

# Implementation and Evaluation of a ReplicaSet Grid Service

Mary Manohar, Ann Chervenak, Ben Clifford, Carl Kesselman  
*University of Southern California Information Sciences Institute*  
{*mmanohar, annc, benc, carl*}@isi.edu

## Abstract

*We present the implementation and performance of a ReplicaSet service based on the specification developed in the OGSA Data Replication Services (OREP) Working Group of the Global Grid Forum. This standard is based on the Open Grid Services Infrastructure. The ReplicaSet service aggregates information about replicated data items and enforces policies for authorization, replica semantics and consistency.*

## 1. Introduction

Scientific applications such as high-energy physics, climate modeling and computational genomics generate large volumes of data, measured in terabytes and petabytes. These large data collections are often stored in geographically distributed locations, and the communities using these data collections may also be large and geographically distributed. Performing computationally intensive analysis on these data collections requires efficient management and transfer of large volumes of data in wide-area environments. Often, this data management includes replication of data items to provide better performance via load balancing and higher availability by avoiding single points of failure.

The OGSA Data Replication Services (OREP) Working Group of the Global Grid Forum has been working to develop standards for ReplicaSets that provide a mechanism to associate and discover replicas of a data object. We have implemented a prototype ReplicaSet Grid service based on the OREP specification [1]. The ReplicaSet service design is compliant with the Open Grid Services Infrastructure specification for Grid services [2]. A ReplicaSet provides a mapping from its handle to handles for its member data services, which are replicas of each other. Thus, each ReplicaSet service performs a similar function to an entry in a Replica Location Service [3], which provides mappings from logical data identifiers to replica locations.

In this paper, we describe our ReplicaSet implementation and present performance results. In particular, we describe a variety of replica

semantics and membership enforcement policies that can be supported by ReplicaSet services, and we discuss the particular policies that our prototype supports, which are based on verifying the checksum of a file being added to a ReplicaSet.

Although the ReplicaSet service is designed to be a component in an overall replica management system, its applicability is not limited to association of replicas. The extensions we define to OGSI ServiceGroups [2] can be generalized to provide mechanisms for creating associations among Grid services based on policies for authorization and membership maintenance.

## 2. Background: Grid Services

The ReplicaSet service design complies with the Open Grid Services Infrastructure specification [2]. The OGSI specification is based on a uniform service model that characterizes everything in a Grid environment (compute resources, data items, networks, programs, etc.) as *services*. Grid services are Web services that provide well-defined interfaces, including standard mechanisms for lifetime management; the definition of *Service Data Elements* for associating state with a Grid service; introspection mechanisms for discovering service state; subscription and notification mechanisms that allow a client to subscribe to the state associated with a service and be notified when that state changes; the dynamic creation of service instances via the use of *Factory* services; and globally unique addressing for Grid services using Grid Service Handles [2].

The ReplicaSet design depends on two other aspects of Grid services: OGSA Data Services and OGSI ServiceGroups.

In a Grid services environment, data objects are OGSI-compliant *data services*. The OGSA Data Service specification [4] describes a data service as a Grid service that represents and encapsulates a data virtualization, which is an abstract view of some data. A data service has service data elements (SDEs) that describe key parameters of the data virtualization and operations that allow clients to inspect those SDEs, access the data, derive new data virtualizations, and manage data virtualizations. Data services inherit basic lifetime

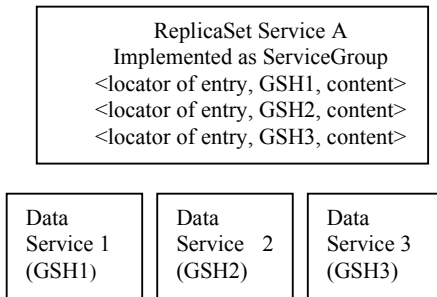
management capabilities from OGSi Grid Services and may be created dynamically using data factories.

In the Open Grid Services Infrastructure (OGSI) Version 1.0 specification, ServiceGroups are defined as Grid services that maintain information about a group of other Grid services [2]. A ServiceGroup contains entries for member Grid services. These entries include a locator such as a Grid Service Handle that refers to the member of the service group. Operations associated with the ServiceGroup include add and remove operations that are used to register and unregister entries.

