

A Federated Grid Environment with Replication Services

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

Grids can be classified as computational grids, access grids and data grids. Computational grids address applications that deal with complex and time intensive computational problems, usually on relatively small datasets. Access grids focus on group-to-group communication. Whereas data grids address the needs of applications that deal with the evaluation and mining of large amounts of data in the terabyte and petabyte range. While SORCER is a federated computation grid environment, a complementing data grid service called Replica Provider is introduced. The newly developed data grid service is used and at the same time as the already existing SORCER compute grid is leveraged for an increased functionality. SORCER service-oriented programs with replication services now has the capability of running data grid applications. It enables better access to such databases and increases reliability by replicating them at multiple locations. A federated grid environment with replication service is presented. It enables large number of files to be replicated on multiple nodes over different heterogeneous computation platforms and provides generic service providers for fast, up-to-date, and reliable access to file storage.

1 INTRODUCTION

Grid concepts are particularly relevant to concurrent bioengineering and bioinformatics applications due to the collaborative nature of such experiments and the increasing complexity of data analysis tasks, and hence illuminate a need for next-generation experiments to exploit large distributed collections of shared resources. Hence the trend towards the creation of so-called bioinformatics Data Grids, i.e., grid infrastructures, tools, applications and service designed to enable concurrent and distributed access to, and analysis of, large amounts of biological data. The broad significance of grid concepts, in advanced scientific collaborations and in business, means that bioengineering and bioinformatics is just one of many research domains that are contributing to and/or driving grid technologies. The resulting interrelationships make it important to understand the state of the art and likely future directions in this field.

One example of these bioinformatics applications is the Basic Local Alignment Search Tool (BLAST) (?). It is used to find string sequences in large DNA or protein databases. The size of each of these sequence databases is very large and growing exponentially. A data grid can help us in easily and efficiently maintaining and updating such large databases in a distributed computing environment. It also allows for faster access to such databases and to increase reli-

bility by replicating these databases at multiple locations.

The SORCER environment (?, b, ?, ?, ?) currently provides a File Store Service (FSS) (?). It supports filtering out information from remote files, thus reducing the amount of data transfers between providers. Even though the File Store Service (FSS) can be replicated like any other service in SORCER, there is a scope of further improvement to make it more scalable and reliable. Firstly, the FSS does not replicate files among federating service. Therefore, if one of the file store services crashes, all its files are not available to participating services. Thus, a replication service that replicates files on multiple nodes in the SORCER network is needed. Replication services enhance the reliability of the file store service and collaborating compute grid services. Hence, each of the dedicated nodes has a copy of the file, and file access time is reduced considerably.

2 PRINCIPLES OF FILE REPLICATION

The ultimate goal of file replication in a data grid is to make a submitted file to a grid immediately available on all dedicated nodes in exactly the same manner it is available on the node it was introduced in. A file should be available in a reliable manner, even if a single data node is off the network.

To achieve this goal data grids use file replication. If a file is introduced into the data grid, it will be immediately replicated among dedicated nodes. Thus, if one of the nodes is disconnected from the network, the file is still available from another replicated node. Usually data is managed in two layers: the actual file content, and metainformation about the content. The metainformation layer for example contains information about which file can be found where and in which version.

The Globus Alliance (?) provides several solutions for managing files in data grids. The Globus toolkit (?) provides a solution for high-performance data grids, based on GridFTP and Replica Management Service that are introduced in a Grid Data Management Pilot (GDMP). (?). Also, it defines a meta-catalog for describing files in replicated data grids. (?). However, none of these solutions fit exactly the requirements of service-to-service (S2S) federated computing.

GridFTP is a data transfer and access protocol that provide, efficient data transfer in Grid environments. The GridFTP protocol extends the standard FTP protocol, providing a superset of the features offered by the various Grid storage systems currently in use. The GridFTP protocol was developed to satisfy the need of Grid environment for fast, efficient and reliable transport mechanism. GridFTP does not support secure transport of data, it is concerned only with secure login.

