HPC File Server Monitoring and Tuning

Rune Johan Andresen

6th August 2005

Abstract

As HPC systems grow, the distributed file systems serving these systems need to handle an increased load of data. In order to maintain performance, these underlying file servers need to distributethe load of data volumes efficiently over available disks. This is particularly true at CERN, the European European Organizationfor Nuclear Research, which expects to behandling Pentabytes of data in the near future.

In this thesis, new utilities that analyze file serverdata which is then used to semi-automatically tune thefiles system, are developed. This is achieved using a commercial database to store the dataand then integrating it with the file server. This requires a database and a system design that can handle a large amount of data.

File server data collections associated with aprocess known as "volumes", can vary in size, and be accessed at any time. To increase the overall system performance, volume history data is analyzed to locate volumes that may be gathered for increased system performance throughgh load balancing. For instance, using the volume history data, it is possible to detect and gather volumes that are most accessed during the day with volumes that are most accessed during the night on one file server. The file server capacity is hence optimized.

As part of this work, a user interface which can visualize the history data for volumes and partitions, is designed and implemented on top of the AFS file system at CERN. Our initial results presented in this thesisreveal that it is possible to locate volumes that have a repeating access period, and thus, gather them on the same partition. Other analyses and suggestions for future work will also be discussed.

Acknowledgments

This thesis was written at European Organization for Nuclear Research (CERN), while working for the CERN IT Department, Architecture and Data Challenges (ADC) group. The work is based upon requests from the CERN and Deutsches Elektronen-Synchrotron (DESY) AFS administrators. I would like to thank my supervisors at CERN, Rainer Toebbicke and Bernard Antoine, for excellent feedback and support during my work; and Nilo Segura Chinchilla for helping me with performance issues related to the Oralce database.

I am most also most grateful to my supervisor at NTNU, Anne Cathrine Elster, for helping realize this thesis and the NOTUR poster related to this work.

Contents

List of Figures

1 Introduction

As HPC systems grow, the underlying file servers need to handle more and more data efficiently in order to maintain performance. This is particularly true for CERN's distributed file system, which will have to handle Terabytes of data. The goal of this project is to efficiently monitor and automate the CERN AFS (Andrew File System) file server information by storing the related information such as volume access, volume moves, used space, etc. in a database (DB). With the AFS server history provided by the DB, applications can then be used to automate decisions regarding how to move data volumes for load balancing the system. Thus, this can ensure that no file server is overloaded and hence unable to serve the AFS users.

A distributed file systems unites the file systems of individual file servers into a single name space. The underlying distributed nature is hidden. Files stored on each file server are as accessible to users as the files stored on the users' local disk. For instance, a group can share disks pace for their project, or single users can have their private accounts. Usually data is backed up on disk and tape, making the file system more reliable than the local disk. Universities and large organizations among others use distributed file systems for sharing common data and centralize data management, increasing system availability and system efficiency. Frequently used software can be distributed over several file servers in order to avoid that a server crash result in unavailable programs. A distributed file system also distributes the workload over several file server machines, which tends to be more utilized than larger file storage machines of a centralized file system. For instance, if a file server is overloaded, some volumes[1]may be moved to other servers that have less work load.

The CERN AFS file system houses a large amount of clients and important projects, among the ATLAS[2] and CMS[3] projects. The data management at present time is complex and is a mix of human and computer processes. Data statistics for monitoring the file system are located in files and not centralized, which make the system administration hard to comprehend. A challenge is to distribute volumes over the file servers in order to improve system performance, which is the main focus of this thesis. At present time volumes are only moved when a partition or a file server is approaching overload. With a volume history it would be possible to create a more intelligent system for load balancing the volumes before a server or a partition reach a critical level and even make the system more efficient.

In agreement with DESY (Deutche Elektronen-Synchrotron)[4] CERN decided to go for a database approach for storing volume history. This will eventually not only result in more intelligent volume moves, but make AFS administration more effective. For AFS administration, volume information data is more available, and less costly to retrieve, from a DB than from the file servers.

[1] A "container" of related files and directories, usually belongs to a project or a single user.

[2] A particle physics experiment for the large hadron collider, LHC, which will explore the fundamental nature of matter and the basic forces that shape the universe. The ATLAS detector will search for new discoveries in the head-on collisions of protons of extraordinarily high-energy.

[3] The compact Muon Solenoid - A detector that validates ATLAS's results and vice versa

[4] CERN and DESY have a similar structure and share the same challenges in AFS management

The work approach is to design and implement the applications and DB for storing and analyzing volume history, in addition to an interface for tuning automation of load balancing. The history for existing volume distribution and moves has to be analyzed in order to locate which variables and history that are more important for optimizing the system. To relocate all volumes for optimizing is not an option due to the size of the distributed file system at CERN, which means that automation of volume moves has to be tuned and optimized over time. Another issue is the user pattern for each volume. At recent time CERN is in a install period of the LHC[5] project. In year 2007 the LHC project will go into a experiment period, which could radically change the usage of the AFS file system. In fact, the CERN AFS cell is expected to increase with a factor of 10 during a three years period. This work will eventually be submitted to the AFS open source project[3].

[5]Large Hadron Collider - An accelerator which brings protons and ions into head-on collisions for recreating the conditions after the Big Bang in order to find (yet) undetected particles

2 Background

As far as management is concerned the AFS is a cluster. File server machines are updated and managed with scripts. Technically they are not in the same physical location nor do they communicate directly with each other. They are only aware of the DB servers that know where the files are located and are not of each other. Cluster file systems (e.g. IBM Storage Tank) and cluster are more static with identical nodes. If one node is unplugged the systems fails. AFS is more dynamic. If one node is going down the system still runs. The nature of physical science requires this kind of HPC cluster than the traditional form of parallel programs running on a cluster. The same program is usually run several times with different values in order to find the one most close to the nature. Parallel programs that are running on several CPUs, interfacing with each other on runtime, is not very common for physicists. Thus, a reliable and fast transfer from the AFS file system to several nodes requesting the same application is necessary to serve the local AFS users in the CERN cell.

This chapter provides a background by looking into the basic concepts of AFS, advantages and common issues for using a distributed file system. First, the concepts of AFS are presented in chapter 2.1, describing AFS and a distributed file system. Chapter 2.2 describes the phenomenon of load balancing.

2.1 AFS - Basic concepts

AFS is a client-server architecture distributed file system, providing location independence, scalability and transport migration [1]. It allows users to share and access all of the files stored in a network of computers in the same way as on a local file system. The files are distributed on several servers, but are available from every client. A Distributed filesystem's main advantages are increased system availability and increased system efficiency.

Cell A *cell* is an independently administrated site running AFS. The cern.ch cell is such an example. The cell's administrators determine how client machines are configured and how much space is given to each user or group. A cell consists of a set of file servers and client computers. A set of computers can only belong to one cell. Two cells cannot share a client or a file server. A cell's file tree looks the same when viewed from any client because the cell's file server machines store the files centrally.

Clients Clients are the working stations in the AFS system. They are using caching in order to increase the speed and efficiency of the file system. Each AFS client computer dedicates a portion of its local disk or memory to cache where it stores data temporarily. The cache-manager communicates with the file server and make sure that the local version is up to date. This is called *callback*. Working on a RW volume on a partition requires more call backs than a read only volume, which will be discussed later. If several clients work on the same file (if they are on the same project and have

the same permissions to edit data), the person that saves last makes the changes in traditionally UNIX style.

File servers The file servers actually stores and manage the data in the AFS system. File servers provide file storage and delivery services in addition to other specialized services for the client computers. There are several file server processes, some of them running on all file server machines while other are more specialized. For instance, every file server machine has BOS, basic-over-seen server, which monitor the server and restarts a process if it fails. The binary distributed machines keep track of the most popular binary files in order to save the Read/Write partitions for unnecessary accesses and database servers keep track of the physical location of volumes.

Server processes

- *The file server process:* The file server is the most fundamental of the AFS processes and run on each file server machine. It delivers the same service across a network as a local file system on a UNIX computer. Furthermore, it delivers data files and programs on the clients demand and stores edited files, maintaining the directory structure, handling requests for copying, moving, creating and deleting directories and files, stores the status information for each file and directory, controls authorization and creates symbolic and hard links between files and grants advisory locks on request.
- *The Basic OverSeer Server process:* Runs on each server and reduces the demands on system administrators by monitoring the running processes. It can automatically restart failed processes and provides a interface for administration tasks.
- *The authentication server process:* Performs network security related functions. It identify users as they log into the system and requires a password. It also help client and server processes to prove their identity for each other. Kerberos[6] is used for these purposes.
- *The protection server process*: Protects files and directories from unauthorized use. Rather than using UNIX file system's three access permission AFS is using seven for making the system more flexible. Users can grand individual users to have access to their directories using a Access Control List (ACL). ACL is a list of all the groups to which the users belongs. It is possible for administrators to create groups where permission is granted for certain IP addresses in addition to the ACL lists.
- The Volume Server *process*: Operates at the level of whole volumes, provides the interface trough creating, deleting, moving and replicating volumes as well as preparing them to be stored on tape.

[6] A network authentication protocol developed at Massachusetts Institute of Technology

- *The Volume Location, VL, Server process:* Maintains a complete list of volume locations in the Volume Location Database (VLBD). When a client's cache manager requests for a file, it first contact the VL Server in order to localize which file server has the wanted volume and file.
- *The update server process:* Updates AFS file server process software and configurations on all file server machines.
- *The backup server process:* Maintains the information in the backup database. The backup server process and the backup Database allow administrators to back up data from AFS volumes to tape and restore it from tape if necessary. The backup approach is to first clone a read/write volume and protect it from HW failure by taping it.
- *The salvager process:* Runs only after the failure of a file server or volume server process and restores any inconsistencies.

Partitions A partition in the AFS file system environment is usually one of several disks on a file server machine. Partitions are divided into volumes.

Volumes Volumes are conceptual containers for a set of related files, which is keeping them togheter on one file server partition. Volumes vary in size, but are smaller than a partition for practical and administration reasons. The volumes can be moved between file servers in order to maintain system efficiency. A volume corresponds logically to a directory in the global file tree. AFS interprets mount points, directories in the file tree, and retrieves the files and directories that the user request from the appropriate location. In order to find a file, only the file's parent directory and the file name is required. The system uses the *Volume Location VL, Server process* (VLDB) in order to keep track of the volume's file server and partition location. Most cells store the contents of each user's home directory in separate volumes. The complete contents of the directory moves together when the volumes moves, making it easy for AFS to keep track of where a file is at a certain time. Volume moves are recorded automatically in the databases, so users do not have to keep track of file locations.

Set of volumes A set of volumes are volumes that belongs to each other. Usually a set of volumes is a project, created by the project administrator.

Volume quota A volume has an upper limit for available space. The quota can be increased by the AFS administrators if more space is required.

Volume used blocks Used volume blocks is the number of blocks used by the volume. This number can never be larger than the volume quota.

Mount points A volume corresponds logically to a directory, or the root directory for that volume. A directory in the file system is mounted to the certain volume using a on-line file that names the volume containing the data files in the directory.

Read/write volumes Read/Write (RW) volumes are the original volumes that can be read from and written to. A RW volume can be backup up on a backup-volume, always located on the same partition as the original RW volume. Furthermore, a volume can be replicated into several clones. The RW volumes have a unique ID number, which is used by the VLBD for identification.

Replications Read only copies or clones of volumes usually contain frequently accessed binaries/programs. The replicated volumes are copied to other disks/partitions to relieve a file server machine or a disk for accesses, which makes the contents more available. On large systems usually one server is dedicated to this purpose. Using files on read only volumes puts less of loads on a file server, cause only one call back is required each read only (RO) volume as opposed to a call back each file for the original volumes that support read/write (RW). If a set of replicated volumes are copies of one RW volume, they have the same ID number. Hence, RO volumes are identified by ID number and location.