### 3. The ReplicaSet Design

The ReplicaSet service that is being standardized through the OREP Working Group represents a set of replicated data objects as a Grid service. Important aspects of the ReplicaSet service design [1] include:

- Replicated data items are OGSA Data Services that are uniquely identified by Grid Service Handles (GSHs).
- Data services form a replica set according to some semantic definition of replication.
- A replica set should be exposed as a Grid service called a ReplicaSet service.
- A ReplicaSet service design should be based on and extend OGSI ServiceGroups, which are collections of Grid services.
- The ReplicaSet may have associated policies for authorization and semantics.
- The ReplicaSet service design may include additional indexes for aggregating information about ReplicaSet ServiceGroups. These indexes should be designed as extensions of ServiceGroups.



**Figure 1: ReplicaSet Grid service and member data services.**

Figure 1 shows the ReplicaSet service implemented as an OGSI service group, where the members of the service group are OGSI-compliant data services. Each entry in the ReplicaSet service associates the Grid Service Handle of a member

data service, a handle for the ReplicaSet entry and some optional additional content.

A scenario for creating a new ReplicaSet and adding a member to the service would include the following steps. First, a client would request creation of a new ReplicaSet instance from a ReplicaSetFactory. If the client is authorized to create a new ReplicaSet instance, one will be created and the handle of the new service will be returned to the client. Next, the client will issue a request to add a data service as a member of the new ReplicaSet. The ReplicaSet will enforce authorization, replica semantics and other policies to determine whether the client is allowed to add a new member of the equivalence set.

### 3.1 ReplicaSet Policies

A key aspect of our ReplicaSet design is the ability to associate policies for authorization, replica semantics and semantic enforcement with a particular ReplicaSet service instance. The ReplicaSet provides a convenient point for policy enforcement.

ReplicaSet authorization policies specify who is allowed to create ReplicaSet instances or to add or remove ReplicaSet members. When a client attempts to perform one of these operations, the client's access rights will be checked, and any unauthorized operations will be disallowed by the ReplicaSetFactory or the ReplicaSet service.

Semantic policies associated with a ReplicaSet define what constitutes a valid member of a replica set. Examples of replica semantics include exact byte-for-byte copies, versions of data items, partial replicas, replicas synchronized within a specified interval, or data items that contain the same content in different formats.

There may also be a range of policies regarding when replica semantic policies are enforced. At one extreme, a ReplicaSet could assume it is operating in a trusted environment and perform no verification that particular replica semantics are satisfied. At the other extreme, a ReplicaSet may continuously monitor members of a replica set to maintain consistency among members and remove non-complying replicas. One mechanism for continuous enforcement would be for the replicaSet service to subscribe to its member data services and receive notifications when updates occur to those member data services. Another enforcement scheme might verify that a new member joining a ReplicaSet service meets the semantic policy for replication at the time of addition, but may not monitor compliance thereafter.

### 4. The ReplicaSet Implementation

Next, we describe our prototype implementation of a ReplicaSet service for use in the Globus

Toolkit version 3.2 Grid services environment. Because various aspects of Grid services and data services are still under development, in some cases our implementation simplifies functionality, as described below.

#### **4.1 Identifying Replicated Data Items**

The Data Services specified by the OGSA Data Services Specification have not yet been implemented in the GT3 environment. For this reason, our implementation uses file URLs as locators for data objects.

#### **4.2 Policy Enforcement**

As already discussed, a range of replica semantics and enforcement policies can be supported in a ReplicaSet. Our prototype implementation currently supports two sets of policies for enforcing replica semantics. In our first set of policies, we assume a high level of trust between clients and ReplicaSet services and perform no verification at the time a member is added to a ReplicaSet. In other words, as long as a client is allowed by the authorization policies of the service (described below) to add a member to the ReplicaSet ServiceGroup, we perform no additional checks to verify that the member being added is actually a replica of existing members of the ReplicaSet by some semantic definition of replication.

