Silenus

Dissertation Proposal Document

Maximilian Berger

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I. INTRODUCTION

Storing of data has always been an issue in computer science. Sure, saving your data to a hard drive is easy and convenient, but there are several things that could happen.

The first problem is that of data theft. Nowadays this has become one of the most important issues, but unfortunately it is still overlooked by many developers. On most PCs a person sitting at the machine can read any data. Now you might say, that it is very unlikely for someone from your competition to walk into your office and turn on your computer, but think about how many people actually have the key to your office? You co-worker, who may not like you, a housekeeper, who is underpaid, and so on. Even if your data is stored on a server, any administrator can usually read all data.

The second, and most noticeable problem is that of computer failure. Computer are not, and will never be failsafe. As a matter of fact, at any given time only 80% of all machines on the network are working. Imagine having an important report on the server and not being able to work on it, because the server is down. There are different possibilities for failure: Planned maintenance, unplanned outages, network failure or server failure. Most of the times these failures are temporary, which is just annoying, but sometimes these failures are permanent. In this case you can only hope that you have a recent backup.

These are just two of the problems with todays file storage system. Both can be solved using much energy and thought. A server could be put in a secure room with an alarm system where only one person has access. There could be multiple network connections, multiple servers, with fail over, a daily backup system, and so on. But solving these issues is very time consuming and requires a lot of maintenance. Smaller companies or even home users will not do all that to protect their data.

So there must be an easier way to manage data files. A way, that enables the average user to take advantage of todays networked world, without buying expensive hardware or hiring an expert. This, however, calls for a new paradigm in networked computing.

Paradigms of computer networking changed over time. When the first multi-user computers where introduced, they used the server-client paradigm. One large server would handle all the time-consuming tasks, and multiple, so called "dumb terminals" did nothing but interaction with the user. Would the server fail, no users could work. The next big trend in the computer industry was the Personal Computer. Instead of being dependent on other machines, no each user had his own machine. Would a machine fail, this persons data could be lost, but no one else would be affected. Handling many of theses systems was a difficult tasks for administrators. They had to physically sit at the machine and disturb the user for each maintenance task. So people began networking their personal computers. They went back to the client-server paradigm for some things, such as storage space, and used their personal computers for other things, mostly computation. This is the current state in most computer networks around the world.

Another networking paradigm has emerged quite recently. It is called peer-to-peer. In a peer-to-peer architecture each client is also a server and each server is a client. Some people call these "servent". The main idea is, that instead of just consuming resources, like a client, or offering services, like a server, a machine will do both. Peer-to-peer software is mostly used in file-sharing networks, such as bit-torrent. Instead of downloading a file from a single location, a user can now download a file from every other user which already has this file. This saves bandwidth and can vastly improve performance. Unfortunately peer-to-peer networks have a bad reputation, since most of the content found in these networks should not be shared in the first place. However, peer-to-peer networks are very recent technology and the area of current research.

The fourth, and most advanced network paradigm is the one of service oriented computing. Peer-to-peer is already an advanced step, but why stop there? In service oriented computing, a service exists on the network. It could provide computational power, storage space, or other things. But most important: Its location does not only matter, it may even change, if a machine becomes unavailable. The software for a service doesn't need to be installed on a computer. Any computer joining the network can automatically pick up services, and provide them to all other computers. This provides a very dynamic and fail proof network. SORCER, developed by Dr. Sobolewski at the

Texas Tech University, provides a framework to support service oriented computing. There have been several projects researching the distribution of computational power, but so far none concerning the distribution of storage space.

And this is how this all comes together. SILENUS is a distributed file storage system that is secure, failsafe and easy to use. It uses a different approach: File Storage is a Service, and the user should not need to know where, when, or on which machines the actual files are stored. Files are automatically replicated and migrated. Data is encrypted and available to only authenticated users.

Short Overview

To give a quick overview over this dissertation, it can be boiled down to a 4-Blocker. Figure I.1, “4-Blocker” gives a one-page summarization of the elements involved.

SILENUS - Sorcer Integrated Local New User's Storage

Maximilian Berger

Problem Statement	Objective / Approach
Existing distributed file stores - are difficult to set up - require high maintenance - have problems with server downtime - don't handle topology changes - don't provide privacy from administrators - don't scale for large number of nodes - are incombatible with service-oriented architectures	Objective: to create a new filestorage based on the SORCER environment Approach: - Authentication for Authorization - Encryption for privacy - Replication for availibility - Multisource download for performance
Schedule	**Benefits**
Background Research 01/04 - 12/04 Initial Proposal 06/04 Prototype Design 06/04 - 12/04 Prototype 10/04 - 06/05 SO Redesign 07/05 - 12/05 SO implementation 10/05 - 10/06 Proposal 03/06 Defense 10/06	- comprehensive security - completely self-managed - fully transparent for the user - adding disk space is easy - compatible with legacy applications via WebDAV interface

Figure I.1. 4-Blocker

Dissertation Outline

This dissertation is written in six chapters. Chapter I (this chapter) introduces the problem of storing data. It then gives an overview over this dissertation.

Chapter II describes the background research and literature review necessary to understand the problems and proposed solutions in this dissertation. It is written in two parts. The first part describes existing file storage solutions, their advances and disadvantages, and how they relate to SILENUS. The second part looks into different qualities for distributed systems.

Chapter III describes the detailed objective of the SILENUS solution. The exact requirements are extracted from the knowledge about existing solutions and their shortcomings.

Chapter IV proposes a solution based on the technology investigated in the background research. It gives a possible solution based on a service oriented architecture. It then describes the individual components.

Chapter V will describe a prototype based on the solution proposed in Chapter IV. It will look at a specific implementation of the proposed solution. It will describe details and algorithms that were necessary to solve problems that will appear during the implementation. It describes which test cases where used to validate the proposed solution.

Chapter VI will summarize the dissertation. It will provide an overview over the things learned. It will compare the new solution with the existing ones. At the end it will describe further work and research directions.

CHAPTER II. BACKGROUND AND LITERATURE REVIEW

The literature review and background includes two parts: In the fist part existing distributed file storage systems are looked at and analyzed. Their content will provide a solid base for the current state of the art. In the second part different techniques and approaches are investigated. This is needed to make a good decision on which approaches to choose for the actual design.

Existing network file storage solutions

Before developing a new solution, one has to look at existing solutions. For once, they might provide very good hints on what is done and what is still missing, but for many people these existing solutions might already provide all the features needed.

This chapter will look at different existing network file storage solutions. Single computer solutions are skipped, as this dissertation is about distributed data storage. Different solutions will be looked at in the order of their complexity and the amount of functionality they provide.

When looking at these file systems three types of file systems have to be distinguished. The first type provides remote access to files, but these files exists in one place only. NFS and CIFS are examples of such file systems. The second type provides file replicas, providing better access and higher availability. AFS and Coda are examples of replicated file systems. The third type of solutions are data grid solutions. These provide full data management, mostly for high-performance computing applications. Globus GridFTP and the Avaki data grid are example solutions.

Non replicated remote file systems

Non replicated remote file systems are network file systems where the actual data exists in only one place on only one machine. This usually means much less overhead, and simplicity. However it also means less safety in the case of failures. If the machine that contains the file is unavailable then the file will not be available.

Network File System (NFS)

NFS is the most widely used network file system in Unix environment. It was originally developed by Sun but is now available on almost any Unix or Unix-like operating system. It is implemented as a set of remote procedure calls (RPC). It provides only host-based authentication and only suggests to obey use permissions. File-locking was not possible until version 3 and still provides problematic between different operating systems. Newer implementations of NFS provide a little more security, but these are less used do to incompatibilities with other operating systems. NFS is absolutely not reliable in the case of network failures: The administrator can chose between "fail" after a certain timeout or "hang forever". Despite all these shortcomings NFS is a very fast network file system with very little overhead. It works very efficient in local area networks (LAN). NFS mounts can be read-only cached for improved performance. Migration of data is impossible: Data is referenced by the server name and the location on the server. [1, 2]

Migration and replication of data has been added to version 4 of the NFS protocol. Unfortunately this specification is still fairly new, and so current implementations are limited. Many existing clients are now just having a fully working NFS v. 3 implementation. Even in NFS 4 a server must still be available to tell clients about the new location of their data. [5]

Common Internet File System (CIFS)

CIFS was first developed by IBM under the name Server Message Block (SMB) protocol. If was then re-used by Microsoft as their network file system protocol and then later renamed to Common Internet File System (CIFS). It this system a user connects to a specific storage on a specific server. Then she can use the remote disk space like any local disk space. Locking and authorization are provided. The biggest drawback of CIFS is that it provides user-based authentication only. An administrator can not mount a file system for all users and give them different permission. Despite other claims CIFS is very secure: Since every user has to authenticate herself there is no need to trust the client computer. User administration is needed on the server only. CIFS is the most commonly

used file system protocol in the windows world. It even provides browsing for available shares. CIFS file systems can be easily migrated to different locations on the same server machine but not across multiple machines. [14, 13]

Despite of these drawbacks, single replica file systems are still the most common used. The main reason for that is their sheer simplicity. Any new remote file storage will have to compete with that. Even though more sophisticated solutions are available, most Unix-like systems still use NFS and most Windows-systems use CIFS.

Replicated file systems

Replicated file systems keep their data on more than one server. There will always be multiple copies of each file. The advantage is that now only one of the machines has to be available. This helps to provide availability in the case of hardware and network failures. Multi-replica file systems are more sparsely used. They require a substantial amount of administration.

Andrew File System (AFS)

The Andrew File System was originally developed at the Carnegie Melon University (CMU). It was then continued by IBM, and eventually made open source. AFS was intended as a replacement for NFS on Unix machines. However, the AFS software is available for all common operating systems. AFS has a wide variety of features: The user does not need to know where the physical file is, only the address of an AFS master server. The master server and all data can be replicated. Replicas are usually read-only, but can be made the upgraded to the master copy in case of a permanent failure. The client software usually creates caches the data locally, giving better performance. AFS security is handled via Kerberos, which is a common standard for authentication. AFS data, however is not encrypted. The number of replicas of a file depends on what store the file is in.[38]

AFS is a very good distributed file system. Many larger organizations such as large companies and Universities use it. One major drawback used to be high license fees, which has disappeared since the software was made available as open source.

The biggest problem with AFS is the time it takes to set up. Configuration is very complicated. It is easier if a Kerberos server is already in place, but it will still take a long time. This makes AFS unusable for the small work group or the home user.