Backup volumes Some volumes in a distributed file system are backed up to increase reliability. All changes are copied into a backup volume located on the same partition every night. Furthermore, backup volumes (BK volumes) can be backed up on tapes. BK volumes are always located on the same partition as the original (RW) volume. The backup volume's ID is unique because there are maximum one backup volume each RW volume.

Caching and Callbacks Caching on the clients increase the speed and efficiency of file accesses in AFS. Clients use local disk or/and memory for caching data. A client cache manager handles the file requests and use the local files if they are present and up to date. Caching reduces network traffic and access to frequently accessed data. *Callbacks* are used in order to maintain consistency for cached files. A File server is calling back to the client if any of the requested files are modified. For RW volumes a call back each file is required, thus only one is required for replicated volumes (RO volumes). Any RW file can be modified at any time, while a RO volume is up-to-date until it is deleted and replaced. In AFS, all replications of a certain RW volume have the same ID.

Transparent access and Uniform name space AFS has *transparent access* to the files in a cell's file space. The users do not need to know which file server or servers their files are located in order to access them. They only need to know the file's pathname, which is by the AFS system translated into a machine location. *Uniform name*

space: A file's pathname is identical regardless of which client machine the user is working on.

Cloning The *volume server* use cloning of RW volumes for moving volumes, creating backup and RO volumes. A clone in these terms is not a copy of the volume data, but a copy of the *vnode index*. The vnode index is a table of pointers between the directories and the files in a volume and the physical data location on the disks.

Volume Moves When a volume is moved between two partitions, it is first cloned and then copied over the network. During the process of cloning it is unavailable for I/O requests, but not during the process of copying. All modifications during the process of copying are added to the new volume after the volume move is completed.

2.2 Load Balancing

Load balancing distributes processing and communications activity evenly across the file server network and is important for making the system efficiency. When a server is overloaded by accesses or empty of disk space it is necessary to move one volume from one file server to another. Moving a volume make the volume unavailable during the cloning process and may take up to several hours depending on the volume size. However, during the moving process the volume is again available. All changes during the moving process are added after the move is accomplished.

As described in chapter 2.1, the following statements are true for file servers, partitions and volumes:

- A file server machine has one or several disks. The number of disks each file server differs
- A partition on a file server machine is usually a disk
- A volume is never larger than a partition, e.g. a volume is not striped over several disks
- A backup volume is always on the same partition as the RW volume

The disks on the file server machines are potential bottlenecks. If a disk reaches the maximum amount of accesses that the file server can handle, all other disks on that file server is unavailable. Thus, load balancing is performed on disk level. Figure 1 demonstrates an ideal volume distribution over three file servers, given that they have the same amount of disks (two in this example), the same HW in general and volumes have the same size. This setup exploits the storage space available and distributes frequent accessed volumes, avoiding that requests are piling up on one file server and, hence, make the system less available.

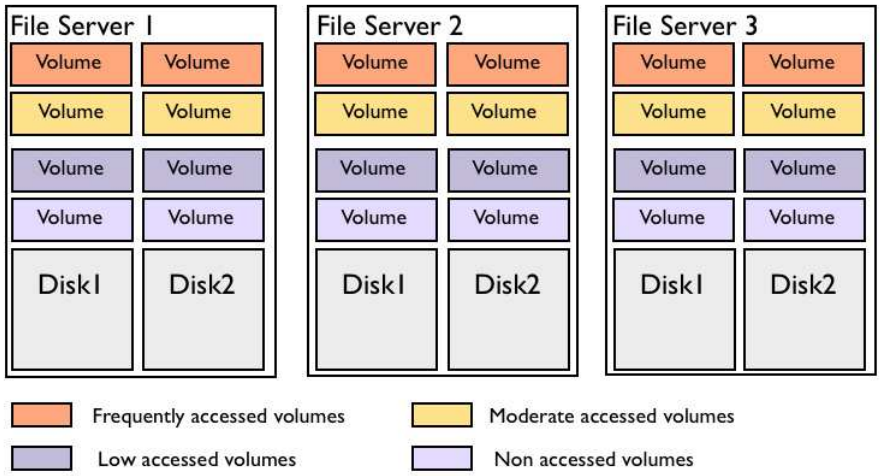

Figure 1: *Ideal volume distribution for the file server system*

In the reality such ideal distribution is difficult. However, there are other issues than only load balancing that have influence on volume distribution. For instance, some volumes may have higher priorities than others, requiring better response time etc., or volumes belonging to a set of volumes could be gathered on few file servers in order to decrease network traffic. The latter is true because users of a project do not need to access two different servers in order to retrieve or store data if all project volumes are located on one server[7]. Furthermore, clustering sets of volumes is beneficial in case of a server crash. In this case only a few projects are influenced by the event, which ease administration.

Volumes differ in size and accesses each volume differs over time. In fact, volume size can change over time as well. Some volumes may be heavily accessed at present time, but not at all in close future. Thus, load balancing on present data information may have serious consequences. Without a history it is impossible to fulfill intelligent automated moves, exception of when disk storage exhaustion occurs. If a disk or server is full, volumes are stored elsewhere. However, most of the load balancing is done manually in AFS cells at current time.

Figure 2 demonstrates a situation where Disk1 houses a volume FS1_D1 (File server 1 - Disk1) that has a frequent number of accesses. In total the volumes on File Server 1/Disk1 reach the upper limit of accesses the server can handle, which leave volume FS1_D2 unavailable for its users. In this case AFS administrators have to manually move the upper FS1_D1 volume to another server, in this case File Server 2. The most suited partition is Disk1, with enough space and no frequently accessed volumes. Hav-

[7] From the AFS administration point of view. The users do not know the physical location of the volumes due to transparent access in AFS

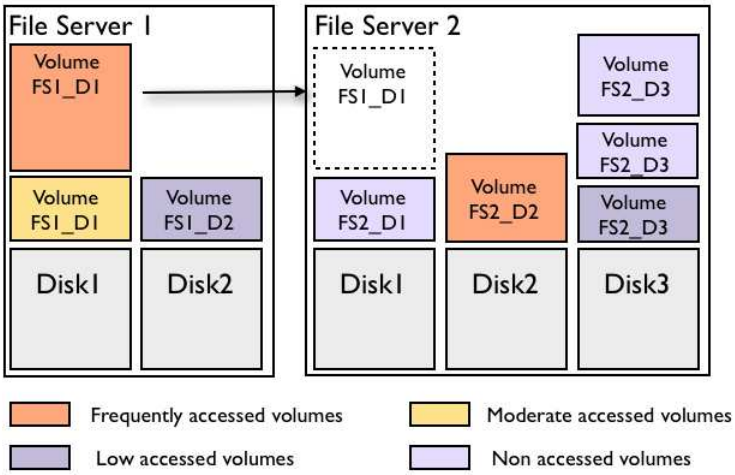

Figure 2: *Realistic volume distribution*

ing more hard disks/spindles, file server 2 can handle more accesses, but it is necessary to take FS2_D2 and FS2_D3 into consideration before moving the volume. There are no point in moving a volume to relieve a server only to exhaust another one. At present time such volume moves are usually performed when a file server reach its upper limit. As already mentioned, intelligent moves are impossible without knowledge of the access pattern for a volume over time. If a volume history could confirm that the FS2_D3 volumes over time stay on a low and none access level in average, it would have been preferable to move one of them to another server (for instance file server 1 / disk 1 which has freed up space), making space for (a) more accessed volume(s).

To find how available a partition is, it is useful to use a metric which calculates total volume accesses and used diskspace for a disk. Thus,

$$DiskAvailability = \frac{TotaAccesses}{GB}$$

An optimized solution for volume distribution is a NP-Complete problem. Given a set of items, each with a cost and value, the number of each item is determined to be included in a collection so that total costs is less than some given costs and the total value is as large as possible. Thus, having a set of volumes and partitions, how can the volumes be distributed over the partitions in order to exploit the storage space and handle accesses in a optimized fashion? A solution is to calculate every possible volume combination on every partition, which is an exponential time growing problem. However, given the size of the distributed file system and additional conditions, a polynomial time approximation algorithm is preferred for this project. [ref coreman]

2.3 Oracle Database

(About Oracle advantages and disadvantages)

The CERN central database service has Oracle databases in production. Exploiting a supported service will ease AFS administration if the database fails, making Oracle a natural choice. Nevertheless, the SW design should be independent of database vendors considering the open source nature of AFS. Several Oracle features like PL/SQL[8] are therefore not exploited.

2.4 Perl

Perl was initially designed as a glu language for Unix, and has long since spread to most other operating systems. Hence, the language is available on several programming environments. Also, no changes are required when moving scripts between different OSs. This ability makes Perl suited for AFS development, considering that distributed file systems usually house different kind of operating systems and architectures. For instance, at CERN the AFS file servers run Linux and Solaris on IBM and SUN machines. Scripts written for the AFS file servers do not need to take the underlying platform into consideration. Nevertheless, Perl is used by AFS developers and is a natural choice for further development[6].

2.5 Perl and Oracle

Some components are required in order to make Perl and Oracle modules communicate efficiently: Perl DBI, DBD:Oracle and OCI[4] (See Figure 3) Perl DBI is a generic application-programming interface (API), similar in concept to Java Database Connectivity (JDBC). Perl object oriented archtecture makes it possible to have a single routing point to many different DBs, which is another argument for using Perl in a Open Source environment where several DBs should be supported. Oracle uses the DBD:Oracle driver, a Perl module that provides the actual communication to the low-level OCI code. The OCI component makes the final connection the Oracle DB.

[8] Procedural Language/SQL.

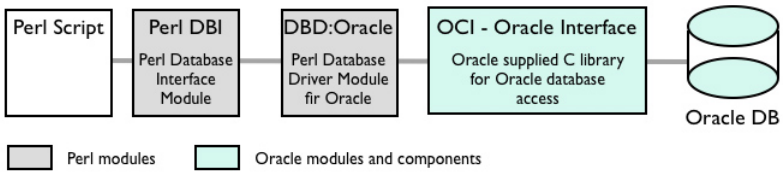

Figure 3: *The Perl/Oracle architecture*

3 State Of The Art

This chapter discusses the CERN AFS cell's setup and dimensions. The following chapter 3.1 discusses how AFS is used at CERN, how data is presented and how decisions are made in order to manage the system efficiently. Chapter 3.2 discusses AFS admin, which is a supplement to AFS, implemented at CERN to ease administration. Chapter 3.3 discusses how load balancing can be improved at CERN.

3.1 The CERN AFS Cell

Several analyses are requires to apprehend the situation in the CERN AFS cell. Management, hierarchy, dimensions and even extensions to the AFS are crucial issues for analyzing and, hence, tuning the load balancing using a DB.

Management hierarchy has significance influence on system efficiency. For instance, replicas or read only volumes have potential importance for system performance, decreasing network traffic and callbacks, increasing responstime and availability. A disadvantage is that local project administrators need to create these replicas manually, which unfortunately results in few read only volumes due to more management overhead.

The CERN AFS cell tends to favor, at present time, relatively less expensive storage disks that are piling up requests, rather than several smaller disks that can handle requests faster. In order to maintain the AFS service at present level during the growth of file servers, users and projects approaching the LHC startup in 2007, there will be installed a larger amount of smaller disks, which makes the system more available due to more spindles. Hence, the disk subsystem is a potential bottleneck. If the number of waiting I/O requests is a sustained value more than approximate 2 times the number of spindles making up the physical disk, there is a disk bottleneck.

Some accesses are more costly than others. In the CERN cell volume changes are backed up on tape every night. These accesses, which are streaming data from disks to tape, can exhaust the disk if they are running simultaneously. Thus, two processes are competing for the disk arm. Some intelligent code would be preferable in order to

detect such situations. However, tuning on this level is not a part of this project. What is worthy to notice is that such effects may have influence on the file server machine monitoring and history.

