SILENUS - A federated service-oriented approach to distributed file systems

Max Berger & Michael Sobolewski
SORCER Research Group, Texas Tech University

A new approach for a distributed file system SILENUS based on the federated service-oriented computing environment SORCER is presented. Instead of services running on specific computers, the system is build from federating services in the network. The services are: byte-store service (Byzantium) for the storage of the actual date; metadata-store service (Midas) for information about the files; WebDAV adapter (Daphne) for support for legacy applications and operating system; and optimizer services that keep the collaborating federations in a healthy state. This system implements file-based data grid services in the SORCER environment that are needed for collaboration of distributed teams.

1 INTRODUCTION

Ever since the introduction of a personal computer there exists the problem of managing distributed data. This data is usually stored on one host and not easily accessible for other users or from other hosts.

The current trend goes away from a personal computer back to a networked one. We want to work anywhere, anytime, and of course always have the same consistent view on the same data available.

It gets even worse when it comes to collaboration in larger, and distributed engineering teams. Data files have to be accessed in real-time at different locations. Versions have to be managed in a consistent manner. There must be support for offline-use of data and for concurrency management.

Several systems exist to access data that is spread across multiple hosts. However, except for a few exceptions, all of them require manual management and knowledge of the exact data location. Very few offer features like local caching or data replication.

In an integrated environment, all entities must first be connected, and they then must work cooperatively. Services that support concurrency through communication, team coordination, information sharing, and integration in an interactive and formerly serial product development process provide the foundation for a CE environment. Product developers need a CE programming and execution environment in which they can build programs from other developed programs, built-in tools, and knowledge bases describing how to perform a complex design process. Like any other services in the environment, a CE distributed file system can be structured as a collection of collaborating distributed services enabling for robust, secure, and shared vast repository of engineering data.

Under the sponsorship of the National Institute for Standards and Technology (NIST) the Federated Intelligent Product Environment (FIPER) (**?**, b, **?**) was developed (1999-2003) as one of the first service-to-service (S2S) grid computing environments. The Service-Oriented Computing Environment (SORCER) (**?**, **?**, **?**) builds on the top of FIPER to drastically reduce design cycle time, and time-to-market by intelligently integrating elements of the design process by providing true concurrency between design and manufacturing. The systematic and agile integration of humans with the tools, resources, and information assets of an organization is fundamental to concurrent engineering (CE).

In this paper, we discuss a novel approach to share data across multiple service providers using dedicated storage, metainformation, replication, and optimization services in SORCER. The access via WebDAV (**?**) adds to the idea of heterogeneous interactive programming, where the user through its diverse operating system interfaces can manage shared data files and folders. The same data can be accessed and updated by different service providers and authorized users can monitor data processing activities executed by the service providers involved with co-operating WebDAV user agents. Like any other services in the environment, the SILENUS services are also peers in the SORCER network.

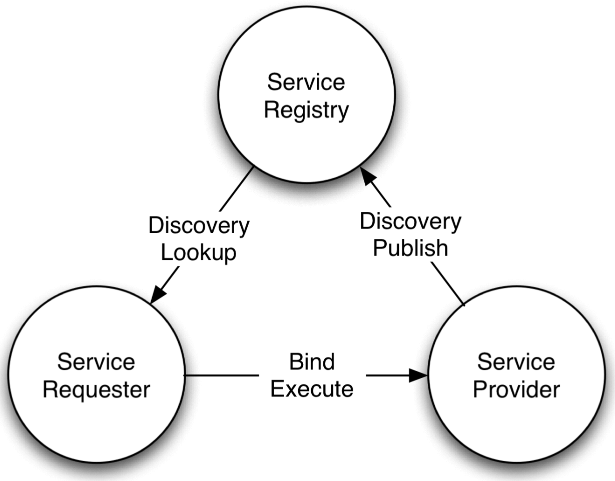

Figure 1: Service-Oriented Architecture

1.1 Service-Oriented Computing

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 1.