Coda

Coda is also developed at the CMU. It is based on the code of AFS. Code provides many more interesting features, such as read-write replicas and hoarding. Coda even has conflict resolution: Should the network connection between two servers fail while two clients are writing on them it will automatically detect conflicts and provide both files. Coda also requires OS support, which is only available for a limited number of operating systems. Coda is still in an experimental stage, and not recommended for production use. Code provides some kind of security, unfortunately some of it has been cut out due to the encryption export restrictions of the USA.

Coda provides very interesting features: Especially the disconnected operation and automatic conflict resolution in code is very sophisticated. Unfortunately the setup of Coda still requires a lot of manual administration.[15, 16]

Data grid solutions

Data grid solutions try to provide common data for computation intensive, distributed applications. They usually require specially written applications to function properly.

Globus file store

The file storage system in Globus was invented from a different viewpoint. While the others tried to supply a file system to all legacy applications, the Globus system tries to supply efficient file storage to new applications, which are specifically written for the Globus system. The Globus system is used to for distributed computing. Since this usually involves a lot of data, the main focus here was on performance. Files can be downloaded from multiple sources to prevent server overload.

The Globus file storage system has very good ideas. The main drawback is its incompatibility with legacy applications and that it never was meant to be a file system for legacy applications. [17, 18, 19, 39]

Avaki

Sybase Avaki Enterprise Information Integration (EII) provides a comprehensive grid data management solution. It stores data at different locations but provides one common interface to the user application. It combines data from different sources on different machines and different locations in an unified view.

Avaki does not use redundant data copies and replicas. It deliverers data from its original source. It is rather optimizer for already existing fast and reliable network infrastructure in high performance computing labs.

Avaki originally started out at the University of Virginia under the name Legion. It was commercialized in 2000. In 2005 Avaki was bought by Sybase and integrated into their line of data oriented services.

The original Legion software was seen as a grid portal rather than a data management solution. It provides unification of different data sources and access through the same interface. It is the common interface that makes Avaki interesting.

A major drawback of the Avaki software is its cost. Being a commercial software the initial costs are very high. The software requires an existing, reliable infrastructure. As such it may be good for larger organizations but is totally unfit for the end user.

The Avaki software is unable to handle disconnected operations. Accessing data from its original source means that the original source must be available: the network must be working, the machine must be up and the software must be running. All these assumptions can only be made in a very controlled environment that hardly exists outside of lab conditions. [20, 21, 22, 23, 24, 40]

File system core features

A set of distributed file system core features can be defined based on the analysis of the existing file storage solutions. Table II.1, “File system core features on remote file storage solutions” shows these features and gives an overview of the existing remote file storage solutions:

Feature	NFS	CIFS	AFS	Coda	Globus	Avaki
Remote access	Yes	Yes	Yes	Yes	Yes	Yes
Migratable on the same machine	#v.4	Yes	Yes	Yes	Yes	Yes
Migratable onto another machine	#v.4	No	Yes	Yes	Yes	Yes
Replicated	No	No	R / O	R / W	R / O	R / O
Self optimizing	No	No	No	No	Yes	Yes
Easy install	Yes	Yes	No	No	No	No
Compatible with existing software	Yes	Yes	Yes	Yes	No	No

Table II.1. File system core features on remote file storage solutions

All of these core features will be implemented in the SILENUS file storage system.

Architectural qualities for distributed systems

When designing a distributed system, several architectural qualities have to be satisfied. First, these qualities have to be identified. Existing solutions have to be investigated. Then possible solutions will have to be proposed.

Transparencies

A good distributed system should provide network transparencies. These transparencies are defined by ISO, however most applications do a poor job of providing all of them. To make SILENUS easy do use, all of these transparencies should be provided: [12] [56]

- Location transparent: it shouldn't matter where the file is stored
- Access transparent: all elements in the file store should be accessible from classical, non-SORCER programs.
- Replication transparent: there should be no difference on what replication the user works
- Failure transparent: the system should still work even if a significant number of hosts is down.

- Read concurrency transparent: multiple users should be able to read the same file at the same time
- Write concurrency transparent: multiple users should be able to write to same file at the same time
- Migration transparent: the system or the user should be able to migrate the physical presence of a file without interrupting any work.

Confidentiality

One of the most important features of any distributed file storage solution is confidentiality. Confidentiality here means that only authorized people are allowed to view the files stored in the system. In a distributed system this becomes even more important since files are stored on multiple systems. Even an administrator on one system should not necessarily be allowed to view all files stored on a particular device.

The term confidentiality is used in contrast to the usual term privacy. Privacy can have other meanings, where confidentiality is clearer in describing that only authorized people are able to view certain content.

Most of the existing file storage solutions check users credentials. Once a user is authenticated, she has full access to all her data. Unfortunately these credentials can be very often bypassed by administrators. Most systems allow administrators to impersonate any user on their system. While this is a good solution for single systems, where an administrator should have full rights, this can be a problem in a distributed system. Users may very often have administration rights on their personal work computer, but they should not be able to read data from other users on the same network.

Even if the user does not have administrative access, network ports are very often unsecured. In many cases, organizations provide network ports for guests, or students in the case of universities. These public ports can very often be used to listen into traffic on the network. A solution may be not to provide any public ports, but some of them might be outside of the organization: A user might want to access her data over the Internet, and there is no telling who could be listening.

Another security hole is direct access to the storage hardware. Even with no administrative rights, users can very often boot systems from an alternative medium and acquire administrative access. This can be prevented, however, there is currently no defense against someone physically taking a hard drive out of a computer. Making the hardware inaccessible is easily possible in large organizations. All that has to be done is

put the servers in a dedicated server room with security cameras and give out a limited numbers of keys to trusted personnel. All the data will be stored in the server room, no data will be stored on the users machines. Unfortunately this solution is impossible for smaller organizations. It also makes redundancy almost impossible to acquire.

Encryption solves the problem of confidentiality: Instead of storing data in so called plain format, the data is encrypted and then stored. To decrypt the data, a decryption key is needed. These keys are much smaller than the actual data. Current key sizes range from about 128 - 4096 bit. Storing a 4096 bit key takes up only 0.5 kilobytes of space and can safely encrypt several gigabytes of data. Sophisticated methods to secure encryption keys have been developed. Most common are pin-numbers, pass phrases and smart cards. [57]

There are two main types of encryption: symmetric and asymmetric encryption. Both have their advantages and disadvantages.

Symmetric encryption

In symmetric encryption the encryption and decryption key are the same. The main disadvantage is that no data can be encrypted without the decryption key present. So no one can leave data in the system for other people to read unless that person has access to the same key. Symmetric encryption therefore requires a lot of trust between involved parties. The main advantage of symmetric encryption is its speed. Symmetric encryption with short key length can be done very fast. The most widely used symmetric encryption algorithms are DES, blowfish and AES. DES and AES where standardized by the U.S. government for use in commercial applications. [7, 8]

Asymmetric encryption

In asymmetric encryption the encryption and decryption keys complement each other. Data can be encrypted with one key, and decrypted with the other. The main advantage here is that the encryption key can be made public: It is almost impossible to calculate the decryption key from the encryption key. This is by far more secure than symmetric encryption: The encryption key can be made public knowledge. Unfortunately asymmetric encryption is by far slower than symmetric encryption and requires longer key length. The most widely used asymmetric algorithm is RSA. [25]

Encrypting decryption keys

Both symmetric and asymmetric encryption can be combined: In current applications, each individual datafile is encrypted using symmetric encryption with a random encryption key. This encryption key is then encrypted using asymmetric encryption with the users asymmetric key. The encrypted symmetric key is then attached to the data file. This method combines the speed of symmetric encryption with the security of asymmetric encryption. It also allows files to be available to a group: The symmetric data key is simple encrypted with multiple asymmetric keys.

This combination has the advantage that a different symmetric key can be generated for every stored item. The encryption keys don't repeat, so a smaller size key can be used. If the encryption on a file is broken, one that one file will be compromised. Smaller keys allow for greater speed and flexibility.

The second advantage is that the secret asymmetric key can be physically carried by a user. It could be saved on a disk, USB key, smart card, or some other small device. This allows for the data to be encrypted on the users computer. It will not be sent unencrypted through a public network. It will never be decrypted on the computer responsible for the actual storage. Thus, administrators and eavesdroppers will not be able to view any data they aren't supposed to.

Existing cryptographic libraries

Instead of relying on a certain implementation, it is important to rely on a cryptographic library that has exchangeable algorithms. Cryptographic algorithms come and go. What is considered safe today may be considered flawed in the near future. To not have to worry about this the algorithms themselves should be exchangeable. Cryptographic libraries provide support for multiple algorithms. The most common used library for the language C is gcrypt. There are several libraries for Java. Fortunately Sun has developed a standard for Java cryptographic extensions (JCE). All cryptographic libraries based on JCE are exchangeable. [41, 9]

Global availability

In todays world, uses switch computers very frequently. A user may have a work computer and a home computer. However, the data should also be available at colleagues work computer, a friends computer, or at a computer in an Internet café half-way around the world. But not only full computer systems, but also smaller devices such as cell phones and PDAs are now connecting to the Internet. A users data should not only be restricted to the use of desktop computers, but should be available on any device anywhere.

In most cases, the users will not have the necessary administrative rights to install file system drivers. In some cases, like the home and work machine, this is no problem. But installing software in an Internet café is usually not possible. Therefore any file storage solution must be able to work with existing operating systems and applications.

Providing support to existing application is an important feature in remote file storage solutions. After all, it is very unpractical to store data and not being able to use it with existing software. Any new file storage solution should provide support for existing application by offering a support for as many operating systems as possible.

WebDAV

The Web Distributed Authoring and Versioning specification (WebDAV) provides a new standard for remote file storage. The name itself is ill chosen: WebDAV has nothing to do with the web, but rather with file storage over the Internet in general. It does not provide version information as the name suggests, but this is added by an extension called DeltaV.

So what does WebDAV specify? WebDAV extends the hypertext transfer protocol (HTTP) with file management function. The original HTTP specification provides support for authentication, uploading, and downloading of files. WebDAV provides additional functions for listing, moving, deleting, and locking files. This provides basic file management functionality. Two extensions to WebDAV provide support for versioning and more sophisticated access control lists (ACL). [3, 4, 6]

The WebDAV standard provides several option levels. Option level 1 provides basic functionality for upload, download and managing of files. Option level 2 provides support for file locking. The DeltaV and ACL extensions provide additional option levels. Each implementor may choose which option levels to actually implement.

