-Tolerant Parallel and Distributed Systems, 1994., Proceedings of IEEE Workshop on*, pages 2–9, 1994.

[8] R. Al-Ekram and R. Holt. Multi-consistency data replication. In *Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on*, pages 568–577, 2010.

[9] B.A. Alqaralleh, Chen Wang, Bing Bing Zhou, and A.Y. Zomaya. Effects of replica placement algorithms on performance of structured overlay networks. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8, 2007.

[10] H. Alustwani, J.M. Bahi, and A. Mostefaoui. Improving data availability in p2p streaming systems using distributed virtual cache. In *Multimedia, 2008. ISM 2008. Tenth IEEE International Symposium on*, pages 384–389, 2008.

[11] L.R. Anikode and Bin Tang. Integrating scheduling and replication in data grids with performance guarantee. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–6, 2011.

[12] J.E. Armendariz-Inigo, J.R. Juarez-Rodriguez, H. Decker, and F.D. Munoz-Escoi. Trying to cater for replication consistency and integrity of highly available data. In *Database and Expert Systems Applications, 2006. DEXA '06. 17th International Workshop on*, pages 553–557, 2006.

[13] Henri E. Bal, M. Frans Kaashoek, Andrew S. Tanenbaum, and Jack Jansen. Replication techniques for speeding up parallel applications on distributed systems. *Concurrency: Practice and Experience*, 4(5):337–355, 1992.

[14] S. Balaji, L. Jenkins, L. M. Patnaik, and P. S. Goel. Workload redistribution for fault-tolerance in a hard real-time distributed computing system. In *Fault-Tolerant Computing, 1989. FTCS-19. Digest of Papers., Nineteenth International Symposium on*, pages 366–373, 1989.

[15] Peter C. Bates. Debugging heterogeneous distributed systems using event-based models of behavior. *ACM Trans. Comput. Syst.*, 13(1):1–31, February 1995.

[16] Adam Beguelin, Erik Seligman, and Peter Stephan. Application level fault tolerance in heterogeneous networks of workstations. *Journal of Parallel and Distributed Computing*, 43(2):147 – 155, 1997.

[17] William H. Bell, David G. Cameron, Luigi Capozza, A. Paul Millar, Kurt Stockinger, and Floriano Zini. Simulation of dynamic grid replication strategies in optorsim. In *Proceedings of the Third International Workshop on Grid Computing*, GRID '02, pages 46–57, London, UK, UK, 2002. Springer-Verlag.

[18] G. Benson, W. Appelbe, and I. Akyildiz. The hierarchical model of distributed system security. In *Security and Privacy, 1989. Proceedings., 1989 IEEE Symposium on*, pages 194–203, 1989.

[19] Tracy D Braun, Howard Jay Siegel, Noah Beck, Ladislau L Blni, Muthucumaru Maheswaran, Albert I Reuther, James P Robertson, Mitchell D Theys, Bin Yao, Debra Hensgen, and Richard F Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810 – 837, 2001.

[20] G. Cabri, A. Corradi, and F. Zambonelli. Experience of adaptive replication in distributed file systems. In *EUROMICRO 96. Beyond 2000: Hardware and Software Design Strategies., Proceedings of the 22nd EUROMICRO Conference*, pages 459–466, 1996.

[21] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, March 1996.

[22] Ruay-Shiung Chang, Jih-Sheng Chang, and Shin-Yi Lin. Job scheduling and data replication on data grids. *Future Gener. Comput. Syst.*, 23(7):846–860, August 2007.

[23] D.R. Cheriton and C.L. Williamson. Vmtp as the transport layer for high-performance distributed systems. *Communications Magazine, IEEE*, 27(6):37–44, 1989.

[24] D.R. Cheriton and C.L. Williamson. Vmtp as the transport layer for high-performance distributed systems. *Communications Magazine, IEEE*, 27(6):37–44, 1989.

[25] J.C.Y. Chou, Tai-Yi Huang, and Kuang-Li Huang. Scallop: a scalable and load-balanced peer-to-peer lookup protocol for high-performance distributed systems. In *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, pages 19–26, 2004.

