Department of Computer Science

College of Science and Technology

Deanship of Graduated Studies

FOODS: Object Oriented Framework For Distributed File System

By

Prepared By: Radwan Fayez Hassan Qasrawi

Registration No: 20011551

Supervisor: Dr.Badie Sartawi

Master thesis submitted and accepted, date: 15/07/2005

The names and signatures of the examining committee members as follows:

1- Dr.Badie Sartawi, Head of Committee Signature:

2- Dr. Nidal Al- Kafri, Internal Examiner Signature:

3- Dr. Mahmoud Al-Sahib, External Examiner Signature:

Al-Quds University
2005

Abstract:

Distributed File System (DFS) has gained prominence since it increases files availability and accessibility. Although several models were employed, we selected the software design reuse based on Object Oriented Framework approach. This model called "FOODS" introduces the multi-master replication (MMR) approach in order to increase the availability as well as the client side cache approach that increases the performance. The "FOODS" takes the responsibility for providing synchronized access and consistent data, the synchronized access achieved by using a timestamp mechanism that determines which server has the latest version of a file, and will notify the one with outdated versions then retrieves the latest version from server to other servers, as well as "FOODS" uses client side cache approach with the Least Recently Used (LRU) cache replacement policy, the data cached on the client side, and only informs the server if updates made to the file system. For design evaluation we built two types of simulators, one measuring the hit and miss rates for reading and writing operation as a functions of cache size, while the second one for measuring the average response time between clients and servers. The simulators show that the FOODS model provides an increase in the performance and stable system.

ملخص البحث:

إن نظام الملف الموزع(DFS) والذي هو عبارة عن توزيع الملفات على عدة أجهزة حاسوب متصلة مع بعضها بواسطة شبكات الحاسوب , قد لقي اهتماما من المختصين لما له من مزايا في زيادة تواجد المعلومات وسهولة الوصول إليها . هنالك نماذج متعددة صممت من اجل تحقيق هذا الهدف . نقدم في هذا البحث نموذجا جديدا مبني بطريقة " إطار البرمجة الكينونة " (OOP) والتي تسمح باستخدام نماذج سابقة من اجل تصميم نموذج جديد محسن وذو كفائه عالية ، حيث يشمل هذا النموذج على عدة آليات تؤدي إلى توافر المعلومات بشكل متطابق. آلية توزيع النسخة الرئيسية من الملفات على أكثر من جهاز رئيسي (MMR) بحيث يمكن المستخدم من القراءة والتعديل على الملفات، مما يساهم في زيادة توافر الملفات بشكل دائم، إضافة إلى ذلك يقدم هذا النموذج آلية تساعد على زيادة تطابق المعلومات حيث تستند على أساس التعميم على جميع الأجهزة التي تحتوي على نسخة الملف قبل تعديله من اجل عدم اعتماد تلك النسخة واعتماد النسخة الجديدة مما يوفر لجميع المستخدمين الأمان في استخدام تلك الملفات .إضافة إلى ذلك يعمل هذا النموذج على زيادة السرعة في الحصول على المعلومات بالسماح بنقل الملفات إلى ذاكرة الجهاز المستخدم بطريقة "استبدال الملف الأقدم استخداماً بالأحدث استخداماً" (LRU) ويتم تبادل رسائل التعميم بين الأجهزة في حالة حدوث تغير على الملفات مما يزيد من التطابق في المعلومات . ومن اجل التحقق من فاعلية هذا النموذج تم بناء برنامج "محاكي أو مقلد" يقوم بقياس معدل كل من وجود أو عدم وجود الملفات في ذاكرة الحاسوب المستخدم وقياس معدل زمن الإجابة للطلب بالنسبة لزيادة عدد المستخدمين . وقد اظهر هذا البرنامج كفاءة عالية في توافر المعلومات و سرعة الحصول عليها إضافة إلى السرعة في الاستجابة لطلب المستخدم بالرغم من كثرة المستخدمين.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Introduction

The object oriented framework is one of the most effective design methods that reduces or completely solve the serious problems that may arise during the design of distributed file systems (DFS). Particularly, when the distributed file system is constructed, difficult problems like network bottleneck, server overload, data security and server failure comes to stage. Such problems prevent clients from accessing information timely and effectively. These problems attracted the attention of many interested researchers and system developers to introduce other designs attempting to reduce the problems that may take place when the data transferred in distributed systems.

Object oriented framework is a reusable design and analysis of an application or subsystem, represented by a set of abstract classes and the way the objects in these classes are interconnected. The main concept of object oriented framework is to capture the commonalities between similar applications. It provides a more generic solution in the level of abstraction for the related domain, rather than giving a concert solution for that domain.