WebDAV support is built into most modern operating systems: Windows and Mac OS X provide native support for WebDAV. Any WebDAV storage can be mounted and used (almost) like a local file system. Both GNOME and KDE provide very good support for data stored in WebDAV. All of these have to be looked at in detail:

Starting with Windows 98 all newer versions of Windows support WebDAV under the name "Web Folders". A WebDAV folder can be mounted like any other file system by going to "My Network Places", selecting "Add Network Place" and then typing in the address in the **http://server/folder** format. The WebDAV folder then appears like any other network folder on the system. Unfortunately files can not be edited directly on the server, they have to be copied to a local directory, edited and then uploaded again. Fortunately many software vendors implement WebDAV support directly into their applications. Among the most notably are Microsoft Office products and the Adobe Creative Suite.

At the time of this writing Mac OS X has the best built-in WebDAV support of all major operating systems. A WebDAV folder can be mounted like any folder in the Finder under Go / Connect to server. Mac OS X has full read-write support. WebDAV folders can be used like any other local drive.

The only shortcoming of Mac OS X is that the Mac OS file systems store a file in two parts: The actual file, and a so called "resource stream". This resource stream contains additional information, such as the file icon. On non HFS (The Mac OS native file system) file systems these resource streams are emulated with files that start with dot-underscore (`._`). Ideally, a file system driver should know about that and emulate the appropriate information.

Unix users which use the GNOME desktop are lucky: The standard file browser in GNOME is Nautilus, which supports WebDAV folders like any other folder. Simply type the address of a WebDAV folder in the address bar, and you can browse the files. Unfortunately you cannot open files directly, so you have to do the same as on Windows: Copy the file to a local directory, edit it, copy it back.

Cadaver is a very simple WebDAV client for all Unix systems. Its interface is the same as the standard command-line FTP client found on all Unix systems. This makes cadaver somehow tedious to use, but makes it highly portable. Use cadaver if you can't use any of the other methods.

Davfs2 is the project of building WebDAV support as a file system into the Linux kernel. Unfortunately, at the time of this writing this project was still in beta stage. [42]

Web-based access to file storage

A web application framework provides the infrastructure necessary to run applications over the Internet. Traditional web servers have support for static web pages only. Web applications however require interactive content. Some solutions work on the client. Client-side Java, Java script and Active-X are the most common examples. These solutions, however, require special support and software installed on the users computer. Other solutions run the application on the server. They provide a user interface by providing HTML pages and using HTML forms for interactivity. They may use client-side software, but do not require it. These solutions provide more security. Users do not need to run applications on their own machine. An example of such technology are Java Servlets and Java Server Pages.

Java Servlets and Java Server Pages (JSP) allow the provision of dynamic content on web pages. Traditional web pages are static and have to be manually updated on the server side. With server-side technology such as Servlet and JSP code can be executed whenever a website is requested. This enables dynamic web applications such as web shops. While static web pages can be protected by authentication, the pages served if authenticated are always the same. Dynamic web pages can provide different content to different users. They may also add special request and response codes to the web page. [50, 51]

Servlets were first thought of in 1995 by James Gosling. At a later time Pavani Diwanji picked up the concept and created Servlets that would eventually be part of the Java Web Server. The first Servlet specification was written by James Davidson. Java Server Pages were conceived by Anselm Baird-Smith, and later specified by Satish Dharmaraj in 1999. [52]

A Java Server Page is a shortcut version to a Servlet. Most Servlet just wanted to add a little dynamic content to an already existing web page instead of creating a whole new page. A JSP is a small part of Servlet code that is added in an otherwise valid HTML page. It is executed and it's results are added right there into the page. It is usually a good compromise between just code (Servlet) and just content (HTML).

The big advantages of Java Servlets and Java Server Pages are the dynamic nature and the large existing software library. Java Servlets allow dynamic content to be created. They may go from as little as just one line of code to reprogramming the HTTP protocol and adding new network commands. There are several solutions for dynamic web applications. JSP and Servlets, however where not just invented for dynamic web applications, and can therefore fall back on a large library of existing software packages. And since they are Java based they work on almost any web server platform.

As with all interpreted programming languages there is a performance loss. This may not be so significant on a single-user system but on a web page with millions of hits every day this is an issue. Fortunately the Java interpreter provides extensive run-time optimization with its Hot-Spot engine. But Java Servlets will always use more memory and CPU than native applications would.

Disconnected Operation

Ideally the Internet would be available everywhere on the world through a high speed connection. Unfortunately this is not the case yet. On the other hand, human expectations are more and more global. Data should be available everywhere whether connected to the network or not. Increasingly users want to use mobile devices, such as laptops. A distributed file storage system should have support for accessing files offline.

Even in places where the network is usually available there are still many network outages. Wired networks at any organizations fail at some point in time. In this case, a distributed file system should not loose any data. It should still provide support for saving and accessing cached files.

The first step to provide support for disconnected operation is to except disconnection. Many existing system make the assumption that the network is reliable, as stated in the section called “Eight fallacies of distributed computing.”. Instead, the exact opposite should be expected: Each machine works independent, and uses data from other machines if available. If not, it should carry on.

Each node will still have to collaborate with other nodes. They need to provide a synchronization mechanism. This synchronization mechanism should not depend on any global state, but rather detect automatically which state two nodes are in. It should then try its best to synchronize the data in the two nodes.

Sometimes disconnection is predictable. In this case, a distributed file system should provide support for hoarding. A user may decide to work on certain files at home. She plugs her computer in at work, selects files for offline work. After a while these files are made available on the users computer for offline usage. Whenever the user connects back to the network, the files are synchronized with the rest of the file system.

Manageability

As soon as a system grows larger or has been used for a while it becomes more difficult to manage. In the case of a file system this means many files, from many users, on many machines. There are several problems that arise here.

Managing many files means mostly migrating and replicating them among multiple machines. Files should be available on multiple machines for safety. They should be available on different machines to not overload a single machine.

A large base of users is another manageability challenge. Each user should have access to different files in the file system, and only to these files. User access rights have to be managed. This can not be done by one single person, there must be a way to delegate access rights to local administrators.

Machine failure and adding machines is a managing problem. When a machine fails, all the files that where on this machine will have to be moved to other machines. To do so, they should have been backed up or replicated to another machine beforehand. When a new machine becomes available, files have to be moved to this machine to actually utilize it.

One way to provide better manageability is to use federated services, as described in the section called “Service Oriented Architecture”. In a service oriented approach each machine provides services. Services are automatically discovered and used when they are available. These services can easily be moved from one machine to another.

Some of these federated services are autonomic optimizer services. These services can make the decisions a human administrator would make. They can check the current available resources and make sure they are used according to the policies set by an administrator. Since federated services are loosely coupled, different optimizer services can be added and removed based on the needs of a particular system.

Scalability

Another problem arising from a larger file system use is that of scalability. A system should still perform well, no matter how many machines, files, and users it serves.

Scalability can be achieved by distributing services across multiple machines. If a service is available on only one machine then this machine will eventually be overloaded. By making it possible to have services available on as many machines as needed, scalability can be provided by adding extra hardware.

A paradigm switch has to be made from client-server to federated services. Classical client-server solutions do not provide good scalability. They depend on a single server. As soon as the number of requests increases, so does the load on the server. Federated services, on the other hand, provide a way to load-balance the system. Instead of sending all requests through one server, the same functionality can be provided by many services. A requester can pick a service with a low load. Should all services be overloaded, an administrator can add extra machines.

Reliability

A quality that is particular important for file systems is reliability. A file saved into a file system should stay there until deleted. Files should never disappear or get lost. Unfortunately most existing file systems move the responsibility for reliability to the underlaying hardware. Should the hardware fail, the files are lost.

Reliability can be achieved by replication. In the case of a distributed file system this means replication among different machines. Every file that should be stored reliable needs to be available on at least two machines at any time. Should one machine fail, there is still another copy available. There should be another backup copy of that made as soon as possible to provide reliability again.

Modifiability

Software systems are never stable. They evolve into newer systems. There are two main reasons a software system needs to evolve: bug fixes and new features.

Every software has bugs. Software is written by humans, and humans make mistakes. Even the best computer scientists make mistakes [43]. So no matter how well a software is written and tested, it will always need to be updated to accompany new bug fixes.

After a while, users grow tired of an existing system and demand new features. Maybe a new device just came out, but the current computer system does not support it. Maybe the system is now used by different people who have a different focus and want different features. In these cases the system needs to be updated to add new features to it.

Dynamic code loading helps to provide modifiability. When an update is available in classical systems, an administrator has to manually download and install this update. This works well on a single machine, but is very hard to manage for multiple devices. It is even worse if there are multiple administrators, but a new version has to be rolled out immediately. With dynamic code downloading the software checks for a new version and downloads it whenever it starts. Rolling out a new version is as easy as publishing a file on a server. All that is needed is for the modified parts of the software to be reloaded. This may also be triggered from the network. With dynamic code downloading system-wide administrators can assure that all nodes have the latest version.

Platform independence

Existing computing devices use a wide variety of processors and operating systems. Supporting each of them with a custom solution is a major undertaking. An easier solution is using a virtual machine. An application would have to be written for that virtual machine. Only the virtual machine has to be ported to different platforms. The

programs are compile into byte code. This byte code can be reused on any of these virtual machines. This makes code mobility possible. The most commonly used virtual machines are the Java virtual machine and .NET.

An example virtual machine specification is the Java virtual machine (Java VM). Originally specified by Sun it is now being developed through a community process. Byte code that is compiled for a certain version of the Java virtual machine will run on any JVM that complies with these specifications. Example Java VM implementations are provided by Sun, IBM, Apple, and several open-source development teams. [10, 44, 45, 46, 47]

Originally intended to run on home appliances, the Java VM is now available on all modern desktop and server operating systems. The Java environment provides both an object-oriented language and a runtime system. The language is similar C++, which used to be the most widely used programming language. The runtime system provides the same functionality across all platforms. Java is a true write once - run everywhere language. Even modern mobile devices, such as personal digital assistants (PDA) and cell phones now provide support for the Java platform. In the heterogeneous environment of the Internet there is almost no way around a platform independent runtime system like Java. [48, 49]

Java also provides many built-in libraries. Unlike traditional programming languages, the Java standard requires a wide range of standard features. If a given Java runtime version is installed on a particular machine, all standard libraries will have be included.

Service Orientation

Service oriented architectures provide most of the given architectural qualities for distributed systems. It is therefore necessary to investigate service orientation and understand how it functions.

Eight fallacies of distributed computing.