[26] M. Chovanec, L. Vokorkos, and J. Perhac. Security architecture based on multilayer distributed intrusion detection system. In *Applied Computational Intelligence and Informatics, 2009. SACI '09. 5th International Symposium on*, pages 301–306, 2009.

[27] A.T. Chronopoulos and D. Grosu. Static load balancing for cfd simulations on a network of workstations. In *Network Computing and Applications, 2001. NCA 2001. IEEE International Symposium on*, pages 364 –367, 2001.

[28] Xie Chuiyi, Zhang Yizhi, Bai Yuan, Luo Shuoshan, and Xu Qin. A distributed intrusion detection system against flooding denial of services attacks. In *Advanced Communication Technology (ICACT), 2011 13th International Conference on*, pages 878–881, 2011.

[29] Byung-Gon Chun, Frank Dabek, Andreas Haeberlen, Emil Sit, Hakim Weatherspoon, M. Frans Kaashoek, John Kubiatowicz, and Robert Morris. Efficient replica maintenance for distributed storage systems. In *Proceedings of the 3rd conference on Networked Systems Design and Implementation Volume 3*, NSDI 06, pages 4–4, Berkeley, CA, USA, 2006. USENIX Association.

[30] George Copeland, William Alexander, Ellen Boughter, and Tom Keller. Data placement in bubba. *SIGMOD Rec.*, 17(3):99–108, 1988.

[31] A. Costan, C. Dobre, F. Pop, C. Leordeanu, and V. Cristea. A fault tolerance approach for distributed systems using monitoring based replication. In *Intelligent Computer Communication and Processing (ICCP), 2010 IEEE International Conference on*, pages 451–458, 2010.

[32] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 202–215, New York, NY, USA, 2001. ACM.

[33] Y.S Dai, M Xie, K.L Poh, and G.Q Liu. A study of service reliability and availability for distributed systems. *Reliability Engineering and System Safety*, 79(1):103 – 112, 2003.

[34] Nhan Nguyen Dang, Soonwook Hwang, and Sang Boem Lim. Improvement of data grid's performance by combining job scheduling with dynamic replication strategy. In *Grid and Cooperative Computing, 2007. GCC 2007. Sixth International Conference on*, pages 513–520, 2007.

[35] Y. Deswarte, L. Blain, and J.-C. Fabre. Intrusion tolerance in distributed computing systems. In *Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on*, pages 110 –121, May 1991.

[36] J.R. Douceur and R.P. Wattenhofer. Optimizing file availability in a secure serverless distributed file system. In *Reliable Distributed Systems, 2001. Proceedings. 20th IEEE Symposium on*, pages 4–13, 2001.

[37] A. Elghirani, R. Subrata, and A.Y. Zomaya. Intelligent scheduling and replication in datagrids: a synergistic approach. In *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, pages 179–182, 2007.

[38] A. Elghirani, R. Subrata, A.Y. Zomaya, and A. Al Mazari. Performance enhancement through hybrid replication and genetic algorithm co-scheduling in data grids. In *Computer Systems and Applications, 2008. AICCSA 2008. IEEE/ACS International Conference on*, pages 436–443, 2008.

[39] Hyeonsang Eom and J.K. Hollingsworth. Speed vs. accuracy in simulation for i/o-intensive applications. In *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, pages 315 –322, 2000.

[40] H.M.A. Fahmy and A.A. El-Hefnawy. Fault-tolerance-based computation of global functions in asynchronous distributed systems. In *Electronics, Circuits and Systems, 2001. ICECS 2001. The 8th IEEE International Conference on*, volume 2, pages 789–793 vol.2, 2001.

[41] Joseph R. Falcone. A programmable interface language for heterogeneous distributed systems. *ACM Trans. Comput. Syst.*, 5(4):330–351, October 1987.