Object oriented frameworks have many benefits such as the reusability, modularity, extensibility, and inversion of control. The main advantages of

object oriented framework are the ability to reuse the design, and the implementation of systems by defining generic components. The latter can be reapplied to create new applications. In addition, the reuse of framework components can yield substantial improvements in programming productivity, as well as enhancing the quality, performance, reliability and interoperability of the software. Moreover, frameworks will encapsulate the volatile implementation details with stable interfaces that in turn enhance the system modularity. Added to these benefits, object oriented framework has the extensibility that is related to the hook methods. These methods systematically decouple the stable interface and behavior of a particular domain during the installation process of a specific application. Framework extensibility is also essential to ensure timely customization of new application services and features [6].

A distributed file system (DFS) is the main part of a distributed computer system that enables the use of remote data and resources. The requirement for information and resources sharing, raise the problem in the development of distributed file system with high performance, availability, consistency, scalability and security. Many distributed file systems are designed using the principle of object oriented paradigm which is far from an optimal design, Andrew file system, which exhibit good performance and scalability, but suffers from availability and consistency [3], Even though the CODA file system, which depends on the idea of Andrew, have solved the problem of

availability, it is still unable to overcome the problem of consistency [3]. The Sun network file system is also suffering from many problems like sharing semantic (time independent). Semantic files update can't be visible before file closing. In addition to these, it is poor in replication, consistency, and scalability [3].

The object oriented framework supports the code and system analysis reuse which can help redesigning these systems. It also considers the weakness and strength for introducing a new design of distributed file system with more efficiency and completeness. Our framework will focus on Andrew file system, Coda file system, and Network file system (NFS) as a case study for developing a new design for the distributed file systems.

1.2 Motivation

Enhancing the efficiency of distributed file system, and reducing the problem of consistency and availability, has motivated us to use the principle of object oriented framework, which also provided us the ability of reusing system design, system analysis and code implementation for the purpose of developing a new design for distributed file system that will significantly overcome the above known problems.

1.3 Problem Definitions

Despite of its heavy usage, the distributed file systems are still causing several problems, like system consistency and availability. In DFS, in order to improve the efficiency and security, the developer usually use server-

replica and pseudo server on local site. Unfortunately, these two important mechanisms can lead to some inconsistency such as file, directory, cache, and server replication conflicts in DFS. The following are some of these main conflicts:

1- File Conflict:

a. Write-Write Conflict: Write-Write conflict means, if a shared File is updated by different clients of the DFS concurrently, after two updating operations, the second update replace the first one, so the file will lose the update.

b. Read-Write Conflict: File is changed by user A according to the contents of another file, which is also changed by user B at the same instance that user A is doing the same on the file.

2- Directory Conflict:

a. Remove/Update Conflict: An entry is removed from a directory in one partition but the corresponding object is updated in another partition.

b. Update/Update Conflict: A directory's metadata, such as its access list, is updated in two or more partitions.

c. Name/Name Conflict: Two different objects with the same name are inserted in a directory in different partitions.

3- Cache Conflict: DFS uses local cache in order to improve their performance and availability. But the uses of the cache may cause some inconsistency. For example, user A wants to read some data, which are already saved in the local cache. But at the same time, user B updates these data and saves it in the server. It leads the data read by user A is inconsistent with that saved in server. The data, read by user A, is out of date.

4- Server Replication Conflict: DFS usually use another technology "Server Replication" in order to improve their performance and availability. Thus, some inconsistency may occur when one of replication servers is broken down.

1.4 Research Objectives

Our research objectives can be briefly described in two categories. Particularly, the first identifies the problem of distributed file system in the concept of framework and the second is to introduce a new detailed design with the principle of object oriented paradigm for the purpose of resolving file conflicts in addition to the full documentation of the system domain analysis and framework design as well.

CHAPTER 6

DESIGN EVALUATION AND CONCLUSION

6.1 Design Evaluation

FOODS is a framework designed to remove both the inconsistency and the unavailability of distinct distributed file systems. The structure of FOODS system is composed of four main parts. Namely, the API (GUI) from which the user interface is represented, the client program which is responsible for communicating user operation with file server, the directory server which is the interface between the file server and client program and the file server. The later is responsible of client services. The details of the design were described in the previous chapter.

It is important to notice that; through this study, due to the focus on the system requirements, analysis and FOODS system design in addition to the lack of real operating system, the constructed FOODS design was not completely implemented. However, the above mentioned problems were removed by constructing a simulator that allows the new design control. The simulator reported here is being built up using theoretical background. The main purpose of which is to validate the new design. Specifically, it simulates the caching and the replication processes.