Another important issue is volume size. A system with small volumes and few files is ideal for optimizing performance. It is faster and less critical to move volumes that are smaller. However, moving large volumes is costly and need to be carefully planned. On the other hand, for project administrators it is more convenient to create large volumes to ease administration, which complicates optimization.

It is preferred to distribute a set of volumes over the file server machines in order to avoid that all of them are involved if a server fails. Such distribution is increasing the system availability for projects, but can easily conflict with volume distribution for increased overall system performance.

These issues are important in order to understand how to analyze AFS history properly and tune load balancing. It is preferred to find approximation algorithm that can take these variables into consideration in order to find a general approach for load balancing a distributed file system.

3.1.1 Basics

A look into the AFS term is preferable in order to analyze the AFS setup.

The AFS hierarchy At CERN the AFS system administrators decide the volumes' physical location for making the AFS system effective. They decide every users' and project's quota. However, the project leader decides how they are exploiting the given space. Several of these decisions are straightforward and are automated. The project leaders decide the size of their volumes (within physical limits) and if they are using replication or not. These local decisions makes it even harder to manage the system administration and load balancing for the AFS administrators. The project leaders decide as well which users are granted, using protection groups. This modification is at present time used at CERN, avoiding that the ACL files are not updated and ease administration work. With protection groups a project leader have more options for granting users, even giving his or her administrator status away to another user if preferable, without interfacing with the AFS administration. Hence, the hierarchy is:

- *The AFS administrators*. At present time there are three AFS administrators at CERN. Their tasks are to load balancing the system for increased performance, deciding disk quota for users and projects, installing new file servers, maintain code and support the AFS system.
- *The project leaders*. There are approximately 5000 project leaders at CERN. They exploit the disk quota given by the AFS administrators[9]. Users granted for

[9]The disk quota is reflecting the size of a project. If a projects need more space there are financial issues involved.

projects volumes, number of volumes, directory tree among other tasks is as well done by the project administrator.

- *Users*. Users have their own accounts, independent of which projects they are granted to, if any.

Dimensions At the CERN AFS cell there are approximately 15000 users, 5000 projects, 8000 client machines and 30 servers for the time being. There are 4000 active users at any time. It is necessary to carefully decide required storage space in order to save historical statistics in a database. The toal disk capacity is 12 Terabytes, which is approximately 410 GB each server. Over a period of 3 years disk capacity is expected to increase to approximately 100 Terabytes. The typicall file servers at current time are SunFire V240 and IBM xSeries models 345 and 346.

Volumes Small volumes make load balancing more straightforward and less time consuming to move volumes between file servers if neccessary. An idealistic volume size for the CERN AFS cell, considering the dimensions for the time being, is 1 GB with 100 large files. Unfortunately, the project leaders using AFS usually prefer large Volumes. Volumes are mounted into the directory hierarchy and several volumes means more overhead and management work. It is more convenient to use few and large volumes than several small ones. Volume size is a tradeoff between the AFS administrators and the local project leaders wishes, regarding that the administrators simply decide the total quota to each project. The project leaders decide the volume partitions in the end. Consequently, moving a volume could be time consuming and costly, maybe several hours, and an eventually move has to be carefully evaluated.

Replications Using replications for reducing call backs and network traffic is unfortunately not a common task. Replications are not transparent and has to be mounted and managed by the local project leaders. Knowledge about AFS and replications are required and is used only by a few projects at CERN. Network traffic is neither the main performance bottleneck.

Database servers There are 3 main database servers at CERN and 2 for backup. The 3 main servers are located in the same building and need backup in case of fire. This is necessary because the database servers are not backed up on tape.

Backup server The backup server backup the data on HDs every evening, where all changes are stored on tapes. Every 30 day the data is fully taped from scratch. Every user and project have direct access to their backed up data. The backup is not a part of their quota. The backup on tape is done by the CASTOR[10] project. CASTOR is a hierarchical storage management system developed at CERN.

[10]CERN Advanced STORage Manager

P, Q, asis and U volumes CERN operates with different kinds of volumes. p and q are on more reliable HW than u. p volumes are backed up while q volumes are not. Asis are volumes with loads of public domain software. The sowftare is not installed on the machine, but rather directly accessed through AFS via symlinks.

3.2 AFS Admin

AFS Admin is a supplement to AFS, implemented at CERN by Wolfgang Frieble from DESY. The purpose of AFS Admin is to ease administration, using routines that are meant to enhance the functionality of the AFS module. The routines retrieve volume, server and partition status information data among others, e,g accesses, total disk usage, volume names, volume location etcc. AFS Admin makes it as well more convenient for project administrators to create, delete and grant users. When administration requests status information, AFS admin interfaces with the file server machines in order retrieve the data. For instance, retrieving data for volume with ID number 537031140 using vos examine in a console results in:

 # vos examine 537031140

> p.lep.higgs.general1 133588940 RW 1335889 K On-line

> afs45.cern.ch /vicepa

> RWrite 537031140 ROnly 0 Backup 537031142

> MaxQuota 1500000 K

> Creation Wed May 26 12:50:08 1999

> Last Updated Fri Feb 18 15:20:34 2000

> 0 accesses in the past day

This approach, accessing the file servers for each AFS admin request, may compete with the AFS users' accesses and is not preferable. Storing volume information in a DB will make AFS statistics more available for administrators and relieves the file server machines. The design and implementation for a more effective AFS Admin is discussed in Chapter 4.

3.3 Load balancing

The AFS administration at CERN prefer to cluster the sets of volumes. If a new volume is created it is stored on one of the file servers that all ready house the volum's set. Among the file servers that house the set, the volume is stored on the file server that has most disk space available. Only when the file server(s) that house a set of volumes are full, a new file server is chosen. (The available file server with most available space. See Figure 4. File server 1 houses the CVS and SUN volume sets. File server 2 and 3 house the ATLAS and CMS volume sets. File server 1 is full and the new SUN volume is stored on the file server that has most disk space available, which in this

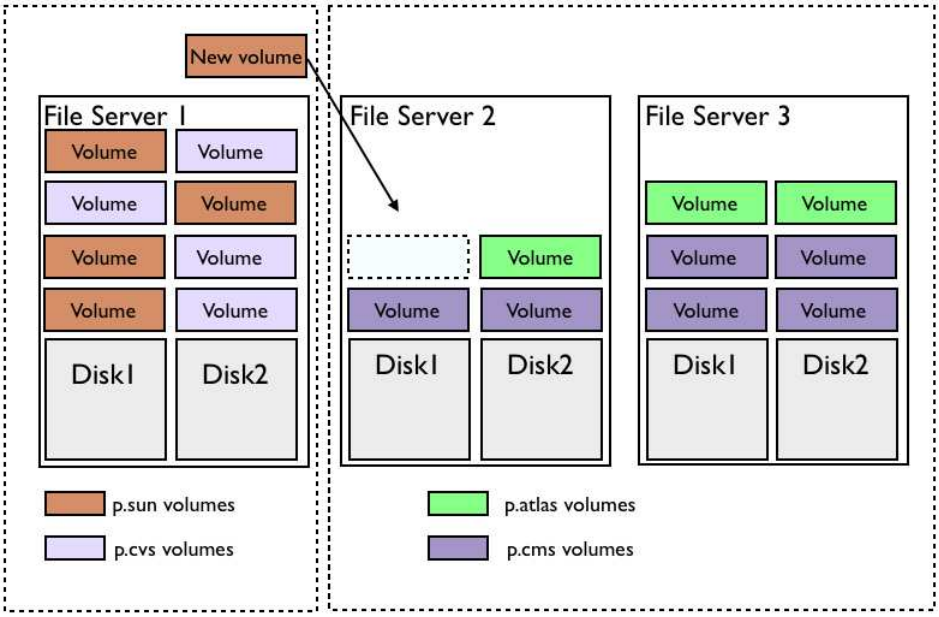

Figure 4: *Load balancing*

example is file server 2. When this happens, File server 1 and file server 2 house the SUN volumes. File server(s) to house a set of volumes is manually chosen when a new project is created. At present time, the only automated load balancing is volume moves if volumes on a partition are increased in size and approach the disk space limit.

This volume clustering may conflict with the optimized load balancing mentioned in chapter 2 due to the decreased number of file server candidates to store new volumes. Nevertheless, this approach is chosen for administration and network traffic in mind. If one volume in a set (a project) is unavailable, usually the whole project is influenced anyway. It is preferable that a whole project is unavailable until the server is restored, in contrast to a scenario where several projects loose some of their volumes.

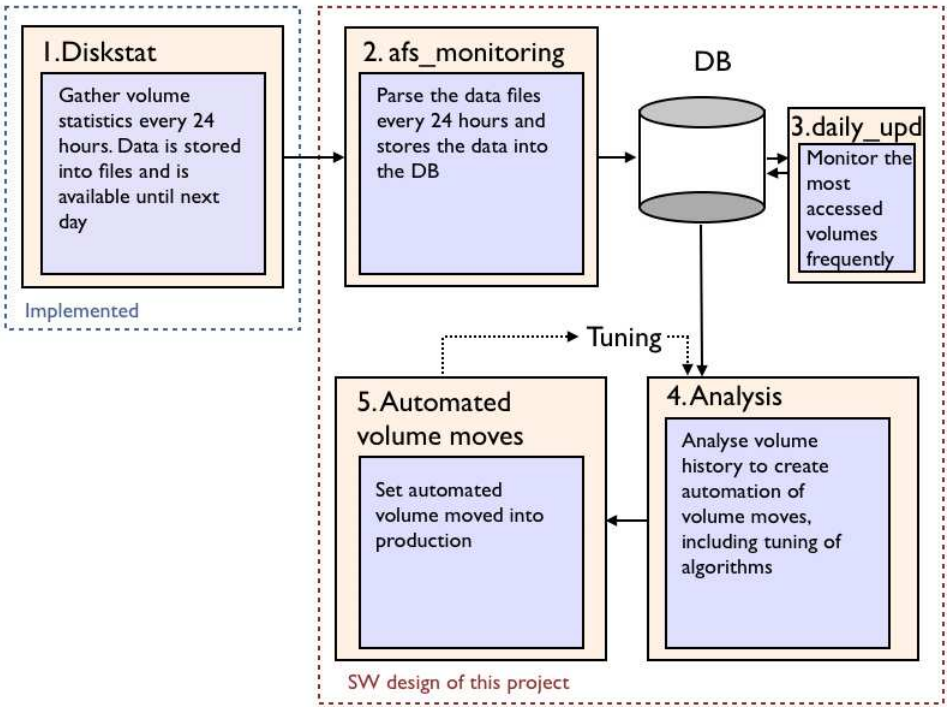

Figure 5: *Load balancing and tuning using a DB.*

4 SW Design

An underlying system was designed and implemented during the work of this thesis. First, a DB was designed and tested before it was set into production. Perl scripts were implemented in order to update the DB with data from the underlying AFS system. Because the DB will be used by independent projects a interface was created, for retrieving, updating and adding data. However, the main purpose of the DB is to partly automate effective load balancing. A graphical user interface was implemented in order to analyze volume history, and thus, better understand the volumes' behavior. The SW design for monitoring and tuning load balancing in the CERN AFS cell is demonstrated in figure 5.

The SW design for monitoring and tuning load balancing depends on already implemented applications and interfaces like Diskstat and the Vos.pm and Volset.pm packages, described in chapter 3.

1. The Diskstat script gather volume data information every 24 hours from the file servers and write the results to a file AFS.snapshot. This file is at present time used to present the file servers' status in terms of disk usage, accesses, volume

quota and so on. Because AFS.snapshot is overwritten every 24 hours history data is lost.