To understand the motivation behind the service-oriented paradigm the common fallacies of network computing have to be investigated first. Peter Deutsch defined eight fallacies of network computing as follows: [55]

> Essentially everyone, when they first build a distributed application, makes the following eight assumptions. All prove to be false in the long run and all cause big trouble and painful learning experiences.
> 1. The network is reliable
> 2. Latency is zero
> 3. Bandwidth is infinite
> 4. The network is secure
> 5. Topology doesn't change
> 6. There is one administrator
> 7. Transport cost is zero
> 8. The network is homogeneous

In a service oriented system, none of the assumptions are made. Instead, it is always assumed that these eight points are false.

Service Oriented Architecture

Instead of thinking of a service offered by a particular host, the paradigm shift should be towards services in the network — *the computer is the network*. In classical distributed applications, it is necessary to know exactly on which host a particular service is exposed. In most distributed file systems, for example, it is necessary to know the name of a host that a particular file is stored on. In a service-oriented environment a service provider registers itself with a service registry. The service registry facilitates lookup of services. Once a service is found a service requester binds to the service provider and then can invoke its services. Requesters do not need to know the exact location of a provider beforehand, they can find it dynamically. They discover a registry and then lookup a service. On the other hand, a provider can discover the registry and publish its own service, as depicted in Figure II.1, “Service-Oriented Architecture”.

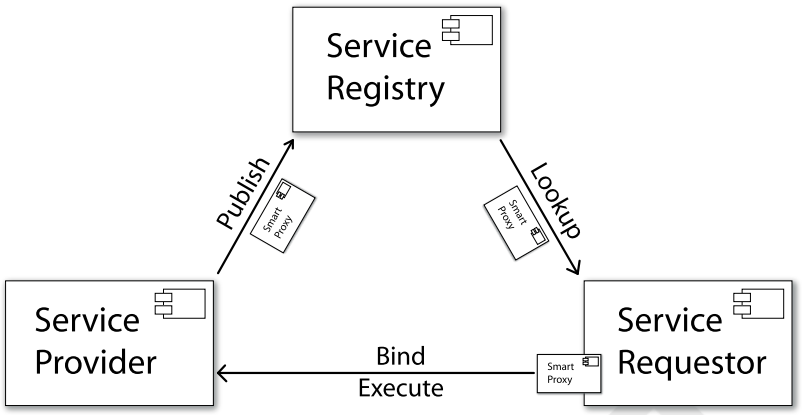


Figure II.1. Service-Oriented Architecture

A service is identified by an interface (type) rather than its implementation, protocol, or name. If a service provider registers by name, the requesters have to know the name of the service beforehand. Registering services by interface has the advantage that the actual implementation can be replaced and upgraded independently from the requesters. Different implementations may offer different features internally, but externally have the same behavior. This independent type-based identification allows for flexible execution of service-oriented programs in an environment with replicated services.

A service-oriented program is composed of tasks, jobs, and service contexts. Figure II.2, “Service oriented Tasks and Jobs” shows an example of service tasks and jobs. These concepts are defined differently than in classical grid computing. A service job is a structured collection of tasks and jobs. A task corresponds to an individual method to be executed by a service provider. A service context describes the data that tasks works on. This approach is different from classical grid computing, where a job corresponds to the individual method. In UNIX analogy the individual tasks correspond to UNIX programs and commands. The context would be the input and output streams. A job corresponds to a shell script or a complex command line connecting the tasks together. Service-oriented programs can be created interactively and allow for a federated service environment. [30]

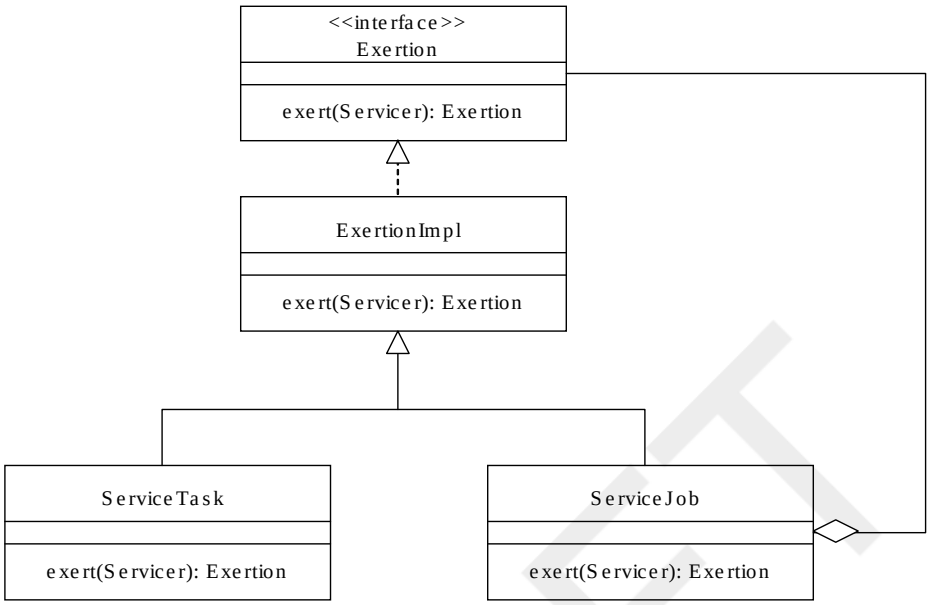


Figure II.2. Service oriented Tasks and Jobs

In a federated service environment not a single service makes up the system, but the cooperation of services. A service-oriented job may consist of tasks that require different types of services. Services can be broken down into small service methods instead of providing one huge all-embracing service. These smaller methods then can be distributed among different hosts to allow for reusability, scalability, reliability, and load balancing.

Instead of applying these grid concepts to computational tasks, they can, and should, also be applied to data. Once a file is submitted to the network it should stay there. It should never disappear just because a few nodes or the network segment goes down. Also, it should not matter what client node the file is requested from. With the SILENUS distributed file system in place, SORCER will also provide reliable and scalable file-based data services complementing the existing method services.

Jini Network Technology

The Jini network technology enables Java software to create dynamic networks that are adaptive to change. Jini uses a Service Oriented Architecture approach to network services. It is especially useful for scalability, evolvability and flexibility. Services can easily be replaced in runtime, started on multiple servers, or even migrated form one computer to another. [26, 58]

Jini technology was originally created by Sun. It was then contribute to the Jini Community in 1999. It is based on an open specification that can be developed through a community process. The reference implementation is provided still provided by Sun.

Jini provides almost everything necessary for service oriented computing, as described in the section called “Service Oriented Architecture”. Jini makes it easy to write services. Each service can register with a service registry. Service registries can be discovered by multicast announcements. Service requesters may use the service registry to find services and use them.

The dynamic nature of Jini is handled with leases. Each network service registering with another network service must obtain a lease. The lease must be renewed in given intervals or it will expire. This allows the detection of unreachable nodes, while putting the actual load on the requesting object, not the provider. Lease times may be adjusted depending on the stability of the network involved. A reliable network can work with higher lease times, while it is very desirable to have shorter leases in unreliable networks to detect disconnection quickly.

Jini also provides a standard to attach user interfaces to services. This ServiceUI standard allows the development of Jini service browsers. A Jini service browser will pick up all the registrars and display their services. If a service has an attached user interface the service browser can download and display that user interface to the user without having to install or configure any software locally. One example of such a service browser software is the IncaX Service Browser. [11, incax]

Peer-to-peer networking

Another network technology widely used for modern distributed architectures is peer-to-peer networking. In peer-to-peer applications each peer is equal. Peers communicate through an overlay network directly with each other. This eliminates the classical bottlenecks in client–server solutions.

Unfortunately peer-to-peer has a bad reputation. It was first widely used by the application "Napster". Users were able to share music files with other users in a fairly fast and reliable way. In the peer-to-peer architecture files are downloaded from other users rather than a central server. This makes peer-to-peer technology hard to control. It is therefore very often used to illegally distribute files. Some companies even want to ban peer-to-peer technology because of that. However, peer-to-peer also has many legitimate uses. Most Linux distributions are now released through peer-to-peer technology to save server capacity and increase download speed. Common peer-to-peer applications today include Gnutella, KaZaa, eDonkey, BitTorrent, and JXTA.

JXTA (short for Juxtapose) is a set of protocols that allow any device on the network to communicate and collaborate. JXTA provides an overlay peer-to-peer network that clients can use to communicate with each other. The JXTA protocols are defined language independent. A reference implementation for Java exists and is very stable. [54]

The JXTA project was originally started at Sun by Bill Joy and Mike Clary. The specifications and implementation where then made open-source and available on the JXTA web page.

JXTA focuses on peer-to-peer technology. Discovery in JXTA is made by the provider sending out service advertisements. These have to be sent out regularly for service requesters to find them. So called rendezvous peers can cache these advertisements. Once a requester has found a service advertisement it can use the JXTA overlay network to acquire a virtual channel between the requester and the provider. This channel can then be used to send messages back and forth.

JXTA is built for far distributed peers in an unstable network. A cached advertisement may provide a link to a service that has not been existent for a long time. It has therefore be actually attempted to use the service before any assumptions of its availability can be made.

When comparing JXTA an JINI the first distinction is the range of its application. JINI is designed for local area networks (LAN) and can be used over WANs with the use of special proxies. JXTA is designed for wide area networks (WAN) and all its network overlay is based on that. Fortunately these two can be combined: Jini requests can be sent over the JXTA network. This provides the best of both worlds: Fast, optimized local access and reliable remote access via the JXTA network. [27]

SORCER

SORCER is a federated S2S framework that treats service providers as network objects with a well defined semantics of service-object-oriented (SOO) programming based on the FIPER technology. [28, 29, 30]

Each SORCER provider offers services to other peers on the object-oriented overlay network. These services are exposed indirectly by methods in well-known public remote interfaces and considered as elementary (tasks) or compound (jobs) program instructions of SOO programming methodology [28]. A SORCER program can be created interactively [30] or programmatically (using SORCER APIs) and their execution can be monitored and debugged in the overlay network [31]. Service providers do not have mutual associations prior to the execution of a SOO program; they come together dynamically (federate) for all component tasks and jobs in the SOO program.

Each provider in the federation executes a task, or a job. A job is coordinated by a Jobber - one of SORCER infrastructure services [28]. However, a job can be sent to any peer. A peer that is not a jobber is responsible to forward the job to an existing jobber in the SORCER grid and return results to the requester. Thus, any peer can handle any job or task. Once the job execution is complete, the federation dissolves and the providers disperse and seek other SOO programs to join. Also, SORCER supports a traditional approach to grid computing - like in Condor [32] and Globus [33] style. Here, instead of SOO programs being executed by services providing a business logic for requested tasks, the business logic comes from the service requesters executable programs that seeks compute resources on the network provided by grid services. These services in the SORCER grid are as follows: GridDispatcher and Jobber for traditional grid job submission; Caller and Tasker for task execution. [34]

To integrate applications and tools on a B2B grid with shared engineering data, the File Store Service (FSS) [35] was developed as a core service in SORCER. The value of FSS is enhanced when both web-based user agents and service providers can readily share the content in a seamless fashion. The FSS framework fits the SORCER philosophy of grid interactive SOO programming, where users create distributed programs using exclusively interactive user agents. However FSS does not provide the S2S flexibility with separate specialized and collaborating service providers for file storage, replication, and meta information that are presented in this dissertation.