A Replica Catalog service was developed with the goal to support replicated data wherein the details of the physical host are removed from the file descriptor. This Replica Catalog service also supports striped data transfer and can provide metadata. The Globus implementation of this service uses LDAP as the database; however any database could be used in general.

Replica Management Service (RMS) integrates GridFTP and Replica Catalog into one system that is transparent to the user to provide replica management capabilities for data grids. The library provides client functions that allow files to be registered with the replica management service, published to replica locations, and moved among multiple locations. The library uses the Globus Replica Catalog and GridFTP technologies to accomplish this work. However, RMS still needs improvement for fault recovery, since the Replica Catalog is a central server with a single point of failure.

Certain shortcomings exist such as, until now, it does not support automatic replication of data. Also, the metadata is stored in one central database, the location for which has to be known.

Therefore, the Replica Catalog service is a right step towards distributed storage and is appropriate for

replication of large amount of data for read-only access. However, write access is not supported.

3 FEDERATED FILE REPLICA MANAGEMENT AND STORAGE

The goal of the grid environment with replication services is to develop a data grid services that manage large number of files replicated on multiple nodes over different heterogeneous computational platforms to allow users and federating service providers for fast, up-to-date, reliable and secure access to distributed file storage. The presented environment has been validated by developing replication providers (service replicas) for service-oriented BLAST.

Initially the service-oriented BLAST has been developed in the SORCER environment (S-BLAST) with no replication service. Below the basic concepts of federated grid computing are described along with explanations how S-BLAST works.

Building on the Object-Oriented paradigm is the Service-Oriented paradigm, in which the objects are distributed, they are network objects that play some predefined roles. A service provider is an object that accepts messages from service requestors to execute an item of work – a task. The task object is a service request – a kind of elementary grid operation executed by a service provider. A service jobber is a specialized service provider (broker) that executes a job – a compound service request in terms of tasks and other jobs. The job object is a service-oriented program that is managed by a jobber and is dynamically bound to all relevant and currently available service providers on the service grid.

The collection of grid providers dynamically (in runtime) identified by a jobber during a job execution is called a job federation. This runtime network or grid federation is the jobs execution environment and the job object is a service-oriented program. In other words, the object-oriented concepts are applied directly to the grid in the service-oriented paradigm.

A task and job is treated as a grid activity called an exertion. An exertion is defined by its context model (data), and by its service method (a pair: interface and selector). A context model is a tree like structure of data being processed. Each path of a tree names a leave node where the data resides. A service method defines a service provider (grid object) to be bound to in runtime. This network object provides data defined by the context or the business logic to be applied to the context. The method is primarily defined by a provider type (interface) and selector (method name) in the provider's interface. The service method may also refer to a piece of code to be downloaded and executed by a provider (mobile code). The information included in the service method allows the SORCER program to bind the task to the network ob-

ject and process the context by the operation, which is defined by the method selector. This type of service provider is called a method provider. Another type of service provider is a data (context) provider that provides shared data to grid method providers. Thus, both context and method providers represent grid data and operations respectively to be used in the grid-oriented programs.

A job in SOCER is usually created interactively as a service-oriented program (?). Alternatively a job creation can be delegated to a job provider (in S-BLAST by BlastProvider) that supplies a downloadable user agent code. Based on the user's input an S-BLAST job (jobs) is created by BlastProvider and is passed on to a jobber. Each instruction in that program (job) is represented by the component task in the job created by BlastProvider according to the user input.

The computing S-BLAST framework has been developed with no replication services initially. A large BLAST job is divided into several independent SORCER tasks that are distributed dynamically and executed in parallel by generic SORCER service providers - taskers. A collection of all service providers (active and inactive) is called a SORCER grid. A job execution federates providers that come together for completing an S-BLAST job, with multiple SORCER taskers as depicted in Figure 1 by bold outlines.