[42] Fei fei Li, Xiang zhan Yu, and Gang Wu. Design and implementation of high availability distributed system based on multi-level heartbeat protocol. In *Control, Automation and Systems Engineering, 2009. CASE 2009. IITA International Conference on*, pages 83–87, 2009.

[43] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. In *High Performance Distributed Computing, 1997. Proceedings. The Sixth IEEE International Symposium on*, pages 365–375, 1997.

[44] Daniel Ford, Franois Labelle, Florentina I. Popovici, Murray Stokely, Van anh Truong, Luiz Barroso, Carrie Grimes, Sean Quinlan, and Google Inc. Availability in globally distributed storage systems.

[45] Wei Fu, Nong Xiao, and Xicheng Lu. A quantitative survey on qos-aware replica placement. In *Grid and Cooperative Computing, 2008. GCC '08. Seventh International Conference on*, pages 281–286, 2008.

[46] Xiong Fu, Ruchuan Wang, Yang Wang, and Song Deng. A replica placement algorithm in mobile grid environments. In *Embedded Software and Systems, 2009. ICESS '09. International Conference on*, pages 601–606, 2009.

[47] Lei Gao, M. Dahlin, A. Nayate, Jiandan Zheng, and Arun lyengar. Improving availability and performance with application-specific data replication. *Knowledge and Data Engineering, IEEE Transactions on*, 17(1):106–120, 2005.

[48] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. 2003.

[49] A. Grzech. Optimization of two-level topological structure of distributed intrusion detection system. In *Systems Engineering, 2008. ICSENG '08. 19th International Conference on*, pages 337–342, 2008.

[50] B. Hamid, A. Radermacher, P. Vanuxeem, A. Lanusse, and S. Gerard. A fault-tolerance framework for distributed component systems. In *Software Engineering and Advanced Applications, 2008. SEAA '08. 34th Euromicro Conference*, pages 84–91, 2008.

[51] J.W. Hanna and J.D. Johannes. A reliable distributed system using dual level fault tolerance. In *Southeastcon '92, Proceedings., IEEE*, pages 610–613 vol.2, 1992.

[52] C. Hannon and J.R. Rinewalt. Addressing security issues in geographically distributed systems. In *Computer Science, 2003. ENC 2003. Proceedings of the Fourth Mexican International Conference on*, pages 182–189, 2003.

[53] S. Hariri and H. Mutlu. Hierarchical modeling of availability in distributed systems. *Software Engineering, IEEE Transactions on*, 21(1):50–56, 1995.

[54] S. Hariri and H.B. Mutlu. A hierarchical modeling of availability in distributed systems. In *Distributed Computing Systems, 1991., 11th International Conference on*, pages 190–197, 1991.

[55] D. Herz and A.M. Deplanche. Fault-tolerance operators for distributed real-time control systems. In *Distributed Computing Systems, 1990. Proceedings., Second IEEE Workshop on Future Trends of*, pages 113–119, 1990.

[56] Hui-I Hsiao and D.J. DeWitt. A performance study of three high availability data replication strategies. In *Parallel and Distributed Information Systems, 1991., Proceedings of the First International Conference on*, pages 18–28, 1991.

[57] Weijian Huang, Yan An, and Wei Du. A multi-agent-based distributed intrusion detection system. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 3, pages V3–141–V3–143, 2010.

[58] Weijian Huang, Yan An, and Wei Du. A multi-agent-based distributed intrusion detection system. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 3, pages V3–141–V3–143, 2010.

[59] Yixiu Huang, Prasad Sistla, and Ouri Wolfson. Data replication for mobile computers. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, SIGMOD '94, pages 13–24, New York, NY, USA, 1994. ACM.

[60] Pankaj Jalote. *Fault tolerance in distributed systems.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[61] B. Javadi, D. Kondo, J.-M. Vincent, and D.P. Anderson. Discovering statistical models of availability in large distributed systems: An empirical study of seti@home. *Parallel and Distributed Systems, IEEE Transactions on*, 22(11):1896–1903, 2011.

[62] A. Kermarrec, E.L. Merrer, G. Straub, and A. Van Kempen. Availability-based methods for distributed storage systems. In *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, pages 151–160, 2012.