We implement a second set of policies for enforcing replica semantics that requires verification that a member being added to a ReplicaSet service has exactly the same checksum as the first member added to a ReplicaSet service. This enforcement is only performed at the time a replica is added to the ReplicaSet. If replicas are later updated, the resulting inconsistency among replicas will not be detected by our ReplicaSet service.

One of the consequences of our policy enforcement is that adding a member to a ReplicaSet requires us to calculate the checksum for the data file. This requires transferring the file to local storage, performing the checksum calculation, determining whether the checksum matches the one required by the ReplicaSet service, and deleting the temporary copy of the file. The overhead of performing this verification is proportional to the size of the file and can be substantial for large files, as shown in our performance results.

An alternative policy enforcement implementation would avoid the overhead of copying files and calculating checksums by allowing a client that wants to add a file to a ReplicaSet to assert a checksum value for the file. In this case, the ReplicaSet service would only need to verify that the asserted checksum matches

the one required by the replicaSet. To accept such a signed checksum, our ReplicaSet service would need to have a sufficient trust relationship with the client to believe that the checksum assertion is valid. We plan to implement an assertion-based ReplicaSet service in the future.

#### **4.3 Factory Authorization Policy**

To create a new instance of a ReplicaSet service, a client makes a request to a ReplicaSetFactory, which is an extension of an OGSF Factory service. The ReplicaSet factory determines whether a client is allowed to create a new ReplicaSet instance. It uses a standard Globus grid-mapfile that specifies the users who are allowed to create instances of the ReplicaSet service.

#### **4.4 ReplicaSet Authorization Policies**

Authorization restrictions are also enforced by each ReplicaSet service to determine whether a particular client is allowed to add or remove entries in the ReplicaSet service. These restrictions are based on a Globus grid-mapfile, which specifies the users allowed to add or remove members of a ReplicaSet. One limitation of the current implementation is that we use a single grid-mapfile for all ReplicaSet instances. It may be preferable to have a different grid-mapfile associated with each ReplicaSet instance or with a group of instances.

#### **4.5 Extending ServiceGroup Operations**

The ReplicaSet service implementation modifies the add and remove operations provided by OGSF ServiceGroups [2].

When a new replica is added to the ReplicaSet service, our implementation first copies the data file from the remote location to the local host using the RFT (Reliable File Transfer) Service or the GridFTP data transport protocol. We verify that the checksum for the new replica instance matches the checksum required by the ReplicaSet, and if so, we add the file's URL to the ReplicaSet and delete the local copy of the file. The first replica added to a ReplicaSet is the "master copy" whose checksum must match all subsequent replicas.

The remove operation removes an entry from the ReplicaSet. If a remove operation unregisters the last member of the ReplicaSet, then the checksum associated with the old master copy is deleted. A subsequent add operation to the ReplicaSet creates a new master copy and resets the checksum for the ReplicaSet service.

#### **4.6 File Transfer for Add Operations**

We use either the Reliable File Transfer (RFT) Service [5] or the GridFTP data transport protocol [6] to copy a file to local memory before verifying

the checksum and adding the file as a member of the ReplicaSet service. The RFT service transfers byte streams reliably. RFT is built on the top of GridFTP data transport client libraries. RFT maintains persistent state about outstanding transfers and restarts partially completed transfers after failures of the source or destination of the transfer.

Our implementation defines three configurable parameters for file transfer: the transfer mechanism to be used (RFT or GridFTP), the local directory where a file can be copied for the checksum calculation and a maximum file size that limits the amount of data copied to the local machine.

#### 4.7 Checksum Calculation

After the file is transferred, we verify that its checksum matches that of the master copy for the ReplicaSet service. We calculate an MD5 checksum for the file using the implementation provided by the Java 2 Runtime Environment version 1.4.0.

#### 4.8 Service Data Elements

We defined two Service Data Elements (SDEs) for introspection by clients of the ReplicaSet service: *numOfReplicas* and *Checksum*. NumOfReplicas indicates the number of members of the ReplicaSet. The checksum SDE provides the checksum for the master copy of a ReplicaSet service, allowing users to verify the checksum of their own replica before attempting to add the replica as a member of the ReplicaSet service.

#### 4.9 Service activation/deactivation

To minimize memory usage, it is possible to deactivate services when they are not actively used. Our ReplicaSet implementation allows users to enable or disable service activation/deactivation.