2. The afs_monitoring script parses and stores the data in AFS.snapshot every 24 hours to the DB, keeping the volume history. In addition to store this data it detects which volumes are most accessed on each file server and stores the volume IDs in a DB table. The most accessed volume IDs are of use for the daily_updates script.

3. Despite intelligent volume placement a file server machine may be overloaded during the day. The daily_update script connects to the DB and detects which volumes are most accessed on each file server the most recent day, due to the fact that these volumes are potential candidates for overloading the file server during the day. The script stores volume data statistics for the given volumes into the DB more frequent than afs_monitoring, making the system (Automated volume moves) able to move off heavily accessed volumes when overloading occurs (see figure 3, chapter 3.1). The script is executed several times a day and use the Vos.pm package to retrieve the vaild data from the file servers. Only a given number of volumes are frequently monitored during the day to not put unnecessary load on the file servers.

4. Analysis of DB data for intelligent volume moves. Volume history data is analyzed in order to find some connections between the variables for automate load balancing. The most obvious optimization gains are automated first, then the script will be tuned over time. Hence, there will be an underlying approximation algorithm. Inital analyses are included in this thesis and will be continued in further work.

5. The script will be tested on the AFS system and will be analyzed in order to improve the approximation algorithm for better load balancing. This is not a part of this thesis, but will be implemented in further work.

Most of the information stored in the DB is updated every 24 hours, which is a costly operation. To avoid overloading the file servers during monitoring of the AFS system, only the ten highest accessed volumes from the previous day, which are likely to be potential problems, are monitored more frequently for load balancing the system. This is a tradeoff between reliability and performance. DB and table design is covered in chapter 4.1, script design for updating the DB in chapter 4.2, interfacing with the DB in chapter 4.3, graphical representation of volume history in 4.4 and finally the AFS Admin in 4.5.

4.1 Database Design

At CERN central DB services are provided by the Database and Engineering group, which use Oracle databases. Due to the DB size and administration issues this service

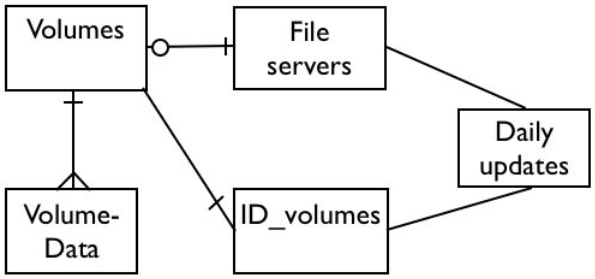

Figure 6: *DB ER-diagram*

was chosen for serving the DB. Because this monitoring approach will be set in production elsewhere the DB design had to be independent of the underlying DB architecture. Nevertheless, some optimization for Oracle is described in chapter 4.1.2

4.1.1 Data distribution in DB

An Oracle DB stores the AFS file system data history, such as volume access, volume moves, used space etc. Normal forms are not preferable in this case. Data tables are divided into the types of data that are most frequently updated in order to avoid unnecessary data replication. Some Volume information is seldom updated, once a week or once a month. Other data is updated more frequently. The basic idea is to timestamp data if there are no modifications (The previous updated data is also valid in the most recent update). If there are any modifications new data is inserted. In this way it is possible to create time boundaries for which period any given data is valid. Thus, volume ID and date is required to retrieve the volume history data from the DB. Because replicas of any RW volume share the same volume ID, server and partition location is also required for RO volumes.

There are 5 tables: *File servers, id_volumes, volumes, volume_data* and *daily_updates*. The tables *Fileservers* and *id_volume* keep the relation between file server and volume ID number and name. The ID numbers are used as keys in the tables volumes and volume_data. Table *Volumes* stores less frequently the volume data while table *volume_data* stores more frequently updated data. There is a one to many relationship between the tables *volumes* and *volume_data*. For one row in table *volumes* there can be several updates for the volume's frequently updated data. Table *daily_updates* keeps track of which volumes are to be monitored every hour for automation of volume moves. See figure 6 for a ER diagram.

Fileservers

id_server	name
1	afs45
2	afs46
...	...

(a)

Figure 7: *Table Fileserver*

id_volumes

id_vol	name	set_id
537045149	p.aleph.wwtf	lep
537045151	p.aleph.wwft.back up	lep
.....	.....	.....

Figure 8: *Table id_volumes*

Table Fileserver and id_volume

Table *fileserver (figure 7)* identify ID_SERVER used in DB with its real name. Table *id_volumes (figure 8)* identifies volume ID, ID_VOL, with volume name.

Table volumes:

The data in *table volume (figure 9 and figure 10)* is volume data which is not updated frequently. For each daily update the script determines if there are any differences between volume data in the DB and the AFS.snapshot file[11]. If there are no edited fields for a given volume, for the fields ID_SERVER, ID_PARTITION, SET_ID, ID_REP

[11] A file updated every 24 hour with file server information

Volumes

id_vol	id_server	id_partition	date_add2db	date_updated
537031140	1	a	10.03.2005:13.39:05	12.03.2005:14:28:22
537031142	1	a	10.03.2005:13.39:05	12.03.2005:14:28:22
...	..	..	...	...

Figure 9: *Table volumes part 1*

or ID_BACKUP, DATE_UPDATED is updated with todays date. No further data is edited. If there are any changes a new row is created, with `RECENT_ROW` identifying that this is the most recent row for the given volume ID. Because selecting with date comparison is costly, the field RECENT_`ROW` is set to 1 for the most recent row for a given volume ID. This is done in order to make select statements more effective in retrieving the most recent volume data. The most recent data is likely to be the most interesting information and should be more accessible than older data. Older rows for the given volume are not updated anymore with timestamps and RECENT_`ROW` is set to 0.

In order to keep the DB at a given size, old volume data is compressed. For instance, all volume data that is between one and two months old can be stored as one (or few) row(s) with the average field values. Because id_server and id_partition can't have average values, for storing volume moves history, there can only be average values for between volume moves.

If a volume is a RW volume, it can have a replica (RO) volume ID and a backup volume IDs. The AFS file server system allows one backup volume and several replicas for a RW volume. Thus, a replica or a backup volume can't have id_rep or id_backup volume IDs in the DB. The RO and backup volumes' parent is the RW volume. Because there can be more than one volume existing on the file servers with the same ID (several RO volumes), DATE_DELETED has to be in table *volumes* and not in the table *id_volumes*. Field DATE_DELETED stores the date when a volume is deleted from the AFS file server system.

Table volume_data

The data in table *volume_data* (figure 11 and figure 12) is updated frequently. It has the same mechanism as table volumes. In order to keep a secondary key for identifying which row(s) belongs to any given row in table volumes, this table is always updated at

id_rep	id_backup	date_created	date_deleted	recent_row	freq_upd
NA	537031142	18-FEB-2000	NA	1	0
NA	NA	13-FEB-2005	NA	0	0
..	..	...	...	...	...

Figure 10: *Table volumes part 2*

Volume_data

id_vol	id_server	id_partition	date_add2db	date_updated
537031140	1	a	12.03.2005:14:28:22	12.03.2005:15:28:22
537031142	1	a	12.03.2005:14:28:22	12.03.2005:15:28:22
...	..	..	...	...

Figure 11: *Table volume_data part 1*

the same time as volumes. Because this data usually is edited more often, several rows are likely to “belong to” one row in table volumes.

Daily updates

Table daily_updates (figure 13) stores which volumes are to be frequently updated during the day.

4.2 Updating the Database

There are two scripts that updates the database. afs_monitoring, discussed in chapter 4.2.1, updates the DB every 24 hours (at midight). Daily_updates, discussed in chapter 4.2.2, monitors the 10 most accessed volumes on each server.

blocks_quota	blocks_used	accesses_int	accesses_ext	file_count	recent_row	freq_upd
300.000	51028	1680	0	100	1	0
100.000	30034	886	0	1237	1	0
..	..	...				

Figure 12: *Table volume_data part 2*

Daily_updates

id_server	id_partition	id_vol
1	a	537031140
..	..	..

Figure 13: *Table daily_updates*

4.2.1 afs_monitoring

The afs_monitoring script stores volume data from the file AFS.snapshot every 24 hours. The abstract approach is:

1. Compares volume data information between the AFS.snapshot file and the DB and updates all changes.
2. Detect and mark deleted volumes
3. Detect and store the most accessed volumes IDs into a table

In order to update the DB, the script compares the updated file information in AFS.snapshot with the DB. If the script finds the same volumes in the DB, it updates its information. If there is no new information to be updated it timestamps the most recent volume information with todays date. If there is any new information for the given volume, it inserts a new row with the recent data from the file.

Update table fileserver If there any new file servers added to the system, the server name is added to table *filservers*. As well, the file server is given a numeric ID to decrease search time.

Update table volume_id If there are any new volumes added, the volume ID and name is added to table *volume_id*.

Updating table volumes Because there is only one RW volume ID and one backup volume ID in the file server system, the IDs works as unique identifications for these volumes. However, clones or replicas of a RW volume share the same volume ID. Thus, location volume location is required in addition to volume ID order to create a unique key for quering the DB. Because there are several rows due to history with the same volume ID in the DB, the final attribute for the unique key is recent_row='1', which locates the most recent row available for the given volume ID.

This eases the update process. Updates for replicas will be explained, where the volume ID is NOT unique.

The script first retrieves the variables from the updated file - AFS.snapshot:

```
$VOL_ID_FILE, $SERVER_FILE, $PARTITION_FILE, $VOLUME_NAME_FILE,
$SET_FILE, $QUOTA_FILE, $BLOCKS_FILE, $ACC_FILE, $DATE_FILE
```

where *vol_id* is a numeric ID for the volume, *server/partition* is the location of the volume, *volume_name* is the char name, *set* is the name of the set which the volume belongs to, *quota* is the maximum disk space for the volume, *blocks* used is the disk space used by the volume, *acc* is the number of accesses since last update, *date* is the date when the volume was created.

The script compares the file information with *table volumes* in the DB:

```
SELECT SERVER, PARTITION, DATE_UPDATED,SET,ID_REP,ID_BACKUP
FROM VOLUME
WHERE ID_VOL=$VOL_ID_FILE, ID_SERVER=$SERVER_FILE,
ID_PARTITION=$PARTITION_FILE,RECENT_ROW='1'
```

RECENT_ROW is a flag that has the value 1 if this is the most recent row of data for a given volume. If a row for a given volume no longer is the most recent, the flag is set to 0, and the row is no longer updated. As explained in the last chapter, the valid time boundary for a row of data is given by the date it was inserted into the DB and the last update date.

If volume data is modified:

If the volume is present but any of the values SERVER, PARTITION, DATE_UPDATED or SET are not equal to the file variables, a new row is inserted into the table *volume*. If the select statement doesn't return any data the volume is entirely new and is inserted into the table.

A new row of a given volume is to be inserted into the DB with RECENT_ROW=1. Any other row with the same volume ID need to be updated with recent_row=0. This is most practical to do before inserting the new row:

```
UPDATE VOLUME
SET RECENT_ROW=0
WHERE ID_VOL=$VOL_ID_FILE
```

Then the new row can be inserted:

```
INSERT INTO VOLUMES
VALUES($VOL_ID_FILE, $SERVER_FILE,$PARTITION_FILE, $TODAYS_DATE,
$TODAYS_DATE,$SET_FILE,$ID_REP_FILE,$ID_BACKUP_FILE,$DATE_FILE,
$NULLDATE, RECENT_ROW='1')
```

Where $TODAYS_DATE is the date of today, $ID_REP_FILE is the volume ID of the volumes' replica and $ID_BACKUP_FILE is the volume ID of the volumes' backup volume (if they are present). $ID_REP_FILE and $ID_BACKUP_FILE are available through the vos.pm[12] package in the AFS system. $NULLDATE is defined as NULL until the volume is deleted from the AFS file servers.