The simulator is composed of two parts. The first is the client cache simulation part. It is constructed to measure the hit and miss rates for the reading and the writing operations.

When user accesses a file that is available in the cache, the cache hit is recorded. On the other hand, when the file is not in cache, cache is miss recorded. Thus, the cache hit rate is defined as the number of requests satisfied by the cache divided by the total number of requests acquired by a specific user.

$$Hit\ rate = \frac{\text{Total Number of Cache Hits}}{\text{Total Number of Requests}} \qquad (1)$$

The cache miss rate is the rate of the number of times that the file is not found in the cache (cache miss) divided by the total number of requests by a specific user.

$$\text{Miss rate} = \frac{\text{Total Number of Cache Miss}}{\text{Total Number of Requests}} \qquad (2)$$

The cache simulator is based on the LRU replacement policy. The Least recently used replacement reserves a temporary data location. The temporary location means that the accessed file will be placed in blocks. These blocks may be probably accessed soon. In the LRU, the block to be replaced is the one that was accessed the farthest in the past (least recent). Thus, LRU is

widely useable on the caching process and it improves the efficiency and performance significantly.

The simulator designed through this work organizes cache in blocks. Each block has fixed size of 4.0 Kbytes out of the total maximum allowed cache size which is 10 M byte. The LRU cache is then implemented using a queue. In other words, when block is to be replaced, the block at the head of the queue will be selected because it is the least recent. At any time hit is made by a read or write process in the cache, the block is moved to the tail of the queue. In other words, the block at the queue tail is always the last block accessed and the block at the head is the block that was accessed the farthest in the past.

The cache simulator, which considers three cache design parameters (size, associativity, and block size), a sequence of addresses and returns statistics includes the number of cache hits and number of cache miss. The address sequence distinguish reads, writes data operations. The least recently used (LRU) strategy is used whenever a cache line needs to be replaced. The Cache simulator data structure is a queue indexed by the cache index and the set is returned. Each entry in the array stores the tag and a clock tick which are used to keep tracking of the element accessing time by indexing it into the address list that is passed to the simulator. The clock information is used to

implement the LRU replacement strategy with the consideration of zero entry in the clock field as an invalid cache entry.

In the LRU replacement policy which is accepted to be the most commonly used replacement scheme, the block chosen for replacement is the one that has been unused for the longest time. It is implemented by keeping track of the time of individual block usage relative to the other blocks. Every time a hit occurs in the cache, the requested element is marked as the most recently used and all other elements updated relative to the most recent. The results of this simulator are displayed in Figure 6.1.

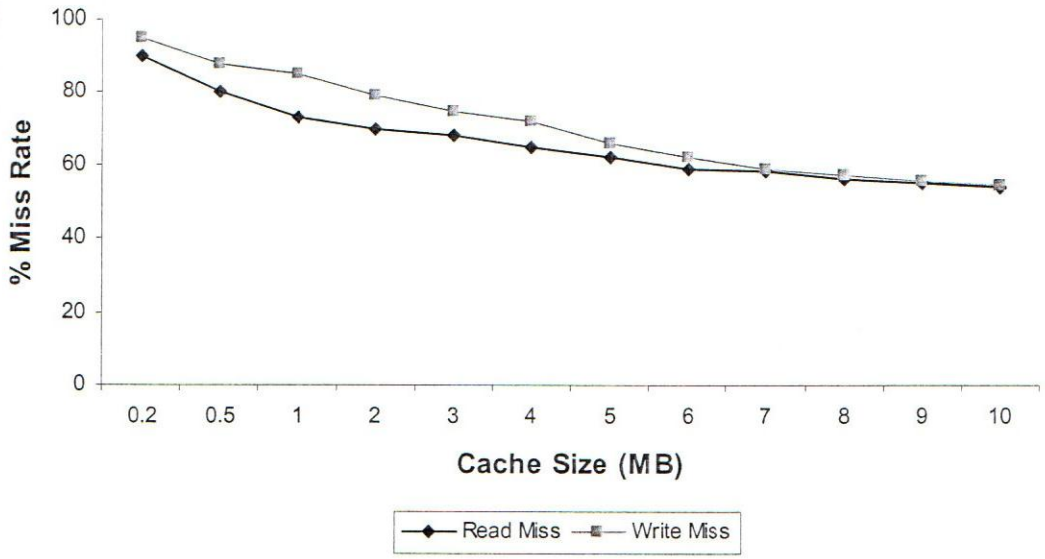

Fig 6.1 (a) % Read -Write miss rates as a function of cache size

The most significant observation from the curves displayed in figure 6.1(a) is the inversely dependent relation between the miss rate and cache size. Namely, when cache size is small the miss rate is relatively high for the read