Our ReplicaSet implementation ensures that when a service is deactivated, all necessary service state is written to disk via a MySQL database. The database has two tables. The ReplicaSet table contains the name of the service and its related state or Service Data Elements, such as checksum. The Mapping table contains mappings from the ReplicaSet service handle to one or more data service handles for members of the ReplicaSet.

A deactivated service is re-activated when a call is made to the service. Upon activation, the service implementation reads relevant service data from disk and initializes the service.

Service activation/deactivation provides the ability to make services persistent across container lifetimes, since service state can be restored from disk even if a container fails and a new one is created. It also ensures scalability by allowing the container to host more services, since deactivated services consume less memory in the container.

## 5. Performance and Scalability of the ReplicaSet Service Prototype

In this section, we present initial performance and scalability results for our ReplicaSet service implementation. The ReplicaSet service was implemented for Globus Toolkit version 3.2. Two workstations on a local area network were used in running these measurements. The first workstation (A) is a dual processor 2.2 GHz Intel Pentium machine with hyper-threading enabled running Red Hat Linux version 9.0. This workstation runs the hosting environment in which the ReplicaSet service is deployed as well as our client program that attempts to add replicas to a ReplicaSet service. This workstation is also the destination of data transfer operations and the machine that calculates MD5 checksums to verify that a replica may be added to the ReplicaSet service. The second workstation (B) is a single processor 2.00 GHz Intel Pentium machine that runs Red Hat Linux version 7.3 and contains the files that are added as members of the ReplicaSet.

First, we measured the capacity of the container to identify the maximum number of services it can host at a time. We also performed tests to determine performance for creating, adding and querying ReplicaSet instances. With service activation/deactivation disabled, we performed the tests on containers having 20 and 1000 ReplicaSet instances. With service activation/deactivation enabled, we performed the tests on containers having 1000, 5000 and 10000 ReplicaSet instances. In each of the cases, the instances had 4 replicas each. All the tests were run on machine A. Machine B was used only to perform data transfers. We used a deactivation interval of 5 seconds, meaning that the services get deactivated if not used for 5 seconds.

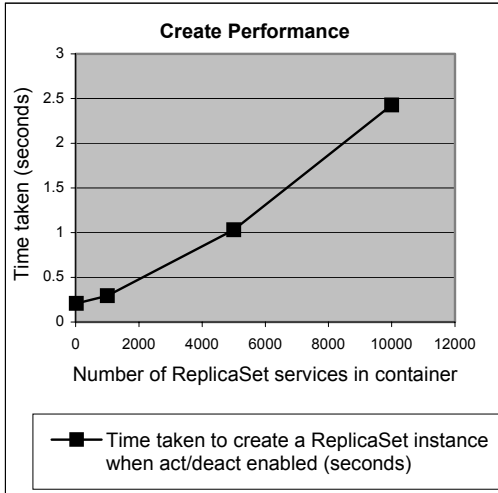
### 5.1 Container capacity

We measured the capacity of the Grid services container to identify the maximum number of services it can host, both with service activation enabled and disabled. With a heap size of 128 megabytes for the Java Virtual Machine (JVM) running the container, we were able to create 1388 ReplicaSet service instances, each having 4 replicas, when service activation/deactivation is disabled, and more than 10000 instances having 4 replicas each when service activation/deactivation is enabled. As the heap size increased, we were able to create more services.

### 5.2 Service creation

Service Creation involves creating a new ReplicaSet service instance by invoking the factory. With service activation/deactivation disabled, it took 0.172 seconds and 0.1775 seconds

to create a new instance when the container had 20 instances and 1000 instances respectively. Thus, the number of existing services has no impact on the creation time in this case.



**Figure 2: Time taken to create a new ReplicaSet service instance**

By contrast, when service activation/deactivation is enabled, the time required to create a service instance is affected by the number of active services, as shown in Figure 2 for a container with 20, 1000, 5000 and 10000 instances. In this case, the implementation writes the service information into the database during service creation. This is to ensure data consistency upon system failures. We note that the create performance when relatively few services are active (20 and 1000 in the graph) is similar to the

performance when activation/deactivation is disabled. However, when a large number of services are deployed in a hosting environment in which activation/deactivation is enabled, create performance is reduced.