The SORCER architecture allows services to share data by using an object shared repository - a space provider (spacer) implemented with JavaSpaces (?). A spacer allows for asynchronous execution of tasks such that a task can wait for a service to be available, in the case of S-BLAST, a tasker. Taskers execute a downloaded code as it is specified by a service method. In S-BLAST that service method communicates with a BLAST database and via a system call executes a BLAST program (C/C++ executable). A jobber drops tasks of the job being coordinated into the shared object space and taskers at own pace pick them up from the spacer and after executing them returns them back to the spacer. The jobber collects all executed task from the spacer and combines them into a resulting job returned to the requestor.

Please note that federating service providers do not have mutual associations prior to the job execution. They come together (federate) for a specific S-BLAST job. Each provider in the federation performs its services according to a jobber coordination strategy defined by the job itself. Once the job is complete the federation dissolves and the providers disperse and seek other jobs to join. The same provider can provide multiple services in the same federation and different providers can provide the same service in different federations. The grid is dynamic in which new services can enter the network and existing

services can leave the network at any instance. The service-based architecture is resilient, self-healing, and self-managing. The key to the resilience is the transparency of service discovery and seamless substitution of one service with another. SORCER defines all decentralized distributed components in the system to be equal by public common interfaces. Each peer may implement multiple specific (other than SORCER common interfaces) interfaces that are published when the peer joins the grid. Both arguments and return values of these specific methods are instances of type ServiceContext that represent service data. By its ispecific interface (type) and optional attributes (e.g., provider name), the network object can be dynamically found on the network without a host name and port required. These specific interfaces and their implementations might change, as they are relevant to particular service providers.

In the presented S-BLAST approach taskers can use a single BLAST database or that database can be local on each tasker node. In the former, having a single database creates the network bottleneck for all taskers accessing the same database and a single point failure at the same time. In the latter, managing multiple frequently evolving databases on multiple hosts is a system management nightmare. The solution is to deploy one database that can be replicated by dedicated SORCER replication providers (replicas). Thus, a tasker can dynamically find available replicas and use one of them as a part of the extended federation for the S-BLAST job being coordinated by a jobber. When any BLAST database is updated, all replicas will synchronize the data accordingly so the tasker will always have access to the most current information.

4 ARCHITECTURE AND DESIGN

The following communication diagrams capture use cases and explain the functionality of different components defined for replication services in SORCER.

4.1 Overview

Basic components and interactions of the replication federation are illustrated in Figure 2. A replica provider is a pair of service providers: a metadata-store (RPMS) and a byte-store (RPBS). In Figure 2 an instance of replica provider is indicated by a dashed ellipse. Coordination and data synchronization between RPMS providers is facilitated by a spacer.

RPMS is responsible for storing and updating information about files to be the same in all RPMS instances. RPMS communicates with the RPBS provider for initiating synchronization and transfer of files among different RPBS providers. RPMS uses an embedded Mckoi database (?) to persist file metadata.