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. 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 (**?**).

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-in-one 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.

1.2 Scenarios

Who would benefit from an advanced file system? From many possible scenarios four are described below.

CE: 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.

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 (**?**) 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. (**?**)

2 BACKGROUND REVIEW

2.1 Distributed Filesystems

To compare SILENUS with existing distributed file storage solutions, several file system features can be considered. These features are then examined with respect to their implementation in the Andrew File System (AFS) (**?**), Coda (**?**) (**?**), and file storage in Globus (**?**, **?**, **?**, **?**).

The existing file systems have a fixed transport protocol. Globus is based on FTP. AFS and Coda use UDP packets and implement their own transport management. The transport in SILENUS is protocol independent. It is based on Jini (**?**, **?**) endpoints (JRMP, JERI, HTTP, HTTPS, SSL, . . .). This makes it configurable without recompilation. It even supports protocols that were not available during development. Thus it can easily be adapted to firewalls and other network specific configurations.

AFS and Coda use UDP packets for better performance than TCP streams. Globus uses TCP streams for better reliability than UDP packets. Since SILENUS is transport independent, it can be tuned for the individual needs accordingly.

The security in Globus, AFS, and Coda is based on Kerberos (**?**). Kerberos is a proven system for authentication. It does not directly provide any other security features, such as confidentiality. Data is still transferred as plaintext, thus allowing everyone in the network to read all transferred data. SILENUS stores file encrypted and decrypts them on the user's computer. Network listeners can observe that a file is transferred, but are unable to read the actual content.

In most file systems, including AFS and Coda, there is no explicit separation of file metadata and file content. To get information about a file, the file has to be found first, and then the information about the file can be read. Globus as well as SILENUS separate file information into file content and information about the file (metadata). This allows for additional types of metadata to be defined, thus describing files more precisely. This additional metadata can be extracted from the file content once and then queried and accessed efficiently.

Among the existing file systems only Coda allows for disconnected operation. AFS and Globus will fail if the network fails, there is no proper recovery mechanism. This makes these filesystems impractical for mobile computing, for example using laptops or mobile phones. SILENUS is designed to handle unexpected network disconnections efficiently.

To support existing applications, a file system needs native support within operating systems. For automatic update in Globus, a file has to be downloaded, then needs to be modified, and has to be uploaded again. AFS and Coda, as well as SILENUS provide a file system driver for most common operating systems.

The usage of storage on AFS and Coda servers has to be optimized manually. Globus introduced an optimizer service that tries to move data for better performance. SILENUS allows for several autonomic optimizer services that can be tailored to a specific environment.

The major difference between SILENUS and the other systems is in the distributed architecture. While the existing file systems are based on a client-server architecture, SILENUS is based on a federated service-to-service architecture. Usually systems need to be configured for each specific environment. In particular, addresses of server machines have to be set on every client, and changed every time the network topology changes. The discovery mechanisms in SILENUS make these configuration issues dynamic and on-the-fly.

2.2 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 (**?**, b, **?**).

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 (**?**). A SORCER program can be created interactively (**?**) or programmatically (using SORCER APIs) and their execution can be monitored and debugged in the overlay network (**?**). 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 SOR-

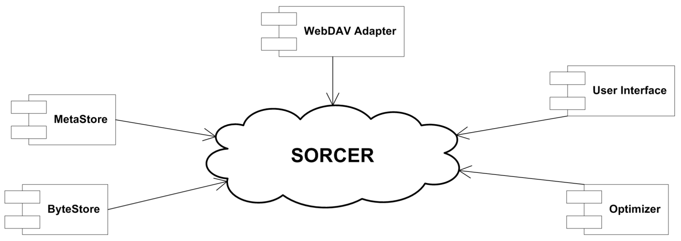

Figure 2: The SILENUS conceptual architecture

CER infrastructure services (?). 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 SSO programs to join. Also, SORCER supports a traditional approach to grid computing - like in Condor (?) and Globus (?) style. Here, instead of SOO programs being executed by services providing a business logic for requested tasks, the business logic comes from the service requester's 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 (?).