[12] Perl interface to vos examine, vos listvol, vos listvldb and vos partinfo

Else - if data is NOT modified:

If the Volume exists in the DB and no variables are edited since last update, the script updates the most recent row for the given volume with a timestamp of today:

```
UPDATE VOLUMES
SET DATE_UPDATED=$TODAY
WHERE ID_VOL=$VOL_ID_FILE AND RECENT_ROW=1
```

NB! The same approach is carried out for the table *volume_data* with the identical timestamp $TODAYS_DATE.

Updating replicas:

The RW and backup updates can easily operate due to the fact that the volume ID is unique, but there can be several replicas of a RW volume. These RO volumes of a given RW volume share the same volume ID. Thus, replica volume identification requires the volume ID, the file server location and the partition location as explained. If no data for a replica has been modified since the last update, the same approach as for RW and backup volumes is used. The script finds the identical row in the DB and updates the timestamp, DATE_UPDATED. The problem occurs when a new replica is present or data is modified. The volume ID is not a unique key. Setting all RO volumes with the same ID to recent_row=0 would influence already up-to-date data for RO volumes on other file servers and partitions.

Because a replica cannot move from one partition or a server to another, a replica that is not updated can be marked deleted. Because of this, a replica is left with the flag RECENT_ROW=1 even though there is new information for the volume ID on other partitions.

Determine which volumes are deleted After the script has compared AFS.snapshot with the DB, it goes through the DB and finds volumes which are marked with RECENT_FLAG=1 and have an older date than the most recent update. These volumes are not updated, which means that they were not present in the AFS.snapshot. These volumes are marked with DATE_DELETED=$TODAY and RECENT_ROW='0'.

```
UPDATE volume
SET recent_row='0' date_deleted=$TODAY
WHERE recent_row='1' AND date_updated<$TODAY
```

RECENT_ROW is set to 0, even though this volume won't be updated again. For practical reasons RECENT_ROW=1 is reserved for existing volumes.

Determine which volumes to be frequently monitored Determines and store which volumes are most accessed for each file server machine. This script is used by daily_update for frequent monitoring. For each server the following query is executed:

```
SELECT id_volume FROM VOLUMES WHERE RECENT_ROW='1' AND ID_SERVER='<SERVER_NR>'
AND ROWNUM<10 SORT BY ACCESSES DESC

FOR EACH FETCH:

INSERT INTO DAILY_UPDATES VALUES (<COUNT_FETCH>, ID_VOLUME)
```

For each server the ID for the 10 most accessed volumes are stored into table daily_updates. The daily_update script retireves the ID for all these volumes, quering table daily_updates, when it is executed during the day.

4.2.2 Daily_updates

The script monitors the 10 most accessed volumes on each file server from the last daily update. This is carried out in order to decide if any of the volumes should be moved to another file server during the day. Thus, the script is important for the further work on automated load balancing.Table *daily_updates* is updated with the ID and position of the ten most accessed volumes from the previous day. These 10 volumes are monitored through Vos.pm package functions every hour. The DB works in a similar way for more frequent updates with the exception of accesses. $TODAY in the perl script support updates down to each second.

Because the access data for a volume is set to default (0) every 24 hours, more frequent DB updates for these volumes has to consider the delta value between two updates in order to get the number of accesses between two time stamps.

For instance, if the script wants to calculate total accesses for a given volume between 09.00 and 10.00 with values A and B, the result is:

Value access $09.00 \rightarrow 10.00 = \triangle(A,B)$

4.3 Interfacing with the Database

The Perl package *db_afs_admin.pm* was implemented for interfacing with the DB. It was beneficial to introduce a interface for the DB for several reasons:

- At this point there are three different known modules that are interfacing with the DB: AFS Admin, a Perl CGI script that represents partition and volume history. Furthermore, the scripts for automated load balancing will interface with the DB.

- AFS Admin is at current time upgraded by other developers that are exploiting the DB developed in this work. Thus, an interface for other AFS projects to interface with the DB will ease developement. A new user account for the DB and the package db_afs_admin.pm is everything that is needed for interfacing with the DB.

The package contains subrotuines with SQL queries for both retrieving, adding and updating the DB.

Nested SELECT statements are used to retrieve file server history. For instance, retrieving the most recent access data for partition A, server AFS45, is carried out the following way:

```
SELECT ACCESSES
FROM VOLUME_DATA
WHERE RECENT_ROW='1' AND ID_SERVER= ( SELECT ID_SERVER FROM
FILESERVER WHERE NAME='AFS45') AND ID_PARTITION='A'
```

This will return the the total number of the most recent volume accesses for partition afs45/a. Because the most recent data is marked with the flagg RECENT_ROW='1', current data is faster to retrieve from the DB, which is beneficial for AFS Admin and other modules that do not need volume history. Selecting older volume data is done with dates. For instance, retrieving all access data for a file server 2 days ago for partition afs45/a:

```
SELECT ACCESSES_INT
FROM VOLUME_DATA
WHERE DATE_ADD2DB<=($TODAY-2) AND DATE_UPDATED=>($TODAY-2)
AND ID_SERVER= ( SELECT ID_SERVER FROM FILESERVER WHERE NAME='AFS45')
AND ID_PARTITION='A'
```

The select statement finds the volume access data for the given date for the partition. The db_afs_admin.pm perldoc has a full overview of all available routines:

```
.::db_afs_admin(3)    User Contributed Perl Documentation   .::db_afs_admin(3)
NAME
        db_afs_admin - Interface to AFS admin History Database
SYNOPSIS
        $used_blocks_volume = db_get_volattrib($vol_id,'Blocks',$day_count);
        $quota_volume       = db_get_volattrib($vol_id,'Quota',$day_count);
        %accesses           = db_get_volattrib($vol_id,'Accesses',$day_count);
        The keys of the %accesses hash are the partition names (e.g. afssrv1/a ).
        If the argument refers to the RW volume, only accesses to the RW volume are
        returned. If the RO name or ID is given, all accesses to the RO volumes are
        stored in the %accesses hash. If the BK name or ID is given, the accesses
        to the BK volume are returned. $day_count is number of days in the past
```

```
it is preferable to retrieve the data from. 0 and 1 is yesterday, 2 is
two days ago etc. 0 is faster than 1 and use the recent_row flag for yesterdays'
records.

$total_accesses_partition = list_partition_db($partition,'Accesses',$day_count);
$total_quota_partition = list_partition_db($partition,'Quota',$day_count);
$total_blocks_partition = list_partition_db($partition,'Blocks'$day_count);

Example: $partition = afs45/a (name server/partition)
insert_volume($volume_id);

@volumes = db_list_volumes($attrib);
Where $attrib is 'id_vol' for the volume id, or 'name' is the volume name.
Returns an @array of ALL VOLUMES in the DB.

%accesses = gd_get_volattrib($vol_id,'Accesses',$day_count, $id_server,$id_partition);
%blocks   = gd_get_volattrib($vol_id,'Blocks',$day_count,$id_server,$id_partition);
%quota    = gd_get_volattrib($vol_id,'Quota'$day_count,$id_server,$id_partition);
%files    = gd_get_volattrib($vol_id,'Files',$day_count,$id_server,$id_partition);

The gd_% functions are written for the graphical presentaion cgi script for performance
reasons. The function returns a history hash for the given attribute, where the date in 'dd/mm/yyyy'
fashion is the key for the given value.

DESCRIPTION
db_get_volattrib returns used $blocks of 1024k for a volume, with arguments
($vol_id,'Blocks',$day_count);

db_get_volattrib returns $quota (Blocks of 1024k) given to avolume, with arguments
($vol_id,'Quota',$day_count);

db_get_volattrib returns %total_accesses for a volume (last day) with arguments
($vol_id,'Accsesses',$day_count);
list_partition_db returns $total_accesses for partition with arguments
($partition, 'Accesses',$day_count);
list_partition_db returns $total_quota in blocks of 1024k for a partition with arguments
($partition,'Quota',$day_count);
list_partition_db  return $total_used_blocks of 1024k for a partition with arguments
($partition, 'Blocks',$day_count);
insert_volume with argument ($id_volume). Use this function after created a new volume
on the AFS system. Only the volume_id is needed, also for ROs.


db_list_volumes returns @array of all volumes in the DB with argument
'id_vol' or 'name'.


gd_get_volattrib returns %hash of access history for a given volume with argument 'Accesses'.

gd_get_volattrib returns %hash of blocks history for a given volume with argument 'Blocks'.

gd_get_volattrib returns %hash of file count history for a given volume with argument 'Files'.

gd_get_volattrib returns %hash of quota history for a given volume with argument 'Quota'

gd_list_partition returns %hash of Accesses history for a given partition with argument 'Accessers'
AUTHOR
Rune J.Andresen
```

AFS Admin uses the db_get_volattrib, list_partition_db and insert_volume subroutines. The same subroutines will be used for automated load balancing. The graphical user interface for analyzing volume history, described in chapter 4.4 uses as well gd_get_volattrib, which are designed to retrieve history data more efficiently for whole partitions.

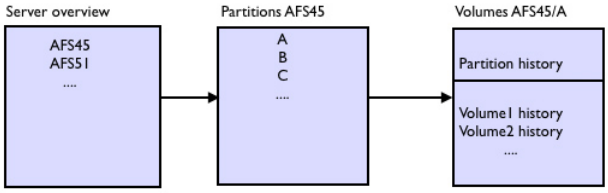

Figure 14: Perl SGI graphical presentaion

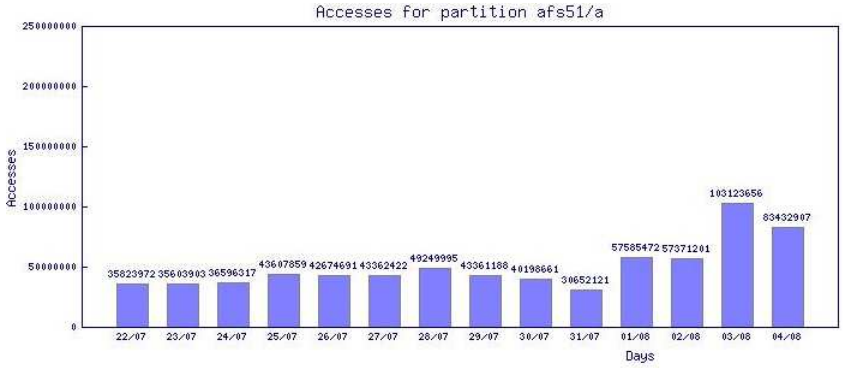

Figure 15: *Access history presented in Perl CGI*

4.4 Presenting statistics

For human interfacing, the Perl GD:Graph module is used for presenting file server history. Load balancing can't be fully automated due situations which is difficult to predict. Thus, AFS administrators need the statistics presented. The representation is as well cruical for analyzing and understanding how to make load balancing more effective. A Perl script updates server statistics images after every DB update (each hour for the selected volumes in the table daily_updates). A HTML/Perl CGI page presenting the statistics is refreshed frequently in order to present the updated data[2].

The first level has an overview of all file servers available (Figure 14). The second level has an overview over all partitions for a given server. It is possible to choose between the total partition or the induvidual volume history on the third level. Furthermore, it is possible to choose history for accesses, disk usage, disk quota and file-count. For instance, Figure 15 presents the total access history for partition AFS51/a. Figure 16 presents the total disk volume usage history from last fully update for partition AFS51/a.