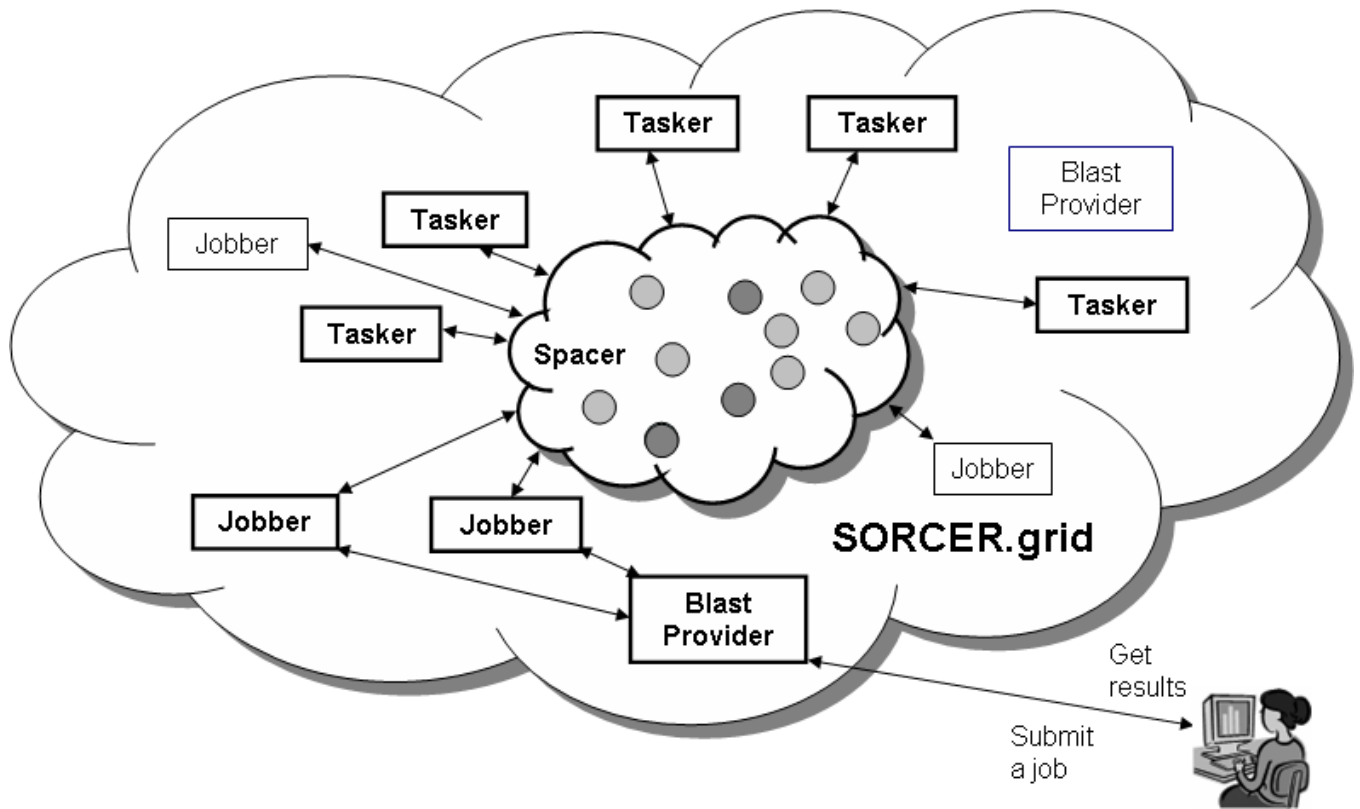


Figure 1: S-BLAST components and data flow without replication services. The federation components are marked by bold outlines.