[63] D. L. Kewley and J. F. Bouchard. Darpa information assurance program dynamic defense experiment summary. *IEEE Trans. on Systems, Man and Cybernetics, Part A: Systems and Humans*, 31(4):331 –336, Jul 2001.

[64] Vishal Kher and Yongdae Kim. Securing distributed storage: challenges, techniques, and systems. In *Proceedings of the 2005 ACM workshop on Storage security and survivability*, StorageSS '05, pages 9–25, New York, NY, USA, 2005. ACM.

[65] K.H. Kim. Designing fault tolerance capabilities into real-time distributed computer systems. In *Distributed Computing Systems in the 1990s, 1988. Proceedings., Workshop on the Future Trends of*, pages 318–328, 1988.

[66] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: an architecture for global-scale persistent storage. *SIGPLAN Not.*, 35:190–201, November 2000.

[67] J. G. Kuhl and S. M. Reddy. Distributed fault-tolerance for large multiprocessor systems. In *Proceedings of the 7th annual symposium on Computer Architecture*, ISCA '80, pages 23–30, New York, NY, USA, 1980. ACM.

[68] James F. Kurose and Rahul Simha. A microeconomic approach to optimal resource allocation in distributed computer systems. *IEEE Trans. Comput.*, 38(5):705–717, 1989.

[69] H. Lamehamedi, Z. Shentu, B. Szymanski, and E. Deelman. Simulation of dynamic data replication strategies in data grids. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 10 pp.–, 2003.

[70] H. Lamehamedi, B. Szymanski, Z. Shentu, and E. Deelman. Data replication strategies in grid environments. In *Algorithms and Architectures for Parallel Processing, 2002. Proceedings. Fifth International Conference on*, pages 378–383, 2002.

[71] Lin-Wen Lee, Peter Scheuermann, and Radek Vingralek. File assignment in parallel i/o systems with minimal variance of service time. *IEEE Trans. Comput.*, 49(2):127–140, 2000.

[72] Wei Li, Shanping Li, and Xingen Wang. Load balance optimization with replication degree customization. In *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*, pages 170–174, 2011.

[73] Ching Lin and V. Varadharajan. A hybrid trust model for enhancing security in distributed systems. In *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*, pages 35–42, 2007.

[74] Yi Lin, Bettina Kemme, Marta Patiño Martínez, and Ricardo Jiménez-Peris. Middleware based data replication providing snapshot isolation. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 419–430, New York, NY, USA, 2005. ACM.

[75] Jianxiao Liu and Li Lijuan. A distributed intrusion detection system based on agents. In *Computational Intelligence and Industrial Application, 2008. PACIIA '08. Pacific-Asia Workshop on*, volume 1, pages 553–557, 2008.

[76] Miron Livny and Myron Melman. Load balancing in homogeneous broadcast distributed systems. *SIGMETRICS Perform. Eval. Rev.*, 11(1):47–55, April 1982.

[77] O.G. Loques and J. Kramer. Flexible fault tolerance for distributed computer systems. *Computers and Digital Techniques, IEE Proceedings E*, 133(6):319–332, 1986.

[78] T. Loukopoulos and I. Ahmad. Static and adaptive data replication algorithms for fast information access in large distributed systems. In *Distributed Computing Systems, 2000. Proceedings. 20th International Conference on*, pages 385–392, 2000.

[79] Thanasis Loukopoulos and Ishfaq Ahmad. Static and adaptive distributed data replication using genetic algorithms. *Journal of Parallel and Distributed Computing*, 64(11):1270 – 1285, 2004.

[80] Najme Mansouri and GholamHosein Dastghaibyfard. Job scheduling and dynamic data replication in data grid environment. *The Journal of Supercomputing*, 64(1):204–225, 2013.

[81] H. Mantel. On the composition of secure systems. In *2002 IEEE Symposium on Security and Privacy.*, 2002.