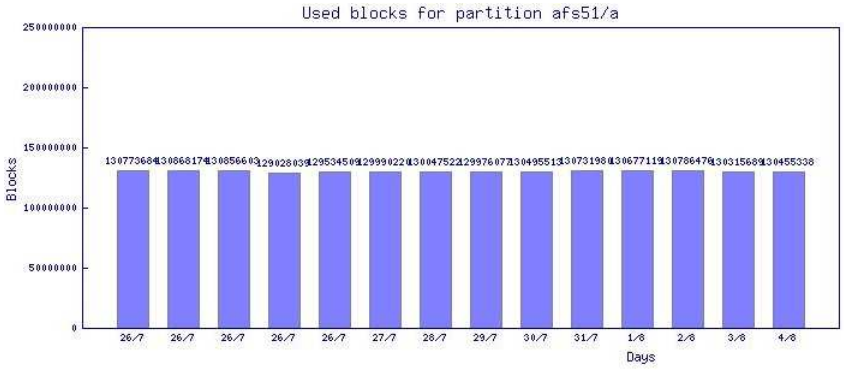

Figure 16: *Used blocks history presented in Perl CGI*

4.5 Interfacing with AFS Admin

The new AFS admin implementation interfaces with the DB using the db_afs_admin.pm package. These modifications are carried out to retrieve data faster without the need of accessing all file servers, as discussed in chapter 3. Thus, instead of examin all file servers every time project administrators need volume information, the information can be located in the DB. Also, when a AFS administrator is creating a new volume, afs_admin will update the DB directly. The DB does not need to wait until next main update to retrieve this volume data.

For most volumes the DB information is from the present day. If current volume information is required AFS Admin use the Vos and Volset interfaces that retrieves the information directly from the file servers.

4.6 Discussion

This chapter explains the most essential parts of the code. Unfortunately, some details had to be left out due to the size of the implementation, which count approximately 4000 lines of code. All files are stored on the CD in addition to a README with file explanations. The system has been running stable for 6 weeks at current time. However, old data in the DB has to be deleted or compressed in order to avoid that the tables grow for infinity. The most straightforward approach is to delete data that is 1 year old. It would be beneficial to compress old data in order to keep older history data, but this approach is more complex to achieve due to the DB design. Also, more rows of data results in slower execution of queries. This problem has to be considered in further work.

5 Methodology

The file server history is to be used for automation of load balancing. Some methods are discussed for tuning the automation, which is estimated to last for several months. Because moving all volumes in an idealistic initial position is impossible (this would make the AFS cell unavailable for several months), volumes have to be moved over time for better utilization of the system. Furthermore, the size of the CERN AFS cell is estimated to grow with a factor of 10 within 3 years and project and volume behavior may change dramatically. This chapter discusses methodology for the volume move implementation. Some central issues to take into consideration are:

- A volume with many files is more time consuming to move than a volume with fewer files.
- A volume that is heavily accessed is likely to be a potential problem wherever it is located. History should avoid that a volume is moved into a loop between servers.
- Volumes that belong to the same set should be clustered over few file servers in order to avoid that all projects are influenced by a server crash. This policy may conflict with an idealistic volume distribution.
- A new volume is stored into the file server, available for the set, with most free disk space. This initial algorithm may be improved in order to make load balancing more effective.

5.1 Metric for volume availability

The file server partitions, or disks, are the potential bottlenecks or hot spots. As mentioned in chapter 2, a disk can be over-exposed with accesses, or there are no free space left. The former may have consequences for the whole file server, blocking other partitions to perform any I/O. In order to find a partition's availability it is useful to introduce a metric, which divide total volume access with used disk space (in blocks of 1024KB).

$$DiskAvailability = \frac{TotalAccess}{UsedBlocks}$$

Calculating disk availability in this straightforward manner makes it easier to detect partitions that are not in harmony. If *diskavailability* is a large number there are many accesses each block, leaving empty space unavailible for other volumes that require less I/O requests. In the other extreme, if *diskavailability* is approaching zero, several idle volumes are occupying the partition and are not exploiting the disk's access capacity. An approximate optimized value for *diskavailability* is total accesses divided by total used blocks in the whole AFS cell over several days. An optimized value for

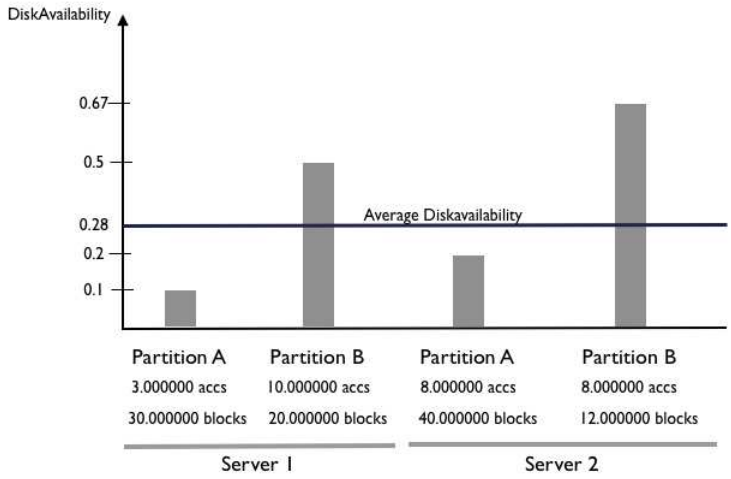

Figure 17: *Server diskavailability*

a given day would require to take disk and I/O capacity for every partition/disk into consideration.

Figure 17 demonstrates an AFS cell with two file servers with two partitions/disks each. Server1/PartitionA has an low *diskavailability* while Server2/PartitionB has an high *diskavailability*. Moving some volumes from the latter to the former would exploit the disk space in a better manner on Server2/PartitionB and exploit the ability to handle more accesses on Server1/PartitionB. Server1/PartitionB and Server2/PartitionA are close to the average *diskavailability* and are not first priorities for an optimization algorithm for load balancing. Furthermore, this scenario requires the *diskavailability* to be approximately the same in average for each partition over time before moving any volumes between servers. (See chapter 2).

5.2 Methodology of Load balancing

There are several approaches and methods for load balance an AFS cell. Initially the load balancing problem is NP complete. Finding one or several approximation algorithm(s) and tune them over time is the most realistic approach. Furthermore, even if there where possible to solve a NP complete problem in polynomial time, moving all volumes in their right position would not be an option considering the size of the CERN AFS cell[5]. Also, number of files for every volume is essential for volume moves. If there are a high file count it is time consuming and costly to move off the volume.

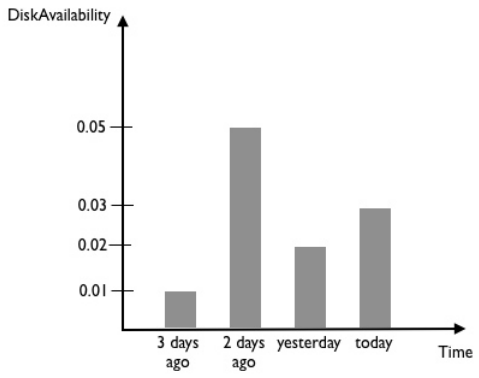

Figure 18: *Four days history for volume p.cvs.cesar*

Chapter5.2.1 will discuss some approaches for finding an approximation algorithm. Of course, this algorithm has to be tuned over several months before it is approximately optimized.

5.2.1 Method1

The most straightforward method is to consider the present *diskavailability* for a volume. If the present *diskavailability* makes the volume a candidate for moving it off to another server, the algorithm takes the volume history into consideration. The importance of Diskavailability will decrease with the history timeline. For instance, the diskavailability of the present date will have more importance than the day before and so on.

Figure 18 demonstrates an example. Volume p.cvs.cesar has a high *diskavailability* at present time and considered to be moved to a partition with less workload. To make sure that is not an exception, history is validated to maximize the probability that the number of accesses the next day will stay at the same level. The statistics of the present date will count most and older history data have less influence. Lets say we take 100% of today's value into consideration added to 90% of yesterday's value and so on. The volumes on the partition with the highest values are candidates for moving. Using history in this fashion the algorithm will find which volumes that have highest (and lowest) *diskavailability* over time, eventually calculate the probability that the volume move will gain system performance. Thus, an approach for load balancing with method1 is:

1. Detect partitions with the highest total *diskavailability*. The number of how many partitions to take into consideration is a variable, *nr_partitions*, and has to be tuned over time. As well, the partitions with the lowest diskavailability have to be detected as a target for volume moves.

2. For each partition, volumes with the highest diskavailability at present date are detected. The number of volumes to be evaluated is an variable, *nr_volumes*, that has to be tuned over time.

3. Calculate a sum for each volume that are selected in stage 2 with the following approach: If *diskavailability* for today is *da0, diskavailability* for yesterday is *da1, diskavailability* for two days ago is *da2* etc, a formula

   1xda0 + 0.90xda1 + 0.80xda2 ... 0.1xda9

   can be used. The decreasing ten percent, *dec_percent*, and ten days in history, *history_days*, are two variables that have to be tuned over time.

4. Sort the volumes by *diskavailability* over time and move off a number of volumes, a variable *nr_volumes*, to partitions with free space and low diskavailability.

5.2.2 Method2

Another method is to locate trends in the volume history. A volume may have a repeating access pattern over time. If this volume, let say, only is accessed in the weekdays, another volume that is accessed only in the weekends could be moved to the same partition as the former volume. Several project volumes at CERN are accessed by cron jobs[13] for storing data, making this approach useful for load balancing. Figure 19 and 20 demonstrate two volume histories that fix together, having the opposite access period. Volume 1 has a weekly pattern with a high diskavailability on mondays and Tuesdays, and a low diskavailability on Fredays. Volume 2 has a weekly pattern with a high diskavailability on Fredays, and a low diskavailability Mondays and Tuesdays.

This repeated period for volumes are only likely to be valid for project accessed by cron jobs. Volume access pattern for volume belonging to end users or humans in general are impossible to predict. Hence, method2 is likely to work more efficient when used on volumes or partitions that are mainly accessed by cron jobs.

5.2.3 Discussion

An approximation algorithm will eventually result in improved system performance, using method1 or/and method2, tuning the algorithm variables over time. Nevertheless, moving volumes between partitions has a cost. Large volumes with a large file count can take hours to move, and volume data is unavailable during the cloning process (Chapter2). Some analysis has to be performed in order to move such volumes. Small volumes with a low file count may be moved between servers for tuning the algorithm variables. If the move turned out to be a bad move, it can easily be moved back to it initial position. A requirement for large volumes is that they are moved between

[13] Cron jobs are scheduled to be executed periodically

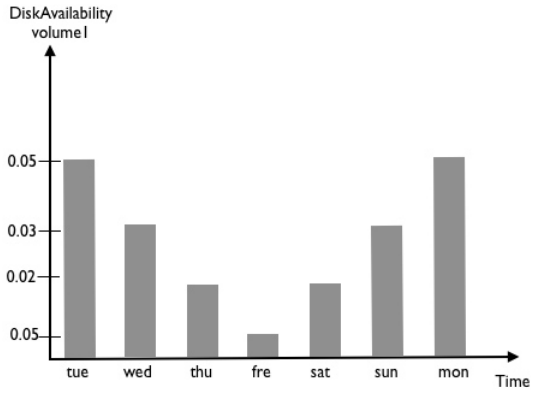

Figure 19: *Diskavailability Volume 1*

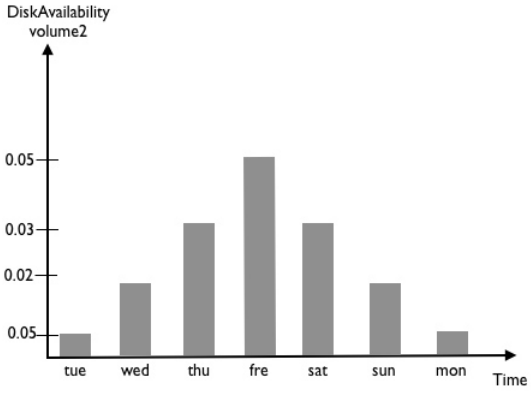

Figure 20: *Diskavailability Volume 2*

servers with a limited frequency, proportional to the size/file count, due to the costs of moving them.