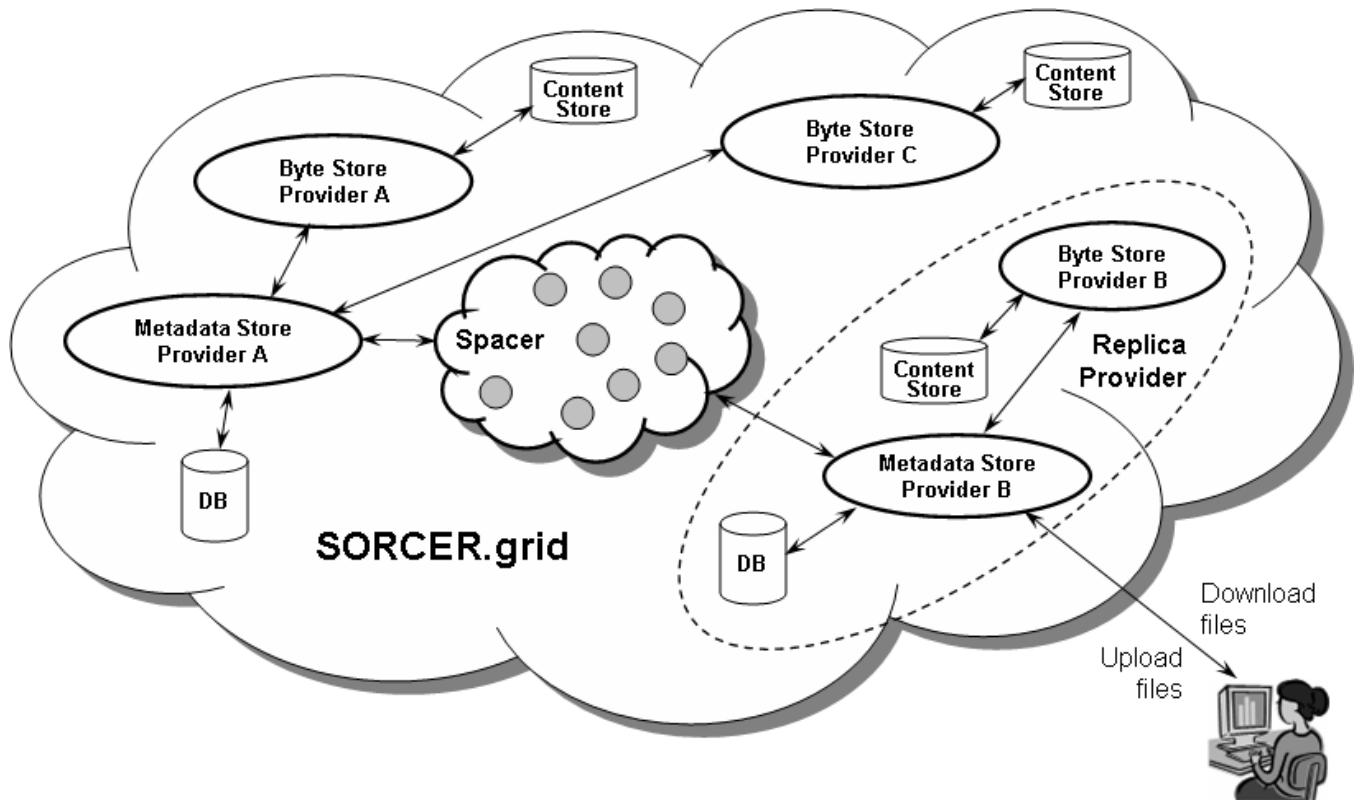


Figure 2: Overview of basic components and the data flow in the replication federation. The federation components are marked by bold outlines.

The RPBS provider is specialized in efficient transfer of files. It uses a smart proxy to transfer files over a direct TCP connection. It stores files in the underlying file system – content store – for future access, while metainformation is stored in a RPMS' embedded database.

An RPMS user agent, available via a service browser, allows the user to upload and download files. Use cases describing these operations are explained below. The RPMS user agent is implemented as a Jini ServiceUI (?).

Jini ServiceUIs are displayed using a service browser. A service browser is a common user interface for all Jini-based Services. IncaX (?) offers a free service browser on their website for download. A service browser has the ability to download and invoke a user interface for any service without prior configuration.

4.2 The Browsing Files Use Case

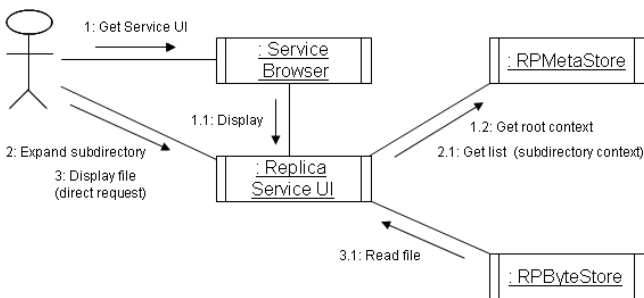


Figure 3: The Browsing Files Use Case

The Browsing File Use Case shown in Figure 3) defines the first step that the user performs for using the replication service. The user interacts with the RPMS service through its ServiceUI from a service browser. The user can interactively select and submit files to be replicated.

To display the directory structure, the RPMS ServiceUI gets the context for the root node, containing information about the root directory and its child elements. With this information it can start listing subdirectory contents. This information is then displayed to the user. The user is now able to browse, by selecting directories, files and read their content.

4.3 The Writing File Use Case

The Write File Use Case (Figure 4) is that of storing a local file in the replication service. The user can select a local file in the RPMS ServiceUI and upload it. Two things happen:

The content of the file is sent to a byte-store provider. This provider writes the contents of the file to a local file system. Then a location information exertion is written into the exertion space, which contains the information where the file can be found. This is used to synchronize it with other replica byte stores.