To integrate applications and tools on a B2B grid with shared engineering data, the File Store Service (FSS) (?) 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 FFS 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 paper.

3 ARCHITECTURE

The SORCER infrastructure was chosen as the base for the SILENUS file storage. Many concepts and components are already implemented and tested and do not need to be reinvented. It provides an already existing S2S-framework with well-defined SOO programming and grid computing services.

The SORCER environment has been actively developed in the SORCER lab at the Texas Tech University (?). It is based on Jini network technology (?, ?) and is implemented in Java.

SILENUS (SORCER's Integrated Local Enhanced New User's Storage) consists of several federating services. Each of the services may be replicated on as many hosts as needed on the network.

The conceptual architecture of SILENUS is depicted in Figure 2. All individual services are described below in more detail.

3.1 Byte-Store

The byte-store service is the lowest level service. In hardware terminology it resembles a hard drive. It is responsible for storing files on local computers. Files are identified by an UUID (Universal Unique Identifier), rather than by an actual filename. This way multiple versions of the same file stored can be stored in the same byte-store.

The SILENUS byte-store implementation called Byzantium is based on John McClain's byte-store implementation Holowaa (?). However, the interface for Byzantium is extended to allow for disk space management, persistence, and multi-source download. One important new functionality in Byzantium is the unique and persistent identification of file content by UUID. In the Holowaa byte-store, file-byte-sequences are accessed by ByteSequenceAccessors to make programming simpler The accessor object already contains all the context information and methods needed to access the file content. Unfortunately these accessors do not persist state, thus if the byte-store dies for any reason, the accessor becomes invalid. To avoid this problem, the Byzantium byte-store introduces persistent UUIDs.

3.2 Metadata-Store

The second SILENUS service is the metadata-store. It is equivalent to a file system in traditional storage systems. A metadata-store actually identifies byte-sequences by name and orders them in a tree-like structure. The SILENUS metdata-store implementation is called Midas. Midas stores the metadata information in a database, for fast access and processing. The McKoi database (?) is used by Midas as it provides an efficient embedded database solution, but it could be replaced by any JDBC (?) compatible database.

The metadata-store persists any kind of metainformation about files. It uses generic key-value pairs. The key is the attribute, such as “filename” or “owner”, while the value holds the actual data such as “bla.txt” and “Max”. This allows for classical file information such as filename, owner, and size, but also other attributes: file icon, checksum, size of an image, artist of a recording, full title of a video, etc.. It is up to the user which additional attributes are needed.

Metainformation is stored in a database that makes searches faster than storing it in unstructured formats. That allows querying for file attribute values without parsing the file content, thus retrieving information quicker.

Midas has built-in support for disconnected operations. Metainformation stored in all metadata-stores is identical, as long as they are connected. In the case of unexpected disconnection, the user will be able to use the file system by just being connected to at least one existing metadata-store. Whenever a reconnection is detected, the information in both sets of metadata-stores are synchronized instantly with the followup updates in relevant byte-stores.

While information entered into the Midas database gets synchronized instantly with all other available metadata-stores, a file uploaded to a Byzantium store does not get replicated in all other byte-stores. Both Midas and Byzantium can run on the same host, but that is not mandatory as either Midas or Byzantium only can be started up. Thus, the number of Midas and Byzantium providers does not need to be equal. The number of byte-stores depends on the size of the file space needed and significance of files used (for details see Section 3.5). The number of metadata-stores depends on the number of clients browsing and editing the file system at the same time.

3.3 WebDAV Adapter

To use SILENUS by applications compatible with the local file system only, a WebDAV adapter is provided. This adapter will ensure full compatibility with existing clients and applications. All major operating systems have support for NFS (?), SMB/CIFS, and WebDAV (?). NFS lacks several security features. SMB/CIFS is partially deciphered by the Samba group (?). At this time, the only open and widely supported specification is WebDAV. Thus, instead of rewriting file system drivers for numerous operating systems the approach is to provide a SILENUS adapter based on the WebDAV protocol.