[82] A. Mei, L. V. Mancini, and S. Jajodia. Secure dynamic fragment and replica allocation in large-scale distributed file systems. *IEEE Trans. on Parallel and Distributed Systems*, 14(9):885 – 896, Sept. 2003.

[83] Zhou Mingqiang, Huang Hui, and Wang Qian. A graph-based clustering algorithm for anomaly intrusion detection. In *Computer Science Education (ICCSE), 2012 7th International Conference on*, pages 1311–1314, 2012.

[84] Naftaly H. Minsky and Victoria Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *ACM Trans. Softw. Eng. Methodol.*, 9(3):273–305, July 2000.

[85] Ravi Mirchandaney, Don Towsley, and John A. Stankovic. Adaptive load sharing in heterogeneous distributed systems. *Journal of Parallel and Distributed Computing*, 9(4):331 – 346, 1990.

[86] G. H M B Motta and S.S. Furuie. A contextual role-based access control authorization model for electronic patient record. *Information Technology in Biomedicine, IEEE Transactions on*, 7(3):202–207, 2003.

[87] S. Naqvi and M. Riguidel. Security architecture for heterogeneous distributed computing systems. In *Security Technology, 2004. 38th Annual 2004 International Carnahan Conference on*, pages 34–41, 2004.

[88] D.M. Nessett. Factors affecting distributed system security. *Software Engineering, IEEE Transactions on*, SE-13(2):233–248, 1987.

[89] D.M. Nessett. Factors affecting distributed system security. *Software Engineering, IEEE Transactions on*, SE-13(2):233–248, 1987.

[90] E.B. Noeparast and T. Banirostam. A cognitive model of immune system for increasing security in distributed systems. In *Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference on*, pages 181–186, 2012.

[91] D.T. Nukarapu, Bin Tang, Liqiang Wang, and Shiyong Lu. Data replication in data intensive scientific applications with performance guarantee. *Parallel and Distributed Systems, IEEE Transactions on*, 22(8):1299–1306, 2011.

[92] Brian M. Oki and Barbara H. Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*, PODC '88, pages 8–17, New York, NY, USA, 1988. ACM.

[93] Brian M. Oki and Barbara H. Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*, PODC '88, pages 8–17, New York, NY, USA, 1988. ACM.

[94] Salvatore Orlando and Raffaele Perego. Exploiting partial replication in unbalanced parallel loop scheduling on multicomputer. *Microprocessing and Microprogramming*, 41(89):645 – 658, 1996. ¡ce:title¿Parallel Systems Engineering¡/ce:title¿.

[95] J. Osrael, L. Froihofer, and K.M. Goeschka. A system architecture for enhanced availability of tightly coupled distributed systems. In *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on*, pages 8 pp.–, 2006.

[96] Prasanna Padmanabhan, Le Gruenwald, Anita Vallur, and Mohammed Atiquzzaman. A survey of data replication techniques for mobile ad hoc network databases. *The VLDB Journal*, 17(5):1143–1164, August 2008.

[97] Seong-Jin Park and Doo-Kwon Baik. A data allocation considering data availability in distributed database systems. In *Parallel and Distributed Systems, 1997. Proceedings., 1997 International Conference on*, pages 708–713, 1997.

[98] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 129–140, London, UK, 1992. Springer-Verlag.

[99] F. Pop, A. Arcalianu, C. Dobre, and V. Cristea. Enhanced security for monitoring services in large scale distributed systems. In *Intelligent Computer Communication and Processing (ICCP), 2011 IEEE International Conference on*, pages 549–556, 2011.

[100] Gerald J. Popek, R.S. Guy, Jr. Page, T.W., and J.S. Heidemann. Replication in ficus distributed file systems. In *Management of Replicated Data, 1990. Proceedings., Workshop on the*, pages 5–10, 1990.

[101] M. Pourzandi, D. Gordon, W. Yurcik, and G. A. Koenig. Clusters and security: distributed security for distributed systems. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 1, pages 96 – 104 Vol. 1, May 2005.