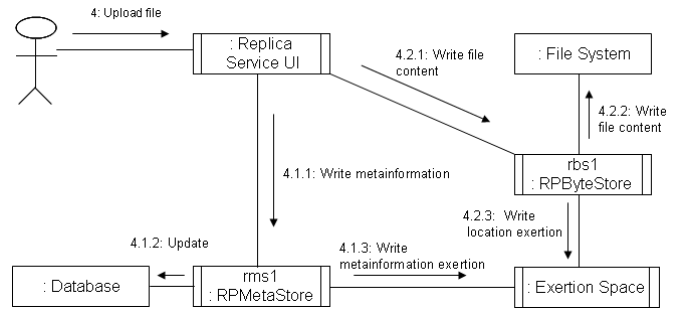


Figure 4: The Writing Files Use Case

The metainformation is passed to the replica metadata-store. It stores this data in its database. Then it writes a metainformation exertion into the exertion space, which has the same information about the file. This information will be read by other metadata-store providers and inserted into their database.

4.4 The Scheduling Update Use Case

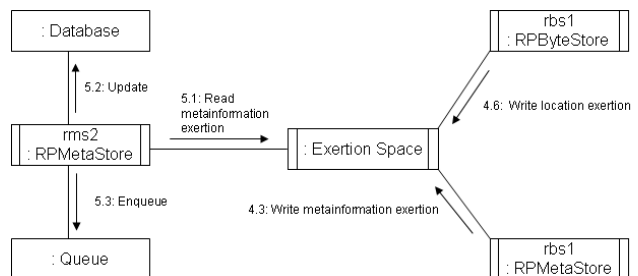


Figure 5: The Scheduling Update Use Case

The Scheduling Update Use (Figure 5) explains the scheduling update functionality of the replica provider. The current replica metadata-store writes the metaexertion into the exertion space. On all replica providers, there is a listener which is notified and reads this metaexertion from the space. The metaexertion is kept in the exertion space, so that all providers can read it. The listener passes the information to its replica provider which updates its database tables to reflect the new metainformation changes. The metadata-store then en-queues this information into a FIFO queue data structure for later processing.

4.5 The Processing Update Use Case

The Processing Update Use Case (Figure 6) is used to have every file available on all hosts at the same time. The downloading process retrieves the first item in the queue, and starts processing the updates by taking the location exertion from the exertion space, if it is available. Taking an exertion will automatically block all other downloading processes from taking the same exertion at the same time. Once the location exertion is received, it reads the file from one byte-store and writes it to its own byte-store. Once this is done, it can now write two exertion locations back into the

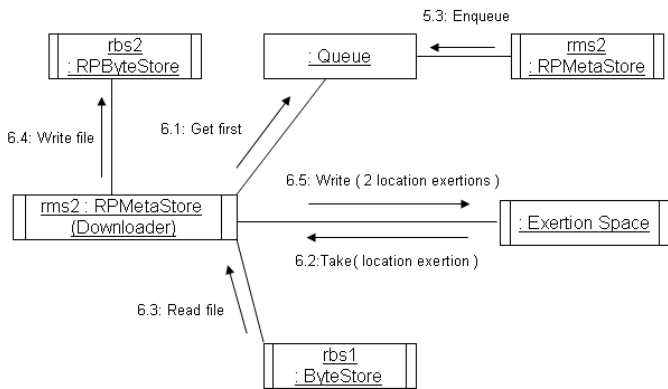


Figure 6: The Processing Update Use Case

exertion space, since the file is now present on two nodes.