Eight truth of networked computing

Based on the fallacies given in the section called “Eight fallacies of distributed computing.” service oriented architectures take into account the following eight truth of distributed networking:

1. The network can fail at any time
2. Network messages arrive in random order
3. The network is always too slow
4. Someone is always listening
5. Machines get added and removed at any time
6. Every system has its own administrator
7. Moving data costs money
8. There will be any possible combination of OS / Architecture out there. They all want to be part of the network!

CHAPTER III. REQUIREMENT ANALYSIS

The requirements for the system need to be identified, before the actual system is designed. It needs to be clear which requirements must be met. After all, there is no point in developing a system that solves the wrong problem.

Most of the requirements have already been identified. the section called “File system core features” describes the features that are desirable in any file storage solution. the section called “Architectural qualities for distributed systems” described architectural qualities for distributed systems in general. To identify the exact requirements system usage patterns have to be investigated. Based on these user roles have to be identified.

Scenarios

To identify the requirements different scenarios have to be looked at first. Who would benefit from an advanced file system? Who would be using this file system and why. And what are the things that are important to this particular user group. Of course, in the real world, there will not be a scenario exactly as described here, but rather a mixture. But theses examples will still help to find the actual uses.

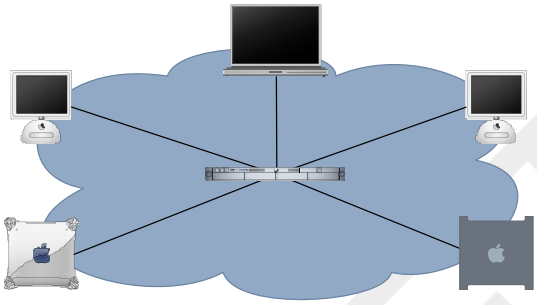
Small work group

I will start with the small work group because this is very easy do describe. Figure III.1, “Small work group” shows an example. In most cases there is one file server, one Internet-gateway (sometimes the same machine) and a small number of client machines (maybe five). Usually all client machines are either personal computers or shared machines. All data transfer is done via shared folders on the server.

This system lacks privacy. In most cases there is a shared folder on the server. All users can read and store files there. There is nothing holding back one user from deleting the file of another user.

There is also usually no or a tedious backup system. All work has to stop should the server fail. And in the case of an unrecoverable crash all previous work could be lost.

Last, but not least, the client machines are not used to their full potential. Most have extra hard drive space and, depending on the type of work, extra clock cycles. Some machines are turned off at night, but others are just running idle, using electricity and providing nothing for other users.



A typical small work group example. This work group has one server and five clients (four PCs, one laptop)

Figure III.1. Small work group

High Performance Computing Lab

A high performance computing lab is very similar to the small work group. The clients here are not idle, the distribution of CPU cycles is already taken care of. But many applications require a common data set. This is usually very large, and therefore not on every machine, but on one single server. Multiple machines (25, 50, 100, ...) are trying to get parts of the dataset at the same time. If not carefully planned, this performance leak can seriously reduce performance.

Large network

A large network is similar to the small network. But suddenly there are more than one server. Some of these might provide backup for other servers. A very important feature here is that users want to be able to log into a different computer, maybe even in a totally different location and still want to be able to access their files. Files should be stored as close to the user as necessary, for performance, but should also be migratable to other machines. Maybe two people from different location have to share common files. They should be able to share these files quickly.

Home user

The total opposite of the large network is the home user. The home user usually has very few computers. Maybe one desktop and a laptop, maybe two PCs for multiple people. Disk space is always low. Machines usually have very different performance features, I might be asked to move to the other computer because my brother wants to play a game. In the case of the home user transparent file access to as much disk space as possible is very important.

Concurrent Engineers

Distributed, concurrent engineering teams would greatly benefit from this system. They work at different physical locations, on different computer systems, with different computer architectures. However common data such as design documents, schedules, engineering data, notes, etc. have to be shared. The support for versioning will allow the team to go back to older versions, if necessary, but most importantly to ensure that the current version is available to all team members instantly. Data will always be downloaded from one of the hosts available. If a file is already available on a host in the local network this location will be preferred over a host at any remote location. This enables faster updates and ensures that slower WAN links are less used.

Student Computer Lab

A computer lab is a large array of computers. All computers should behave identically to the user, and offer the same file space. These lab systems usually use a central file storage server, which is a single point of failure. However, each lab host has a big hard drive nowadays, which is hardly used, if at all.

Astronomy

In a sky survey [36] the amount of data collected is very large. There must be some way to spread data files over multiple computers, or to make whole or partial files available to different users on different hosts. These files are usually associated with metadata. The metadata has to be kept in some kind of database to allow fast retrieval of the important data.

High energy physics

When the Large Hadron Collider (LHC) study of subatomic particles and forces at CERN will launch in 2007, it will be one of the greatest data management challenges. More than a gigabyte of data will be generated every second. This data will have to be distributed among researchers around the world. With these large amounts of data it is very important to prefer local replica over remote replica locations to minimize bandwidth usage. [37]

Machine types on the network

Based on these usage scenarios, machines participating in the network can be classified. Each machine type has different properties.

Server

Server machines are usually very reliable. They might have a RAID system, have fail over power supplies, multiple network interfaces, etc.. Server class computers are the easiest to use for administrators of distributed storage systems. One one system has to work, only one system has to be backed up. Unfortunately there also have to be client systems to make actual usage of the server.

Always up client

Administrators favorite client machines are the ones that are always up. These can easily be maintained remotely. They can be also be used to provide additional server features. However, not too many server features, since there is always a person wanting to work on the machine.

Work time up client

Work time up clients are usually on 40 hours a week. A person turns her personal computer on whenever she enters the office, and turns it off whenever she leaves. Most leave the machine running during lunchtime, but even that is uncertain. Usually these are personal machines, the user is concerned about speed of her own files, and feels the machine slow down if other people access data on the same machine.

Laptop

A laptop is the most complicated system to support when it comes to distributed file systems. Usually laptops are moved around from one network to another, connecting and disconnecting it from servers all the time. Fortunately laptop users are used to this, and therefore can be expected to specify which files they want to work on before they disconnect. But as soon as the laptop is connected to the Internet the laptop user wants to be able to access her files.

Mobile client

The last type of user is a special case of the laptop user, the so called mobile client. When talking about mobile, I mean small devices like personal digital assistants (PDA) and cell phones. These devices usually connect temporary to the network with a very low bandwidth. Users don't expect to have access to all data, but they do want to have certain files available, usually calendar, address book and notes.

Usage roles

Based on these usage scenarios different usage roles can be defined. These roles are: Regular file system users, administrators, optimizer services, service provisioners, and intergrid service providers.

File system users

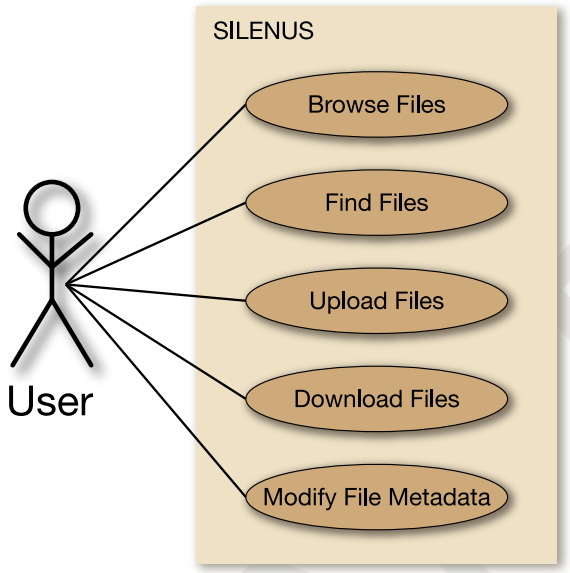


Figure III.2. Typical user cases for a file storage system

Figure III.2, “Typical user cases for a file storage system” shows the use cases that are identified for the regular user. These are typical tasks that can be executed on any existing file system.

Administrators

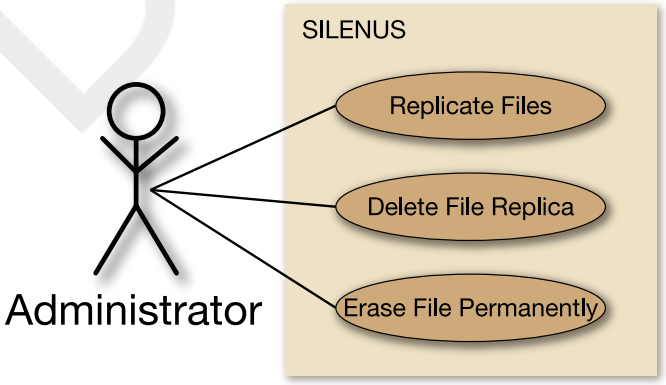


Figure III.3. Administrator use cases for a replicated file system

Figure III.3, “Administrator use cases for a replicated file system” shows the use cases for administrators. The administrator has the power to initiate all replication manually. If needed, administrators should be able to completely delete files.

Optimizer services

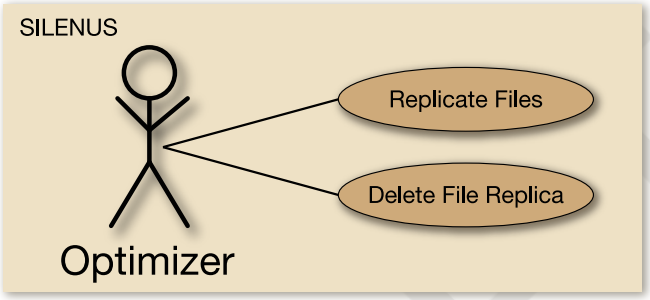


Figure III.4. Optimizer use cases for a replicated file system

To provide manageability the system should provide internal optimizer services. Figure III.4, “Optimizer use cases for a replicated file system” shows the use cases for these optimizer services. They have to be able to manage file replication by creating and deleting file replicas.