[102] D.K. Pradhan and S.M. Reddy. A fault-tolerant communication architecture for distributed systems. *Computers, IEEE Transactions on*, C-31(9):863–870, 1982.

[103] Xia Qing. The structure design of a new distributed intrusion detection system. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, volume 7, pages V7–100–V7–103, 2010.

[104] Xiaohong Qu, Zhijie Liu, and Xiaoyao Xie. Research on distributed intrusion detection system based on protocol analysis. In *Anti-counterfeiting, Security, and Identification in Communication, 2009. ASID 2009. 3rd International Conference on*, pages 421–424, 2009.

[105] M. Saito, T. Yokoyama, and M. Shimada. Lazy fault tolerance-a method for dependable distributed systems. In *Object-Oriented Real-Time Dependable Systems, 1994. Proceedings of WORDS 94., First Workshop on*, pages 124–131, 1994.

[106] Harjinder S. Sandhu and Songnian Zhou. Cluster-based file replication in large-scale distributed systems. *SIGMETRICS Perform. Eval. Rev.*, 20(1):91–102, June 1992.

[107] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[108] Peter Scheuermann, Gerhard Weikum, and Peter Zabback. Data partitioning and load balancing in parallel disk systems. *The VLDB Journal*, 7(1):48–66, 1998.

[109] Robert R. Seban. A high performance critical section protocol for distributed systems. In *Aerospace Applications Conference, 1994. Proceedings., 1994 IEEE*, pages 1–17, 1994.

[110] J. Sen. A robust and fault-tolerant distributed intrusion detection system. In *Parallel Distributed and Grid Computing (PDGC), 2010 1st International Conference on*, pages 123–128, 2010.

[111] J. Sen, I. Sengupta, and P.R. Chowdhury. An architecture of a distributed intrusion detection system using cooperating agents. In *Computing Informatics, 2006. ICOCI '06. International Conference on*, pages 1–6, 2006.

[112] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[113] Chao Shen and Shengjun Xue. Design and implementation of distributed collaborative intrusion detection system model. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on*, volume 3, pages 1224–1228, 2010.

[114] E. Shokri, H. Hecht, P. Crane, J. Dussdault, and K.H. Kim. An approach for adaptive fault-tolerance in object-oriented open distributed systems. In *Object-Oriented Real-Time Dependable Systems, 1997. Proceedings., Third International Workshop on*, pages 298–305, 1997.

[115] M. Shorfuzzaman, P. Graham, and R. Eskicioglu. Distributed placement of replicas in hierarchical data grids with user and system qos constraints. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2011 International Conference on*, pages 177–186, 2011.

[116] G. J. Simmons. How to (really) share a secret. In *CRYPTO '88: Proceedings of the 8th Annual International Cryptology Conference on Advances in Cryptology*, pages 390–448, London, UK, 1990. Springer-Verlag.

[117] Swaminathan Sivasubramanian, Gustavo Alonso, Guillaume Pierre, and Maarten van Steen. Globedb: autonomic data replication for web applications. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 33–42, New York, NY, USA, 2005. ACM.

[118] D.L. Smarkusky, R.A. Ammar, and H.A. Sholl. A framework for designing performance-oriented distributed systems. In *Computers and Communications, 2001. Proceedings. Sixth IEEE Symposium on*, pages 92–98, 2001.

[119] S.H. Son, Fengjie Zhang, and Ji-Hoon Kang. Replication control for fault-tolerance in distributed real-time database systems. In *Aerospace Conference, 1998 IEEE*, volume 4, pages 73–81 vol.4, 1998.

[120] M. Soshi and M. Maekawa. The saga security system: a security architecture for open distributed systems. In *Distributed Computing Systems, 1997., Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of*, pages 53–58, 1997.

[121] Ming Tang, Bu sung Lee, Xueyan Tang, and Chai kiat Yeo. The impact of data replication on job scheduling performance. In *in the Data Grid, Future Generation Computer Systems*, pages 254–268, 2006.