6 Analysis

This chapter analyzes the statistics data from the DB and measure the time of volume moves, for better understanding the state of the CERN AFS cell. First chapter 6.1 analyzes the history data for the server level. Chapter 6.2 analyzes the partition level. This is followed by chapter 6.3 analyzes trends in different kinds of volumes, that is, history trends for user volumes, project volumes and scratch volumes. The graphical interface to the DB is used for retrieving the data, described in chapter 4.4. Finally, in chapter 6.4, the analysis is evaluated in the light of the cell's size and performance.

6.1 Analysis on server level

The distribution of volume types is different for each server. Some servers have a majority of user volumes and others have mostly project volumes. Because of the hypothesis that project volumes are accessed with cron jobs and have a repeated access pattern that is easier to predict than user volumes, this chapter analyzes history data for a server with a majority of user volumes and another server with a majority of project volumes. It is beneficial to find a connection between number of user volumes, project volumes and accesses. If file servers with a majority of project volumes have a repeated period of accesses, this can be exploited for better load balancing. On the other hand, user volume accesses that are predicted to not recur at regular intervals, or nonperiodic, may turn up to have some intervals that are beneficial for load balancing.

6.1.1 Analyzing AFS45

AFS45 has at present time 4% user volumes, 58% asis volumes, 30% project volumes and 8% q volumes. Table 1 includes the history data for total accesses, used blocks and quota. Figure 21 is a graph with the following *diskavailability* for the server. *Diskavailability* is as explained in chapter 5 accesses divided by used blocks.

Date	10.07.05	11.07.05	12.07.05	13.07.05
Accesses	8.599.521	10.434.954	19.972.144	6.602.221
Used blocks	234.096.533	185.049.331	185.049.331	185.049.331
Quota	354.470.706	356.470.706	356.470.706	356.470.706

14.07.05	15.07.05	16.07.05	17.07.05	18.07.05
12.748.033	6.764.156	4.040.520	4.683.897	6.284.044
185.049.331	253.999.686	253.999.686	253.999.686	249.003.920
356.470.706	356.470.706	356.470.706	356.470.706	356.470.706

19.07.05	20.07.05	21.07.05	22.07.05
14.873.407	11.160.834	12.931.755	9.911.630
249.003.920	249.003.920	249.003.920	249.003.920
356.470.706	356.470.706	356.470.706	356.470.706

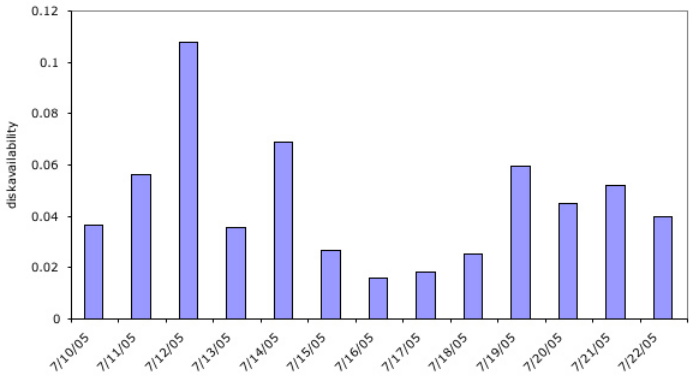

Figure 21: *Diskavailabily history for afs45*

Table 1: Accesses, used blocks and quota history for file server AFS45 from 10.07.05 to 22.07.05

The *diskavailability* history in figure 21 demonstrates a trend where the *diskavailability* during the weekends is lower than in the weekdays. There are also two local tops on the 12.07.05 and 19.07.05 (Tuesday) if the weekend 16.07.05 and 17.07.05 is considered a universal lower point. On the dates 13.07.05 and 20.07.05 (Wednesday) the *diskavailability* is lower than Tuesdays, and is growing on Thursdays. Hence, the graph has as a period that repeats. However, the total values are lower after Sunday 17.07.05 than the week before.

6.1.2 Analyzing AFS91

AFS91 has at present time 18% user volumes, 20% q volumes and 60% project volumes.

Date	11.07.05	12.07.05	13.07.05	14.07.05
Accesses	40472263	34422306	23843116	12474376
Used blocks	186467835	186467835	186467835	186467835
Quota	586074366	586074366	586074366	586074366

15.07.05	16.07.05	17.07.05	18.07.05	19.07.05
15558158	7884740	21190065	20378314	15843409
186467835	186467835	186467835	186467835	186467835
586074366	586074366	586074366	586074366	586074366

20.07.05	21.07.05	22.07.05	23.07.05
16626915	24825922	42647840	33022732
186467835	186467835	186467835	186467835
586074366	586074366	586074366	586074366

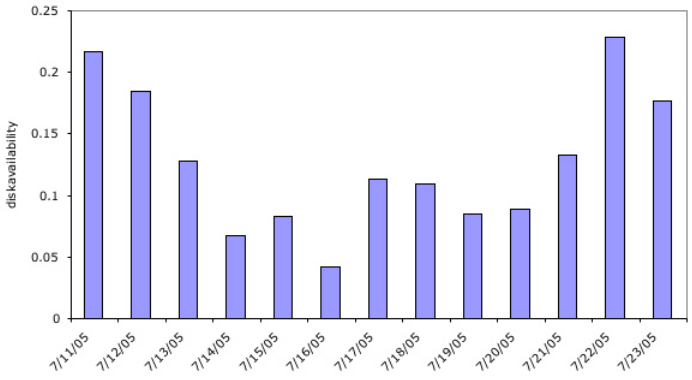

Figure 22: *Diskavailabily history for afs91*

Table 2: Accesses, used blocks and quota history for file server AFS91 from 11.07.05 to 23.07.05

Figure 22 reveals no repeating period or pattern despite the high number of project volumes. From Monday 11.07.05 until Saturday 16.07.05 the *diskavailability* is decreasing every day. The next week the diskavailability is growing to reach a global maximum for this history graph on Friday 22.07.05.

6.1.3 Comparing AFS45 to AFS91

During a 12 days period it is possible to see a repeating period for AFS45. The relationship between weekdays remained approximately the same. This server has a majority of project and AFS configuration volumes. Analysis of AFS91 did not reveal any weekly repeated period. AFS91 has 18% user volumes and AFS45 has 4% user volumes. The low number of user volumes in the latter may be one reason for the repeating period of *diskavailability*.

6.2 Analysis on partition level

Server analyses take all volumes on all partitions into consideration for a given server. These analyses do not differ between the partitions. Thus, it is desirable to analyze the individual partitions, which are the potential bottlenecks. Volumes sets are as well distributed over partitions, not servers. A disk failure is more common and more likely to occur than a server crash. However, a server has a upper limit for I/O requests, which makes both server and partition analyses important.

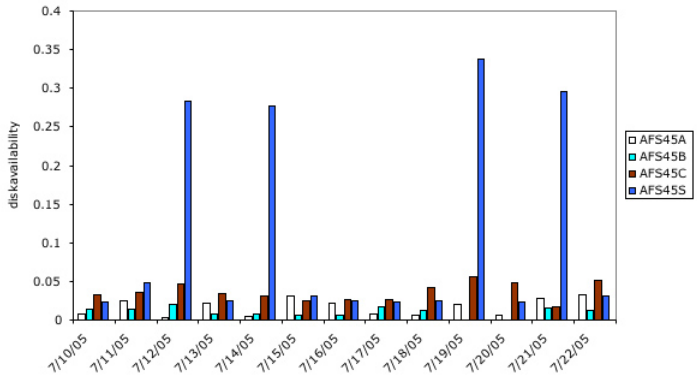

Figure 23: *Accesses, used blocks and quota history for file server AFS45*

6.2.1 AFS45 Partitions

AFS45 has four partitions, AFS45/a, AFS45/b, AFS45/c and AFS45/s. Calculating *diskavailability* for each partition reveal some tendencies for AFS45. *Diskavailability* for the whole server (see figure 21) has a lower limit 0.02 and an upper limit 0.1. Figure 23 demonstrates that *diskavailability* for the partitions have a lower limit 0.004 and an upper limit of 0.35. It is possible to recognize the pattern from figure 21, for server AFS45, when analyzing figure.23. Nevertheless, the maximum values, dated 12.07.05, 14.07.05, 19.07.05 and 21.07.05, differ more for the partitions than for the server. The relatively low diskavailability for AFS45/a, AFS45/b and AFS45/c compared to AFS45/s smooth out the differences for the total server history. This demonstrates the importance of analyzing the partitions in order to find potential bottlenecks. For instance, an increasing amount of accesses for partition AFS45/s may over exhaust the server capacity, which is not straightforward when only considering the total server statistics, given that number of used blocks for each partition does not vary much. AFS45/s houses a majority of asis volumes[14], which eventually have a repeating period for accesses. The asis accesses may be regeneration of symlinks for public domain software.

6.2.2 AFS91 Partitions

Figure 24 reveals that partitions AFS91/b and AFS91/s have very few accesses. Partition AFS91/c *diskavailability* history is nearly identical to the server history, Figure 22. AFS91/c houses only Atlas project volumes, which indeed are frequently accessed. This partition *diskavailability* history has no obvious repeating period, which for this case disproves the hypothesis that project volumes are easier to load balance due to a repeating access pattern.

[14] AFS configuration volumes

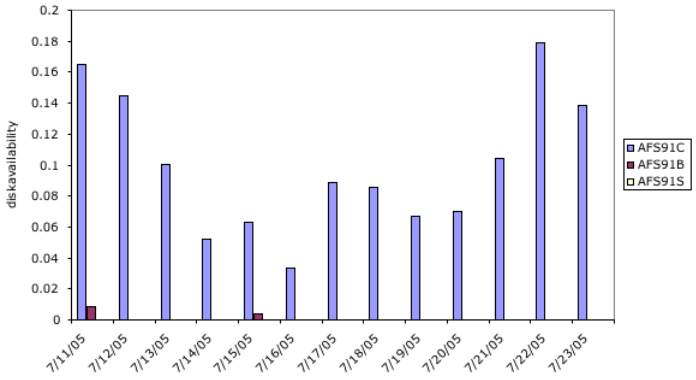

Figure 24: *Accesses, used blocks and quota history for file server AFS91*

6.3 Volume moves

Moving volumes between partitions has a cost, which has to be considered before a volume actually is moved. As discussed before, a large volume is time consuming to move, and thus, could make the volume data unavailable for several hours during the cloning process. If large a volume is decided to be moved, it is particularly important that the move gain the overall performance, and it does not to be moved again in the near future. Because volume moves move data between partitions, they have a influence on the server's response time during the moving process, and other volumes may be less available. Measuring volume moves when a server is heavily accessed could give very different results. Thus, the following measure of volume moves are performed during the night when most of the servers are less accessed, and thus, only give the best case results. Nevertheless, most volumes should be moved at night to avoid competing with the volume accesses during the day. Table 3 includes the average volume moves between two IBM servers with CERN Linux installed and two Sun servers with Solaris installed. The volume sizes are 2GB, 4 GB and 6GB.

size	time IBM/Linux to IBM/Linux	time Sun/Solaris to Sun/Solaris
2GB	4m5.436s	20m2.315s
4GB	4m46.019s	22m18.660s
6GB	8m46.265s	30m57.39s

Table 3. Timed volume moves between IBM/Linux servers and between Sun/Solaris servers

Figure 25 demonstrates the difference in time between moving volumes between two IBM/Linux servers and two Sun/Solaris servers. The Sun servers are older than the IBM/Linux servers, which is likely to be the main reason for the time gap. The Sun/Solaris servers use as well RAID5 disks for backup, which save disk usage but slow down the writing process. The IBM/Linux servers use disk mirroring, which in-

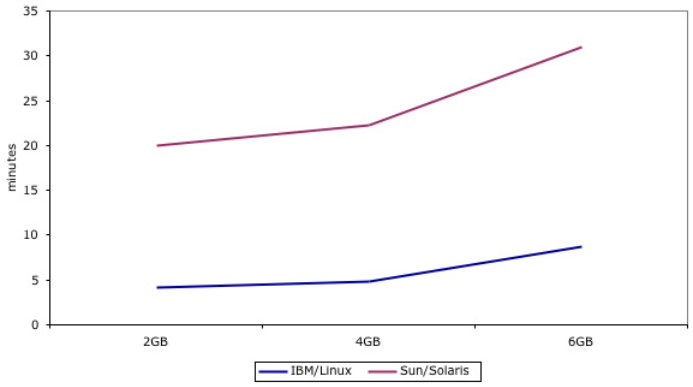

Figure 25: *IBM/Linux volume moves vs Sun/Solaris moves*

crease the write process, but requires more disk space. Nevertheless, in the CERN AFS cell there is a 2:1 relationship between reading and writing, which makes the reading process the important one. For both the IBM/Linux and the Sun/Solaris servers the measured time for volume moves for 2GB and 4GB are very close. The initial volume-move process, cloning the volume, copies the vnode index, which is a table of pointers between directories and files in the volume. Hence, the volume file count is the critical variable for the initial state for volume moves, which is the same value for all volumes in this experiment. From 4GB to 6GB the Sun/Solaris servers volume moves are more increased in time than the IBM/Linux servers. This is eventually due to more time moving the volume than actually creating the clone, and the Sun/Solaris servers (using RAID5) write slower than the IBM/Linux servers.