Service provisioners

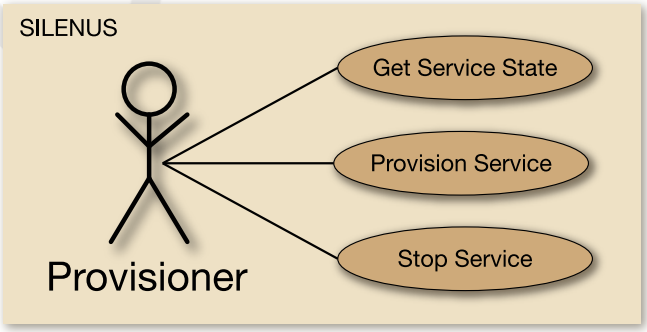


Figure III.5. Provisioner user cases for a replicated file system

The service provisioner is another type of optimizer service. As Figure III.5, “Provisioner user cases for a replicated file system” shows a provisioner has to be able to start (provision) and stop services in the network. To make the decision which services to start or stop it needs to be able to query the current state of each service.

Intergrid service providers

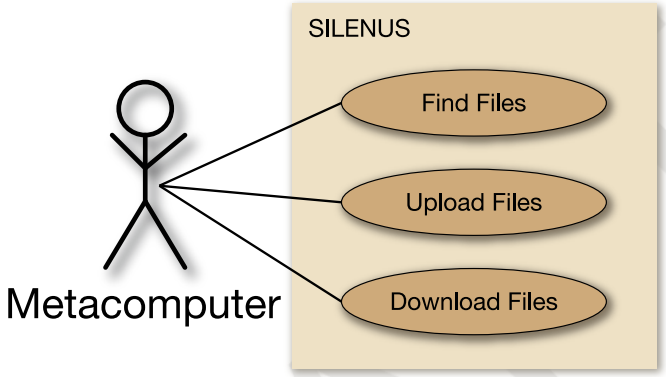


Figure III.6. Use cases for the intergrid meta computer

Another type of usage roles are intergrid service providers. These provide computing services, providing a meta computer. Figure III.6, “Use cases for the intergrid meta computer” gives an overview of the use cases required for meta computing. A service has to be able to find data files provided, download them, and upload them after it is done processing.

CHAPTER IV. DESIGN

A service-oriented approach is chosen to satisfy the given requirements. The system will be broken up into smaller components which will be implemented as services. Each service has a specific responsibility. Since all services are dynamic in nature, there is no specific deployment to any particular host. Each host can host none, one, some or all of the services. These services will use the SORCER network to communicate with each other.

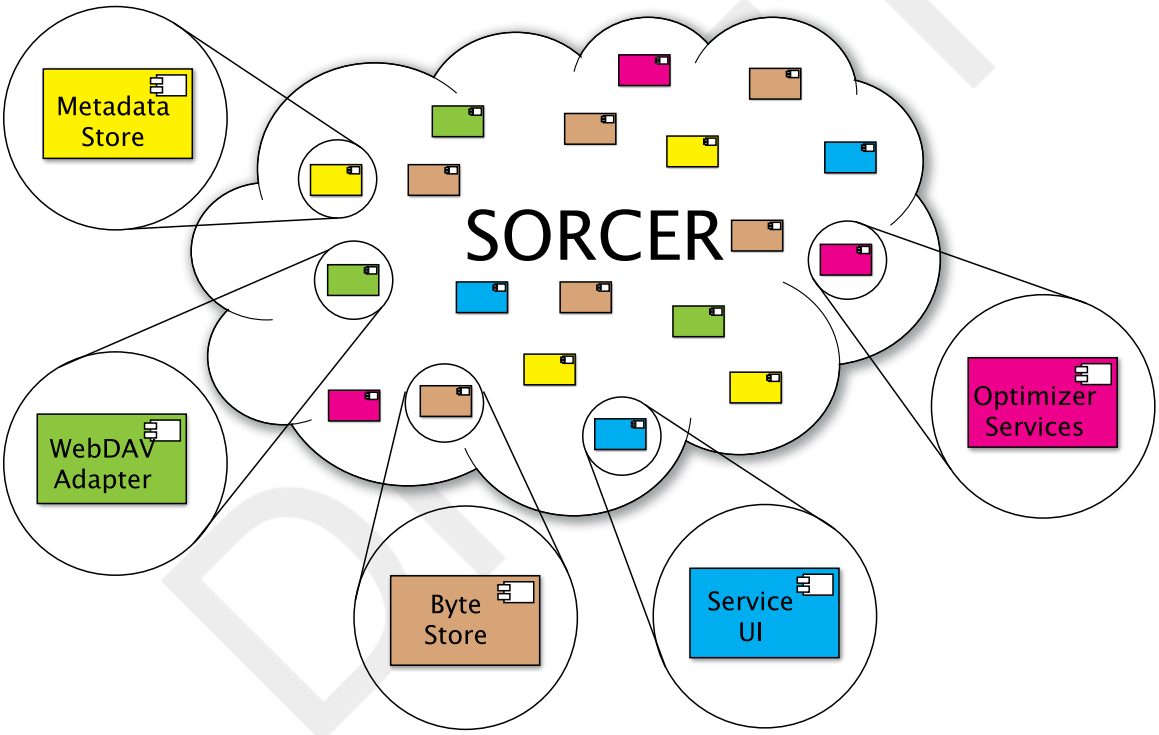


Figure IV.1. Components in the SORCER network

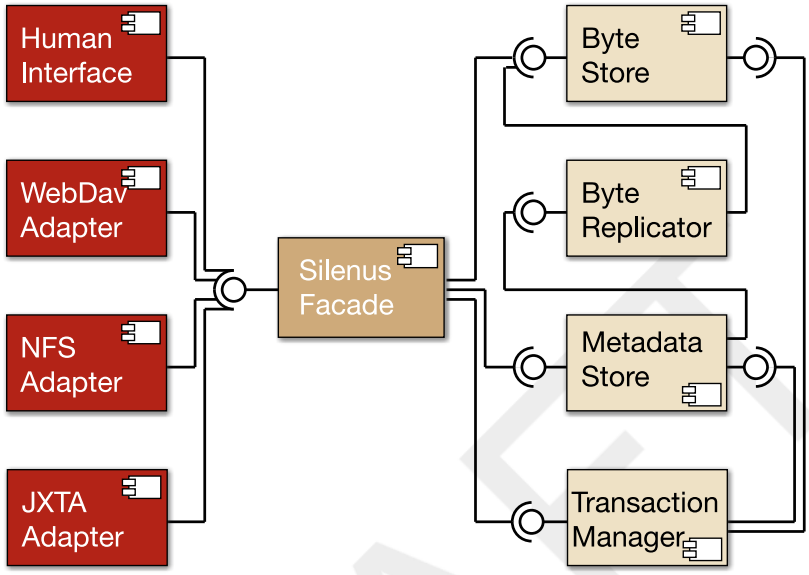


Figure IV.2. SILENUS components

Figure IV.2, “SILENUS components” gives an architectural overview over the SILENUS system. It shows the SILENUS components and the how they interact with each other. The individual parts are from the left to the right:

The leftmost components provide external interfaces with existing systems. The ones given here are just examples, adapters could be written for any other existing file storage solution. The human interface (ServiceUI) provides support for file storage and management through a proprietary user interface. It provides access to the extra features which are not available through the other interfaces: Advanced features such as manual migration, number of replicas, log-file viewing, etc.. The service interface should only be needed for these extra features and can be ignored by most users. The WebDAV adapter provides support for existing applications, as explained in the section called “WebDAV”. This gives current operating systems the possibility to use the file storage without having to install a client. A NFS adapter provides support for older Unix systems. The content management service (CMS) interface is a standard used for file sharing across the JXTA network.

The gateway to the SILENUS file storage is the SorcerFileStore interface, provided by the SILENUS facade component. This component provides a facade to the underlaying services. It takes care of transactional semantics between file and meta information storage. It provides one easy interface for the user.

The SorcerMetadataStore interface, implemented by the metadata store service provides support for the storage of file metadata. File metadata is all the information that is either included in the actual file data or that can be derived from the file data, such as file name, creation date, file type, type of encryption, etc.. Information on where the actual file is located is also stored in here. Multiple versions of one file may exist in the database for recovery purpose.

The SorcerByteStore interface, implemented by the byte store services provides support for the storage of the actual file data. It provides fast access to the files stored on the provider's machine. Files are usually stored encrypted, but can be unencrypted for performance reasons

The TransactionManager interfaces, provided by Jinis Mahalo service is needed to coordinate file uploads between the metadata store and the byte store. Only if both succeed should the file actually be available.

The SorcerOptimizer interface provides support for network optimizer services. One example of these services is the ByteReplicator service. It will make sure that uploaded files are replicated among different byte store nodes to provide redundancy. Optimizer services can request log information from the storage providers, and can automatically initiate replication and migration. They can detect usage patterns and make sure that the files are available to the user. They can detect non-responding systems and automatically replicates all files that were stored on it. They also makes sure that all storage servers have the latest version of the files.

After this overview over the services and their interactions the individual services can now be looked at in more detail.

Service user interface

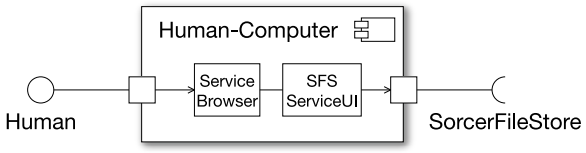


Figure IV.3. Component diagram for the user interface

To work with the file system, users need an interface. None of the compatibility interfaces can provide access to all of SILENUS capabilities. Therefore and additional user interface is provided.

The user interface is dynamically downloaded when needed. Unlike traditional systems that require installation on a client computer, SILENUS user interface is dynamic. The users needs to have a service browser installed. This service browser can detect services running in the network. It can then download and display these provided user interfaces. There is no actual configuration needed on the client computer.

WebDAV adapter

Figure IV.4. Component diagram for the WebDAV adapter

The WebDAV adapter provides the connection from existing applications and file systems to the SILENUS file storage system, as shown in Figure IV.5, “The WebDAV adapter”.