[122] B. M. Thuraisingham and J. A. Maurer. Information survivability for evolvable and adaptable real-time command and control systems. *Knowledge and Data Engineering, IEEE Transactions on*, 11(1):228 –238, 1999.

[123] Li Tian. Design and implementation of a distributed intelligent network intrusion detection system. In *Electrical and Control Engineering (ICECE), 2010 International Conference on*, pages 683–686, 2010.

[124] Y. Tian, M. I. Alghamdi, Y. Shu, J. Xie, J. Zhang, M.-K. Qiu, Y.-M. Yang, and X. Qin. Secure fragment allocation in a distributed storage system with heterogeneous vulnerabilities. In *Proceedings of the 2011 IEEE International Conference on Networking, Architecture, and Storage*, NAS '11. IEEE Computer Society, 2011.

[125] B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, and D. Gunter. The netlogger methodology for high performance distributed systems performance analysis. In *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*, pages 260–267, 1998.

[126] R. Tirtea, G. Deconinck, and R. Belmans. Fault tolerance adaptation requirements vs. quality-of-service, realtime and security in dynamic distributed systems. In *Reliability and Maintainability Symposium, 2006. RAMS '06. Annual*, pages 296–303, 2006.

[127] P. Triantafillou and D.J. Taylor. Velos: a new approach for efficiently achieving high availability in partitioned distributed systems. *Knowledge and Data Engineering, IEEE Transactions on*, 8(2):305–321, 1996.

[128] C. Trinitis, M. Walter, and M. Leberecht. Balanced high availability in layered distributed computing systems. In *Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on*, pages 713–717, 2003.

[129] Manghui Tu, Peng Li, I-Ling Yen, Bhavani Thuraisingham, and Latifur Khan. Secure data objects replication in data grid. *IEEE Transactions on Dependable and Secure Computing*, 7:50–64, 2010.

[130] V. Varadharajan and S. Black. A multilevel security model for a distributed object-oriented system. In *Computer Security Applications Conference, 1990., Proceedings of the Sixth Annual*, pages 68–78, 1990.

[131] V. Venkatesan, I. Iliadis, Xiao-Yu Hu, R. Haas, and C. Fragouli. Effect of replica placement on the reliability of large-scale data storage systems. In *Modeling, Analysis*

*Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pages 79–88, 2010.

[132] L. Vokorokos, M. Chovanec, O. Latka, and A. Kleinova. Security of distributed intrusion detection system based on multisensor fusion. In *Applied Machine Intelligence and Informatics, 2008. SAMI 2008. 6th International Symposium on*, pages 19–24, 2008.

[133] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: a scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pages 307–320, Berkeley, CA, USA, 2006. USENIX Association.

[134] Ouri Wolfson, Sushil Jajodia, and Yixiu Huang. An adaptive data replication algorithm. *ACM Trans. Database Syst.*, 22(2):255–314, June 1997.

[135] T. Y C Woo and S.S. Lam. Authentication for distributed systems. *Computer*, 25(1):39–52, 1992.

[136] Jun Wu, Chong-Jun Wang, Jun Wang, and Shi-Fu Chen. Dynamic hierarchical distributed intrusion detection system based on multi-agent system. In *Web Intelligence and Intelligent Agent Technology Workshops, 2006. WI-IAT 2006 Workshops. 2006 IEEE/WIC/ACM International Conference on*, pages 89–93, 2006.

[137] T. Wu, M. Malkin, and D. Boneh. Building intrusion tolerant applications. In *SSYM'99: Proceedings of the 8th conference on USENIX Security Symposium*, pages 7–7, Berkeley, CA, USA, 1999. USENIX Association.

[138] J. J. Wylie, M. W. Bigrigg, J. D. Strunk, G. R. Ganger, H. Kiliccote, and P. K. Khosla. Survivable information storage systems. *Computer*, 33(8):61 –68, Aug 2000.

[139] Li Xiao, Sonqing Chen, and Xiaodong Zhang. Dynamic cluster resource allocations for jobs with known and unknown memory demands. *IEEE Trans. Parallel Distrib. Syst.*, 13:223–240, March 2002.