WebDAV is based on the HTTP protocol (?). It is well supported by open-source Slide project (?). As a matter of fact, this allows us to use the standard Java Servlet implementations for the adapter. It can be used within any Java Servlet engine, such as Apache Tomcat (?).

Support for the WebDAV protocol is build into many several operating systems: Windows, Mac OS X, and Linux. In Windows WebDAV is supported under the name "*WebFolders*". WebFolders can be opened and browsed with the build-in support for Windows Explorer. In Mac OS X WebDAV resources can be connected to like any other remote file system. They can be used with Finder and all native applications. Linux support is on its way with the Davfs2 project (?).

There are also several operating system independent clients available for WebDAV. The Nautilus file browser, part of the GNOME system supports WebDAV folders natively. DAVExplorer is a graphical

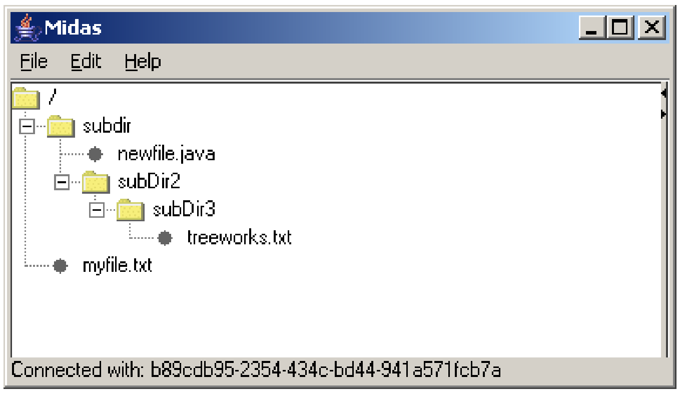

Figure 3: The Midas metadata-store ServiceUI window

Java-based client written at the University of California, Irvine (?). And Cadaver (?) is a simple command line client. All these are available for a wide variety of platforms.

Extension to the WebDAV protocol are defined in later RFCs. Version support is added by (?). Access Control lists are defined in (?). These extensions offer a standard way to access the extended functionality. Unfortunately the existing WebDAV extensions can not provide all of SILENUS functionality, thus complementary user agents have been developed and are described below.

The WebDAV adapter provides access to all applications compatible with the local file system operations using the existing flexible support available on almost any platform. Most widely used platforms even have native support for WebDAV.

3.4 User Agents

SILENUS can be accessed through different types of user agents. The WebDAV adapter described above provides support to existing applications. UI objects can be attached to SORCER service providers as what is called ServiceUI (?). ServiceUIs for Jini services can be accessed within service browsers. SILENUS APIs offer programatic support as well.

SILENUS' ServiceUIs can be displayed through a service browser. A service browser is a common user agent for accessing Jini-based services. A service browser has the ability to download and create the user interface for any service. The service browser is similar to a web browser. It does not require any configuration to find and access services, but is able to download and show ServiceUIs on its own. Currently, service browsers are offered by IncaX (?) and by the JGrid Project (?, ?). The IncaX browser has been used in all SILENUS development and deployment.

Each SILENUS service provider implements a custom ServiceUI. Byzantium ServiceUI provides access to low-level byte-store functions such as deleting file replicas or retrieving raw file content. Midas ServiceUI offers support for browsing, viewing, moving, copying, uploading and downloading files. It has also full support for viewing and editing file attributes. A Midas ServiceUI user agent accessed via IncaX and displaying the Midas root directory is presented in Figure 3.

3.5 Optimizers

Optimizer services are provided to add intelligence to the SILENUS file system. Since Byzantium instances in the SORCER grid can store different content, it is possible that a downloaded file can come from a remote Byzantium instance while a local one is available with no copy of that file. In this case that file is automatically replicated onto the local Byzantium provider. However, with the solution above there is no smart replica management in the grid. Eventually, all files will be replicated on all Byzantium nodes, and all the nodes would run out of disk space. To solve this problem, optimizer services are introduced.