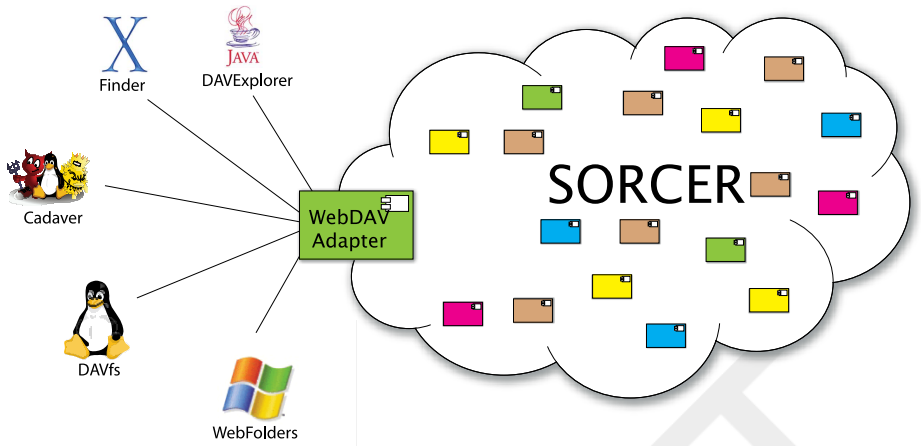


Figure IV.5. The WebDAV adapter

The WebDAV adapter uses Java Servlet technology to handle requests instead of rewriting a complete new server software. WebDAV is based on HTTP, as explained in the section called “WebDAV”. Therefore, existing application servers that handle HTTP can be reused to provide a WebDAV server. One of these technologies is Java Servlets, as explained in the section called “Web-based access to file storage”. The Servlet standard provides functionality for handling HTTP requests with the HttpServlet interface. It is very easy to add the additional functionality required for WebDAV.

Incoming WebDAV requests will have to be mapped to the appropriate file store requests. Most requests are straightforward: GET and PUT will be implemented using the upload and download functions. PROPFIND uses the request node info, and PROPPATCH will set node info. LOCK requests can be ignored, but need to be handled internally to provide consistency.

The implementation and details of the WebDAV adapter is a pending master thesis topic for Fajin Wang.

File store

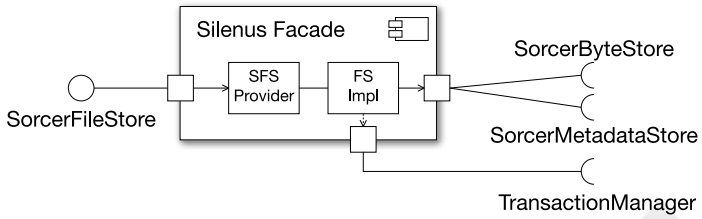


Figure IV.6. Component diagram for the SILENUS facade

The SORCER File Store interfaces provides a facade to the SILENUS network for clients that want to use the system. Since the metadata and actual file contents are stored in different services there is need to coordinate between these two services. To make use of the file system easier this functionality is combined in the SILENUS facade with the File Store interface.

Most file store functionality is very straightforward and just consists of forwarding a request to the appropriate service. Actions like retrieving file metadata or setting file metadata can be directly forwarded to a metadata store.

File download has to be coordinated between two services: The file metadata has to be retrieved from the metadata store. This metadata contains information about the byte store that carries the file contents. A connection has to made with that particular byte store.

File upload requires the use of transactional semantics. When a file is to be uploaded, two things have to be created: A new node in a metadata store, and the file data has to be uploaded to a byte store. To save time both requests can be started in parallel. However, it is very important that, should one of them fail, the other one is cancelled. Figure IV.7, “File upload transactional semantics” shows this transactional semantics.

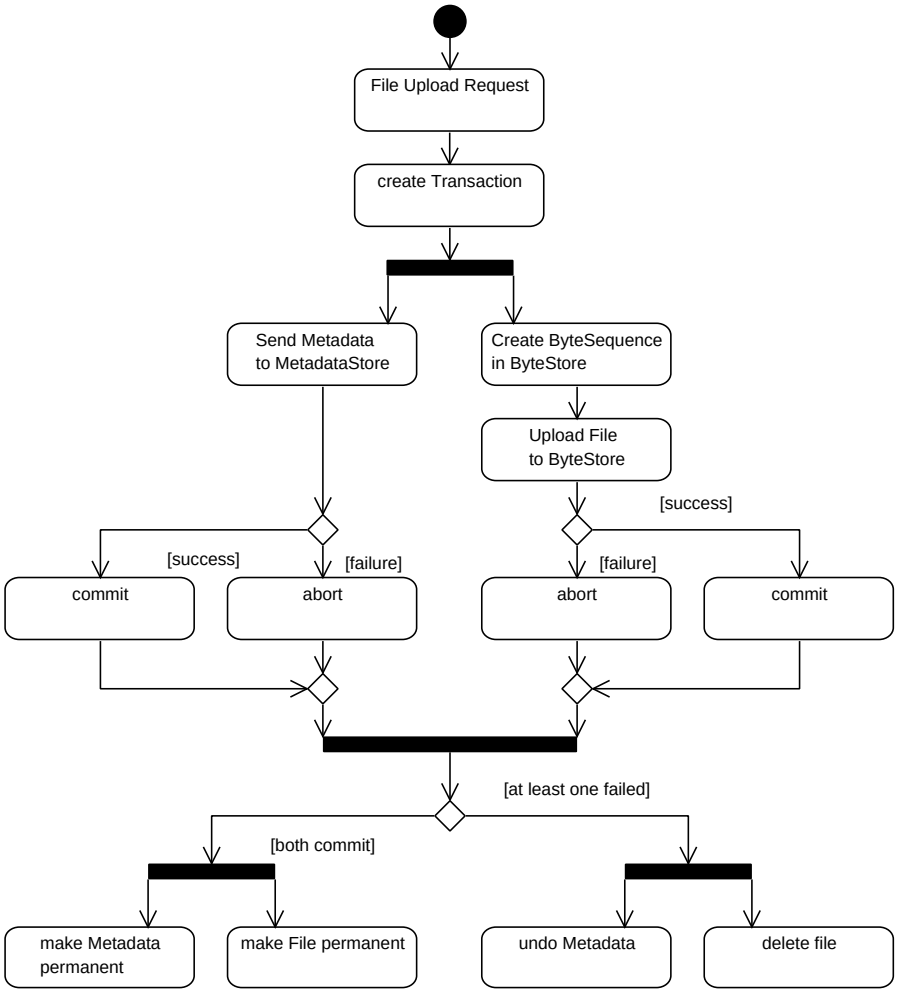


Figure IV.7. File upload transactional semantics

To support the transactions a separate transaction service is needed. Fortunately Jini already provides a standard for the Transaction Manager interface. It also provides a reference implementation, called Mahalo, that implements this interface. The SILENUS facade can use either this or any other service that provides transactions to ensure that both operations succeed.

Metadata store

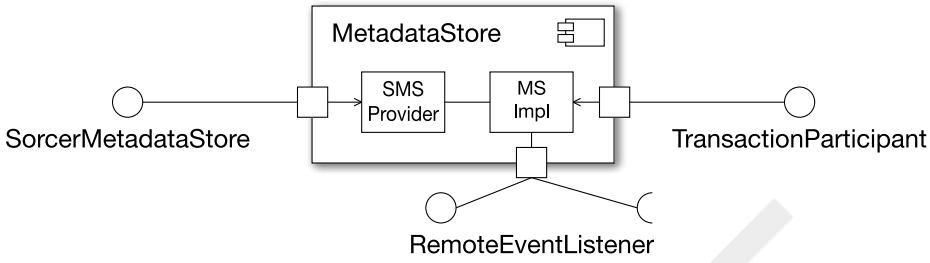


Figure IV.8. Component diagram for the metadata store

The file meta information is stored in key-value pairs for each file. The key describes the kind of attribute (e.g. file name, creation date), where as the value describes the value of the attribute.

There are two types of file attributes. Basic attributes are of type string or are easily represented in string form. Extended attributes can be any Java object. This distinction is necessary when retrieving file attributes: Instead of having to choose a list of attributes, a client can chose to get either just the basic attributes or all attributes. This makes look-ups for basic attributes fast, but does not limit the attribute types.

The two attributes parent and mime type are used to create the well-known hierarchical file system structure. Every node except for the root directory has exactly one parent node. The mime type describes the type of the file. A special mime type is used for directories.

The metadata store meta information is needed for metadata store synchronization. The metadata store needs to keep track of which file versions it has and when the last synchronization has occurred.

As in internal database an embedded database is chosen. Using an embedded database makes installation much easier, it does not require the installation of external database software. The database access itself is done using the data access object pattern to extensibility and support for other databases if needed. A high performance computing lab, for example, could set up commercial database software to increase performance.

Byte store

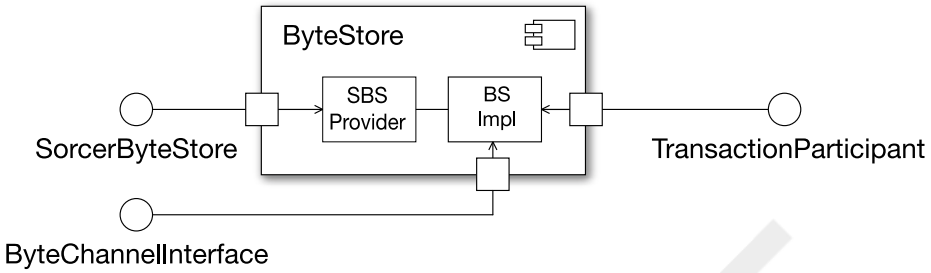


Figure IV.9. Component diagram for the byte store

The byte store service stores the actual file data. In the analogy of hardware this would be the actual hard drive.

Files in a byte store are identified uniquely by the ID of the byte store and an entry ID in the byte store. These ID numbers never change. This makes the file storage independent from file metadata such as the file name. The byte store services provides nothing but support for file storage. The advantage is that this service can be then optimized for performance. Adam Turner is currently working on his master thesis investigating potential performance optimization using a BitTorrent like file distribution.

Unlike the metadata stores the byte stores are not synchronized. File data is much larger than file metadata. Would the file data be replicated on every node the storage capacity would be filled very quickly. It is the job of the optimizer services to provide file data replication.

Optimizer

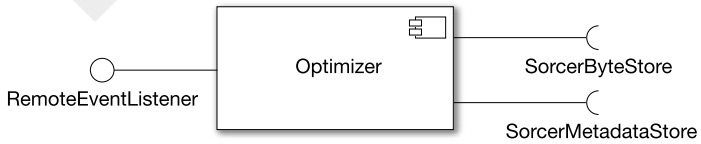


Figure IV.10. Component diagram for the optimizer

The optimizer services keep the network in good shape. There can be many different optimizer services that provide different optimizations.

One example service is the ByteReplicator optimizer service. This service gets triggered when a new file or a new version of a file is uploaded to the file system. It will then look for another byte store that has enough storage space. It tells the other byte store to replicate the file. After the file is replicated, it will update the metadata stores to have the new location information. This ensures reliability by providing multiple copies. But not only new files can trigger replication. If a byte store service becomes unavailable, all files that where stored on that services are potential candidates for replication: They may now exist in the network only once, not providing reliability. In this case, the ByteReplicator has to trigger another replication.