6.4 Discussion

Finding repeating periods for disk partitions and servers containing project volumes where not as straightforward as predicted. User volumes may be as well easy to predict than project volumes, considering that most of them are used during the week and are idle during the weekends and nights. Nevertheless, because sets of volumes are gathered on few partitions, it should be possible to locate the projects that have a stable repeating access pattern, if any, analyzing on partition level. Querying every volume that belongs to a set could easily be costly and time consuming. Of course, when the DB is growing and more history data is available, other trends for the different kinds of volumes may be revealed. Considering the approximate 20 000 volumes in the CERN AFS Cell at current time, finding matching volumes, discussed in chapter 5.2.2, is neither a straightforward task, which in theory requires approximately 20 000 compare operations for finding the best match.

Timing volume moves revealed a difference between the Sun/Solaris and IBM/Linux

servers, which is eventually critical for load balancing. It is more costly to move off a volume from a Sun server than from a IBM server.

7 Trouble shooting

The volume accesses statistics are reset every 24 hour. Because of a bug in the Open AFS code this was done 08.00 in the morning for the Solaris fileservers, and not at midnight. Diskstat is updating the data at 05.00, which means that the access statuses for the Solaris servers are at another stage than the Linux servers. The fix is to run diskstat just before the access data is reset, and set the Solaris servers to be reset at midnight. The original script used a time zone correction function that did not work.

Performance for the updating script was poor due to many select and update statements (approximately 43000 loops each time the update script was run). The program was updated with hash tables to be capable of retrieving more data in one select query. The script accesses the hash tables more frequently than accessing the database, which reduces execution time (From approximately 5 hours to 3 hours).

8 Conclusion

As the distributed file systems grow, it becomes even more critical for them to maintain performance regardless of increased data handling. This is particularly true at CERN which through its LHC project is expected to have to handle Petabytes of data in the future.

In this thesis, we showed that by developing a system that stores file system volume history data in a commercial database, and integrating this data with a GUI for analysis, the file system could be significantly optimized.

Volume distribution could be performed more efficiently by using more knowledge of the volumes' access pattern, disk usage and disk quota. By distributing the highly accessed volumes among all available file servers, overall system performance gain was achieved. Using volume history data, it was also possible to calculate a better prediction of the volumes' behavior in the future.

Our developemet target environement was the AFS files system at CERN, and Oracle was used for our underlyingdatabase since it is already supported at CERN. The database setup and required scripts for interfacing with the database were designedas part of this work. Furthermore, the database was in addition to making load balancing more effective, also intended to make AFS administration more straightforward. Administrators do now not need to collect data from all servers in order to retrieve volume status unless very recent data is required.

In order to avoid unnecessary data replication,database tables where divided into thetypes of data that was most frequently updated. Volume data that was not modified since thelast update, was time stamped, makinga time boundary for when a row of data is valid. If volume data was modified since the last update, a new row was added to the database.

The database interface was a Perl package, intended for communication betweenload balancing scripts and the database. Perl scripts were written for updating the database. A graphical user interface was designed and implemented in order to analyze volume history data for better understanding of volume accesses, volume quota and disk usage in the CERN AFS cell. The GUI was implemented in Perl CGI, using the database interface for retrieving volume history data.

These analyses will eventually be used for approaching a fully or partly automated load balanced file system, which will require history data from several months and tuning over time.

Our history data for 11 days revealed that some volume behavior is easier to predict than other. A hypothesis was that project volumes were easier to load balance than user volumes due to an assumed repeated access and disk usage pattern. Unfortunately, analysis of the Atlas project volumes turned up to have irregular access patterns over a period of 11 days.

Another hypothesis was that user volumes would likely to have irregular access patterns, which was partly true considering available history data. However, user volumes were mostly accessed during the day, and not during the nights and the weekends. Such

volumes could hence eventually be gathered on partitions with volumes thatare mainly accessed by cron jobs during the nights for better system performance.Thus, available history data at recent times is not sufficient for making any final conclusions. Nevertheless, taking our initial analysis into consideration, all kinds of volumes should be considered for moving in order to gain overall system performance.

To ease load balancing a new parameter on *disk availability* default was introduced. *Diskavailability* is is accesses divided by used data blocks of 1024K. This parameter reflects how much a volume is accessed with respect to the disk usage. On the one hand, this is important because heavily accessed volumes should be distributed over all partitions in order to avoid that few servers are overloadedby accesses. On the other hand, idle volumes should also be distributed over all partitions to avoid that whole servers are idling, and thus, not helpingthe overall system to receive requests. Hence, knowledge of disk vailability helped load balancing by locating volumes and even whole partitions that had too many accesses to each used block, or the opposite,too few accesses to each used block, compared to the total disk availability and accesses capacity.

Further work for load balancing will be based on methodology and analysis introduced in this thesis. More volume history available from the database should beanalyzed in order to find trends for volume sets and kinds, which can makeload balancing even more reliable and effective.

Appendix

A Database

A.1 Sqlplus Script for Database Tables and Indexing

```
Alter table volumes drop constraint v_id_server_fk;


       Alter table volumes drop constraint v_id_vol_fk;
       Alter table volume_data drop constraint v_d_id_server_fk;
       Alter table volume_data drop constraint v_d_id_vol_fk;
       drop table fileservers;
       create table fileservers(
           id_server number(25) not null,
           name varchar2(25) not null,
           CONSTRAINT fileservers_pk PRIMARY KEY(id_server)
           )
           ORGANIZATION INDEX;
       drop table id_volumes;
       create table id_volumes(
           id_vol number(20) not null,
           name varchar2(45) not null,
           set_id varchar2(20) not null,
           CONSTRAINT id_volumes_pk PRIMARY KEY(id_vol)
         )
        ORGANIZATION INDEX;
       drop table volumes;
       create table volumes(
           id_vol  number(20) not null,
           id_server number(25) not null,
           id_partition varchar2(25) not null,
           date_add2db date not null,
           date_updated date not null,
           id_rep number(20),
           id_backup number(20),
           date_created date not null,
           date_deleted date,
           recent_row number(1),
           freq_update_row number(1),
           CONSTRAINT volume_pk PRIMARY KEY(id_vol,id_server,id_partition,date_updated),
           CONSTRAINT v_id_server_fk FOREIGN KEY (id_server) REFERENCES fileservers,
           CONSTRAINT v_id_vol_fk FOREIGN KEY (id_vol) REFERENCES id_volumes
           );
       drop table volume_data;


create table volume_data(


           id_vol number(20) not null,
           id_server number(25) not null,
           id_partition varchar2(25) not null,
           date_add2db date not null,
           date_updated date not null,
           blocks_quota number(20),
           blocks_used number(20),
           accesses number(20),
           recent_row number(1),
           freq_update_row number(1),
           files number(10),
           CONSTRAINT volume_data_pk PRIMARY KEY (id_vol,id_server,id_partition,date_updated),
           CONSTRAINT v_d_id_server_fk FOREIGN KEY (id_server) REFERENCES fileservers,
           CONSTRAINT v_d_id_vol_fk FOREIGN KEY (id_vol) REFERENCES id_volumes     );
       drop table daily_updates;
       create table daily_updates(
           id_server number(25) not null,
           id_partition varchar2(25) not null,
```

```
    id_vol number(20) not null      );
drop table time_full_update;

create index performance on volumes (decode(RECENT_ROW,1,RECENT_ROW));
create index performance2 on volume_data (decode(RECENT_ROW,1,RECENT_ROW));
```

A.2 Oracle indexes and functions

Function based index to select "better" the recent_row=1 condition:

decode(RECENT_ROW,1,RECENT_ROW);

Gather statistics for the Cost Based Optmizer (in sqlplus):

exec dbms_stats.gather_schema_stats(ownname=>'RJANDRES',cascade=>true, estimate_percent=>80, method_opt=>'for all indexes for all indexed columns size 30');

B Volume Moves Timing Tables

B.1 IBM/Linux to IBM/Linux

Timing volume moves for 2GB, 4GB and 6GB during one night, Linux to Linux:

Nr	1	2	3	4	5	6
2GB	4m1.931s	4m53.340s	3m43.299s	3m29.384s	4m45.192s	3m39.289s
4GB	5m10.543s	5m54.286s	4m36.380s	4m42.071s	5m43.724s	4m46.375s
6GB	8m38.818s	9m37.727s	8m13.224s	8m23.505s	9m40.441s	8m17.321s

7	8	9	10	11	12
3m30.167s	4m42.817s	3m33.874s	3m28.404s	4m39.144	3m38.394s
4m43.424s	5m48.504s	4m33.633s	4m45.588s	5m48.405s	4m39.208s
8m26.665s	9m37.676s	8m9.263s	8m30.040s	9m32.552s	8m7.888s

B.2 Sun/Solaris to Sun/Solaris:

Timing volume moves for 2GB, 4GB and 6GB during one night, Solaris to Solaris.

Nr	1	2	3	4	5	6
2GB	21m8.771s	28m30.730s	18m57.411s	20m2.177s	19m23.365s	18m47.078s
4GB	22m25.202s	32m27.999s	19m37.687s	26m33.581s	20m0.724s	19m51.056s
6GB	32m11.850s	29m22.501s	37m36.545s	34m25.657s	28m27.262s	29m53.709s

7	8	9	10	11
19m29.774s	18m8.880s	18m49.656s	18m59.859s	18m7.762s
22m51.147s	19m55.288s	20m49.093s	21m10.755s	19m42.733s
31m50.523s	28m52.349s	28m9.173s	30m55.931s	28m45.740

References

[1] Afs administration reference version 3.6.

[2] http://afs-monitoring.web.cern.ch/afs-monitoring/cgi-bin/stats.cgi.

[3] What's new at cern? http://public.web.cern.ch/public/content/chapters/aboutcern/cernfuture/whatlhc/whatlhc june 2005.

[4] Andy Duncan and Jared Still. *Perl for Oracle DBAs*. O'Reilly, 2001.

[5] Thomas H.Coremen, Charles E.Leiserson, Ronald L.Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[6] Larry Wall, Tom Christiansen, and Jon Orwant. *Programming Perl*. O'Reilly, 2000.