[140] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian, J. Majors, A. Manzanares, and Xiao Qin. Improving mapreduce performance through data placement in heterogeneous hadoop clusters. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–9, 2010.

[141] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian, J. Majors, A. Manzanares, and Xiao Qin. Improving mapreduce performance through data placement in heterogeneous hadoop clusters. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–9, 2010.

[142] Tao Xie and Xiao Qin. A security-oriented task scheduler for heterogeneous distributed systems. In *Proc. 13th Annual IEEE Intl Conf. High Performance Computing, Lecture Notes in Computer Science (LNCS 4297), ISSN 0302-9743*, pages 35–46, 2006.

[143] Naixue Xiong, Yan Yang, Ming Cao, Jing He, and Lei Shu. A survey on fault-tolerance in distributed network systems. In *Computational Science and Engineering, 2009. CSE '09. International Conference on*, volume 2, pages 1065–1070, 2009.

[144] Chao-Tung Yang, Chun-Pin Fu, Chien-Jung Huang, and Ching-Hsien Hsu. Frcs: A file replication and consistency service in data grids. In *Multimedia and Ubiquitous Engineering, 2008. MUE 2008. International Conference on*, pages 444–447, 2008.

[145] Chao-Tung Yang, Chien-Jung Huang, and Ting-Chih Hsiao. A data grid file replication maintenance strategy using bayesian networks. In *Intelligent Systems Design and Applications, 2008. ISDA '08. Eighth International Conference on*, volume 1, pages 456–461, 2008.

[146] Yun Yang and Jia Mi. Design and implementation of distributed intrusion detection system based on honeypot. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, volume 6, pages V6–260–V6–263, 2010.

[147] Chihsiang Yeh. The robust middleware approach for transparent and systematic fault tolerance in parallel and distributed systems. In *Parallel Processing, 2003. Proceedings. 2003 International Conference on*, pages 61–68, 2003.

[148] W. Yurcik, G. A. Koenig, X. Meng, and J. Greenseid. Cluster security as a unique problem with emergent properties: Issues and techniques. In *5th LCI International Conference on Linux Clusters: The HPC Revolution 2004*, pages 18–20, 2004.

[149] G. Zanin, A. Mei, and L. V. Mancini. Towards a secure dynamic allocation of files in large scale distributed file systems. In *HOT-P2P '04: Proceedings of the 2004 International Workshop on Hot Topics in Peer-to-Peer Systems*, pages 102–107, Washington, DC, USA, 2004. IEEE Computer Society.

[150] Kailong Zhang, Ke Liang, Xingshe Zhou, Kaibo Wang, Xiao Wu, and Zhiyi Yang. A similar resource auto-discovery based adaptive fault-tolerance method for embedded distributed system. In *Parallel Processing Workshops, 2007. ICPPW 2007. International Conference on*, pages 21–21, 2007.

[151] Wuqing Zhao, Xianbin Xu, Zhuowei Wang, Yuping Zhang, and Shuibing He. Improve the performance of data grids by value-based replication strategy. In *Semantics Knowledge and Grid (SKG), 2010 Sixth International Conference on*, pages 313–316, 2010.

[152] Lin Zhao-wen, Ren Xing-tian, and Ma Yan. Agent-based distributed cooperative intrusion detection system. In *Communications and Networking in China, 2007. CHINACOM '07. Second International Conference on*, pages 17–22, 2007.

[153] L. Zheng, S. Chong, A.C. Myers, and S. Zdancewic. Using replication and partitioning to build secure distributed systems. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, pages 236–250, 2003.

[154] Ming Zhong, Kai Shen, and Joel Seiferas. Replication degree customization for high availability. *SIGOPS Oper. Syst. Rev.*, 42(4):55–68, April 2008.

[155] Jing Zhou, Yijie Wang, and Sikun Li. Data dependence-based optimistic data consistency maintenance method. In *Computer and Information Technology, 2006. CIT '06. The Sixth IEEE International Conference on*, pages 120–120, 2006.