4.6 The Reading File Use Case

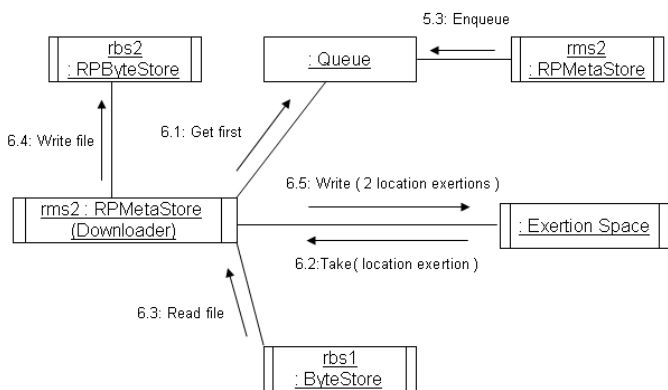


Figure 7: The Reading File Use Case

The the Reading File Use Case two distinct cases have to be looked at. If the file is already available at the local byte store, the ServiceUI is connected to the local byte-store, and the user can download the file from there.

When a read file request is made for a file which is not available in the local byte store (Figure 7), the current metadata-store writes a request exertion to the exertion space, which contains meta-information about the missing file. The metadata-store that has a copy of the requested file takes this request exertion from the exertion space and writes a location exertion specifically for the requesting metadata-store A. The requesting metadata-store takes the location exertion from the exertion space which contains information where the file can be found. It can now copy the file from the byte store provided in the location exertion to its local byte store, and then make it available to the user.

5 IMPLEMENTATION OF FEDERATED REPLICATION SERVICES IN S-BLAST

The system was implemented in the S-BLAST service. The S-BLAST services are distributed across

multiple nodes. Each of these services requires a large database file for the actual processing.

In earlier versions of S-BLAST this database file had to be copied manually to every machine. With Replication services this changed. The user does not even have to know that there is a replication service in the background. Figure 8 shows the user interface. Now all a user has to do is select a blast database file to be used. As soon as this file is uploaded into the replication services, it is already replicated among all nodes, before the user has even finished selecting the other BLAST settings. This gives this system a big advantage over other systems, because during this time the network would normally be idle. The logarithmic file replication proved to be very scalable. No matter if four hosts were used (in the lab) or twenty, the system did not overload. The replicated S-BLAST was successfully utilized, and provided results, which were collected and displayed to the user.

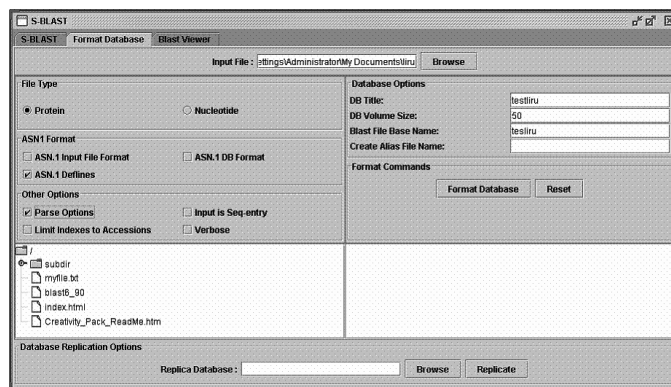


Figure 8: User interface for Service Oriented Basic Local Alignment Search Tool (S-BLAST)

6 CONCLUSIONS

This particular services proved to be very fast and reliable. Data was available immediately on all clients, but the network was not overloaded.

Operation testing proved to be successful for the replica provider as all the functionalities which included uploading, reading a file, making changes and writing a file were found to be working according to the use cases defined. Maintenance and portability was tested by starting the replication services on different platforms wherein before the service is made available for use, it is synchronized automatically by the update functionality. Reliability tests proved to be successful as no files in the replica byte store were lost even when some services were restarted while others were running.

Other grid data management systems, like Globus provide only a pull-functionality. Data has to be requested before it is actually transferred to the host of choice. With this replication service, however, the

knowledge that the data will be needed can be used to optimize data transfer and to provide fast access.

Grid approach to storage is designed to let users focus more on how they use data, not how they store it. Applying the grid concept to a computation network lets us harness available but unused resources by dynamically allocating and de-allocating capacity, bandwidth and processing among numerous distributed computers. A computing grid can span locations, organizations, machine architectures and software boundaries, offering power, collaboration and information access to connected users.