Optimizer services are self-contained services in the SILENUS federation. These services are independent from metadata-store and the byte-store services and even from each other. Multiple instances of each optimizer service can be active or even they can be absent. In the former the file system converges to efficient disk space utilization with SILENUS self-management, in the latter manual configuration and extensive administration is needed.

Special multiplicity attributes describing the significance of files in SILENUS are considered. SILENUS Multiplier (SM) optimizer service keeps the number of copies of a particular file in a range between Min-Multiplicity and Max-Multiplicity. A Watermark-Level number of recommended file replicas is based on a short-term strategy of file usage. Thus, the maximum number of copies can reach Max-Multiplicity but when usage drops then the number of copies will be reduced to the current Watermark-Level. Min-Multiplicity and Max-Multiplicity can be adjusted based on the long-term strategy of file usage.

Most regular files have Min-Multiplicity equal to 2, which means when Byzantium holding that file goes down, there will always be a spare file replica available. However, some files are less significant, or just too large to be replicated so Min-Multiplicity in this case is set to 1. Some files, for example working thesis documents, are very important. They should be always available even if a catastrophe destroys a couple of buildings hosting some SILENUS service providers. These files should have a higher Min-Multiplicity of file replicas, for example in the range from 10 to 100. SM always evaluates multiplicity attributes in all metadata-stores, checks for availability of required number of files in all byte-stores and if more are needed multiplies them or deletes excessive copies.

A second optimizer service is the SILENUS Provisioner (SP). This provider controls the number of Byzantium, Midas, and optimizer services running in the network. SP takes into account the considerations as described at the end of Section 3.2. If SILENUS is running low on available disk-space then SP can provision more byte-stores. If a byte-store is hardly used, then SP reallocates the data from this byte-store to other ones and then shuts down the one with low usage. Additional metadata-stores are provisioned or existing less used are removed by SP based on the number of requests for their metadata. Optimizer services are provisioned in relation to the network size: more metadata-stores and byte-stores are running, more optimizer services are needed.

Other optimizer services that take into account capabilities of different hosts on the network may be considered in the future. A server might be available 24/7, and therefore is appropriate to store critical files. A work computer is predominantly available for half a day so working files can be saved there and used by the person which is usually at that particular computer. A laptop that is constantly disconnected is probably an inappropriate place to store any files, except for the ones needed while working at that computer.

These are only a few examples of what optimizers can and should do. Since they are independent from the other service providers, new optimizers can be added to improve the overall SILENUS performance. It is totally up to the administrator of the file system to decide which particular optimizers to deploy, otherwise SP will maintain the right number of Midas, Byzantium, and complementary optimizer services in the federation autonomically.

4 CONCLUSIONS

A solution for storage management in file-based data grids is proposed. Splitting up the file system in federating services provides for high scalability. Replication and autonomic provisioning of services improves reliability and solves the problem of network and computer outages. The SILENUS federated framework allows for complementary services to satisfy future needs.

Although the basic design has been completed, there are still some open issues, and some of the ideas presented in this paper have not actually been implemented yet.

Two versions of Byzantium have been implemented. One is based on the Holowaa code that uses direct TCP connections for data transfer. Another one uses Java RMI for data transfer. Both implementations perform well in initial tests, however detailed performance analysis is needed to make appropriate recommendations.

Midas and its ServiceUI have been developed as well. The current implementation is based on work done by Vivek Khurana (**?**). The previous version was successfully tested with large datafiles for the Basic Local Alignment Search Tool (BLAST).

Other SILENUS services, in particular optimizers, are in the design phase. Selection of relevant heuristics for optimizers is under ongoing investigation. For the WebDAV adapter the support in different operating systems is under detailed investigation as well.