Another type of optimizer services is an autonomic provisioner. When the file system becomes full, the provisioner may start more byte store services. When the file system is sparsely used, these byte store services may be shut down. When the metadata stores and the SILENUS facade get to many requests, the provisioner may start provision new services in the network. When the number of requests go down, the provisioner may stop these services.

CHAPTER V. VALIDATION

TBD

CHAPTER VI. CONCLUSION

TBD

BIBLIOGRAPHY

NORMATIVE DOCUMENTS

[1] *RFC 1094*. Sun Microsystems. “NFS: Network File System Protocol specification”. IETF. 1989. http://www.ietf.org/rfc/rfc1094.txt.

[2] *RFC 1813*. B. Callaghan, B. Pawlowski, and P. Staubach. “NFS Version 3 Protocol Specification”. IETF. Jun 1995. http://www.ietf.org/rfc/rfc1813.txt.

[3] *RFC 2518*. Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. “HTTP Extensions for Distributed Authoring – WEBDAV”. IETF. 1999. http://www.ietf.org/rfc/rfc2518.txt.

[4] *RFC 3253*. G. Clemm, J. Amsden, T. Ellison, C. Kaler, and J. Whitehead. “Versioning Extensions to WebDAV (Web Distributed Authoring and Versioning)”. IETF. 2002. http://www.ietf.org/rfc/rfc3253.txt.

[5] *RFC 3530*. S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. “Network File System (NFS) version 4 Protocol”. IETF. 2003. http://www.ietf.org/rfc/rfc3530.txt.

[6] *RFC 3744*. G. Clemm, J. Reschke, E. Sedlar, and J. Whitehead. “Web Distributed Authoring and Versioning (WebDAV) Access Control Protocol”. IETF. 2004. http://www.ietf.org/rfc/rfc3744.txt.

[7] “Data Encryption Standard (DES)”. *FIPS PUB*. 46. U.S. Department of commerce. National Institute of Standards and Technology. Jan 1977.

[8] “Announcing the Advanced Encryption Standard (AES)”. *FIPS PUB*. 197. National Institute of Standards and Technology. Nov 2001.

[9] *Java Cryptography Extension (JCE) for the Java 2 SDK, v 1.4*. http://java.sun.com/products/jce.

[10] *The Java™ Virtual Machine Specification (2nd Edition)*. Apr 99. Addison-Wesley Professional. Tim Lindholm and Frank Yellin. 0201432943.

[11] *The ServiceUI API Specification, v. 1.1a*. Jun 2005. Bill Venners. http://www.artima.com/jini/serviceui/Spec.html.

[12] *Open Distributed Processing*. Reference Model. 10746. ISO/IEC. 1995.

ARTICLES

[13] Paul Leach and Dan Perry. “CIFS: A Common Internet File System”. Nov 1996. *Microsoft Interactive Developer*.

[14] Richard Sharpe. *Just what is SMB?*. Oct 2002. http://samba.org/cifs/docs/what-is-smb.html.

[15] M. Satyanarayanan. “Coda: a highly available file system for a distributed workstation environment”. 114–116. http://ieeexplore.ieee.org/iel5/267/3322/00109279.pdf. *Workstation operating systems: proceedings of the Second Workshop on Workstation Operating Systems (WWOS-II), September 27--29, 1989, Pacific Grove, CA*. IEEE Computer Society Press. 1989. 0-8186-2003-X. 0-8186-5003-6 (microfiche).

[16] M. Satyanarayanan. “Coda: A Highly Available File System for a Distributed Workstation Environment”. Jul 1999. http://www.cs.cmu.edu/afs/cs/project/coda/Web/docdir/wwos2.pdf.

[17] Ann L. Chervenak, Bill Ahcock, Carl Kesselman, Darcy Quesnel, Ian Foster, Joe Bester and John Bresnahan, Sam Meder, and Steven Tuecke and Veronika Nefedova. “Data Management and Transfer in High-Performance Computational Grid Environments”. Sep 2002. http://www.globus.org/research/papers/dataMgmt.pdf.

[18] Asad Samar, Bill Allcock, Brian Tierney and Heinz Stockinger, Ian Foster, and Koen Holtman. “File and Object Replication in Data Grids”. Sep 2002. http://www.globus.org/research/papers/FileRepCluster02.pdf.

[19] Gurmeet Singh, Shishir Bharathi, Ann Chervenak, Ewa Deelman, Carl Kesselman, Mary Manohar and Sonal Patil, and Laura Pearlman. “A Metadata Catalog Service for Data Intensive Applications”. http://www.sc-conference.org/sc2003/paperpdfs/pap242.pdf. *SC2003: Igniting Innovation. Phoenix, AZ, November 15--21, 2003*. ACM Press and IEEE Computer Society Press. 2003. 1-58113-695-1.

[20] Anand Natrajan, Marty A. Humphrey, and Andrew S. Grimshaw. “Grids: Harnessing Geographically-Separated Resources in a Multi- Organisational Context”. 2001.

[21] *Peer-to-Peer Computing*. Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Mar 2002.

[22] *The Legion Grid Portal*. Anand Natrajan, Anh Nguyen-Tuong, Marty A. Humphrey, and Andrew S. Grimshaw. 2002.

[23] *Symphony A Java-based Composition and Manipulation Framework for Computational Grids*. Markus Lorch. Jul 2002.

[24] *Multivariate Minimization Using Grid Computing*. Kandle Kulish, Jerry Perez, and Phil Smith.

[25] R. Rivest, A. Shamir, and L. Adleman. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. *Communications of the ACM*. 21 (2). 120 - 126. 1978.

[26] *Build a Compute Grid with Jini™ Technology*. Dec 2004. http://www.jini.org/whitepapers/JINI_ComputeGrid_WP_FINAL.pdf.

[27] *Beyond Web Services*. Combining Jini™ Network Technology and "Project JXTA" to Take Advantage of Edge Computing. *JavaOne, Sun's 2003 Worldwide Java Developer Conference*. Carlos Queiroz, Bruno Souza, and Einar Saukas.

[28] Michael Sobolewski. "Federated P2P Services in CE Environments". 13–22. *Advances in Concurrent Engineering*. A.A. Balkema Publishers. 2002. 90-5809-502-9.

[29] Michael Sobolewski. "FIPER: The Federated S2S Environment". *JavaOne, Sun's 2002 Worldwide Java Developer Conference*. 2002. http://servlet.java.sun.com/javaone/sf2002/conf/sessions/display-2420.en.jsp.

[30] R. Kolonay and Michael Sobolewski. "Grid Interactive Service-oriented Programming Environment". 97–102. *Concurrent Engineering: The Worldwide Engineering Grid*. Tsinghua Press and Springer Verlag. 2004. 7-302-08802-0.

[31] Sekhar Soorianarayanan and Michael Sobolewski. 89–95. "Monitoring Federated Services in CE". *Concurrent Engineering: The Worldwide Engineering Grid*. Tsinghua Press and Springer Verlag. 2004. 7-302-08802-0.

[32] Douglas Thain, Todd Tannenbaum, and Miron Livny. "Condor and the Grid". *Grid Computing: Making The Global Infrastructure a Reality*. John Wiley. Fran Berman. Anthony J.G. Hey. Geoffrey Fox. 2003. 0-470-85319-0.

[33] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. "Grid Services for Distributed System Integration". *Computer*. 35. 6. 37–46. Jun 2002. 0018-9162. http://csdl.computer.org/dl/mags/co/2002/06/r6037.pdf.

[34] Vivek Khurana, Max Berger, and Michael Sobolewski. "A Federated Grid Environment with Replication Services". *Next Generation Concurrent Engineering*. Omnipress. 2005. 0-9768246-0-4.

[35] Michael Sobolewski, Sekhar Soorianarayanan, and Ravi-Kiran Malladi Venkata. 633–639. "Service-Oriented File Sharing". *CIIT conference (communications,internet and information technology)*. Nov 2003.

[36] Robert Lupton, F. Miller Maley, and Neal Young. "Data Collection for the Sloan Digital Sky Survey–-A Network-Flow Heuristic". *Journal of Algorithms*. 27. 2. 339–356. May 1998.

[37] Eva Arderiu Ribera. "LHC Distributed Data Management". Nov 1998. http://wwwinfo.cern.ch/asd/rd45/papers/proc_108.ps.

INTERNET RESOURCES

[38] *OpenAFS*. http://www.openafs.org.

[39] *Globus Alliance*. http://www.globus.org.

[40] *Sybase Avaki EII*. http://www.sybase.com/products/developmentintegration/avakieii/distributedarchitecture.

[41] *Libgcrypt*. http://www.gnupg.org.

[42] *WEB-DAV Linux File System(davfs2)*. Sung Kim. http://dav.sourceforge.net/.

[43] *Knuth reward check*. http://en.wikipedia.org/wiki/Knuth_reward_check.

[44] *Java Technology*. Sun Microsystems. http://java.sun.com/.

[45] *Java technology*. IBM. http://www-128.ibm.com/developerworks/java.

[46] *Java for Mac OS X*. Apple Computer. http://www.apple.com/macosx/features/java/.

[47] *Kaffe.org*. http://www.kaffe.org/.

[48] *Java Technology*. http://www.java.com.

[49] *Java 2 Platform, Micro Edition (J2ME)*. http://java.sun.com/j2me.

[50] *JavaServer Pages Technology*. http://java.sun.com/products/jsp/.

[51] *Java Servlet Technology*. http://java.sun.com/products/servlet/.

[52] *Jim Driscoll's Blog*. Servlet History. Jim Driscoll. http://weblogs.java.net/blog/driscoll/archive/2005/12/servlet_history_1.html.

[53] Phil Bishop. *IncaX*. http://www.incax.com.

[54] *JXTA*. http://www.jxta.org/.

[55] *The Eight Fallacies of Distributed Computing*. Peter Deutsch. http://today.java.net/jag/Fallacies.html.

BOOKS

[56] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall. Jan 2002. 0130888931.

[57] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition*. Wiley. Oct 1995. 0471117099.

[58] Jan Newmarch. *A Programmer's Guide to Jini Technology*. Apress. Nov 2000. 1893115801.

COLOPHON

This thesis was written with the XMLmind XML editor using docbook xml 4.4. It was translated using a customized version of the docbook-xsl stylesheets 1.69.1. The translation was done using the xalan processor 2.7. The typesetting was done using fop svn-jan-06. Automation was provided by ant 1.6.5.