### 5.3 Adding Replicas to ReplicaSet Service Implementations

As described above, one variation of our ReplicaSet service implementation performs data transfers using the GridFTP data transport protocol to transfer files. In Table 2, we present performance results for this version of the implementation. We measured the time required to add a replica to a ReplicaSet service for files ranging in size from 100 megabytes to 10 gigabytes. Each data point in the table shows a mean of five ReplicaSet addition operations; these numbers exhibited low variance.

The second column in the table shows total observed time at the client for a replica addition to a ReplicaSet service. The other columns show the breakdown of this total observed time, including the time for the GridFTP copy operation, the checksum calculation and other overhead. The GridFTP data transfer time is proportional to the size of the file. The checksum calculation is roughly proportional to file size, with larger files requiring longer calculation times.

**Table 2: Performance when Adding a Replica to a ReplicaSet Service Using GridFTP for File Transfer**

<b>File Size</b>	<b>Total Replica Addition Time at Client (seconds)</b>	<b>GridFTP Copy (seconds)</b>	<b>Checksum Calculation Time (seconds)</b>	<b>Additional Overhead (seconds)</b>
100 MB	22.103	13.271	3.75	5.082
500 MB	75.35	50.72	20.45	5.29
1 GB	146.014	100.83	40.05	5.134
2 GB	273.28	188.5	78.89	5.89
4 GB	539.74	376.37	158.01	5.36

**Table 3: Performance Using Reliable File Transfer Service for File Transfer**

<b>File Size</b>	<b>Total Replica Addition Time at Client (seconds)</b>	<b>RFT Copy (seconds)</b>	<b>Checksum Calculation Time (seconds)</b>	<b>Additional Overhead (seconds)</b>
100 MB	25.324	17.527	2.761	5.036
500 MB	78.807	54.135	19.533	5.139
1 GB	151.179	107.314	39.133	4.732
2 GB	287.812	204.373	78.099	5.34

The rightmost column shows other overheads, including Grid service overheads and the time required to add a new member to the ReplicaSet ServiceGroup. These overheads of 5 to 6 seconds are fairly constant and do not depend on file size.

Table 3 shows performance for our ReplicaSet implementation when using the Reliable File Transfer service to perform the data transfer before the checksum calculation. We are limited by the current RFT implementation to files of size 2 gigabytes or less. In each case, the time to add a replica in the ReplicaSet implementation that uses RFT is somewhat longer than the time for the corresponding file size using GridFTP. While checksum calculation times and other overheads are about the same for both implementations, the data transfer takes approximately 8% longer using RFT compared to transfers using GridFTP. This extra time is due to the time required to invoke the RFT grid service to perform the data transfer as well as the periodic checkpoint operations performed by RFT to save the state of the transfer to a database. This checkpointing provides the reliability of RFT, allowing the service to resume transfer operations after failures of source or destination machines or after the failure of the RFT service itself.

A final variation of our ReplicaSet service implementation performed no verification or checksum enforcement. Only authorization policies were enforced; a client was allowed to add a replica to the ReplicaSet if the grid-mapfile of the ReplicaSet permitted this operation. Because no data transfer or checksum verification was required in this case, the time to add a replica to a ReplicaSet service depended only on Grid service overheads and not on file size.

These three implementation variations of ReplicaSet show significantly different performance, reflecting the tradeoffs that are inherent in defining and enforcing policies for replica semantics. In a highly-trusted environment, overheads are low because no verification must be performed. By contrast, an implementation that transfers files and verifies their checksums suffers a significant performance penalty, particularly for large files. It is also worth noting that our implementation performs the checksum verification only when a file is added to a ReplicaSet. An enforcement policy that required continuous monitoring of replicas to make sure that they remain consistent would suffer additional overheads.

We also examined how performance of the add operation relates to the number of services already in a container and whether service activation/deactivation is enabled. In this case, we disabled checksum verification and used the setups described in the beginning of the section.

When service activation/deactivation is disabled, the average time to add a replica to an existing ReplicaSet service instance when the container has 20 and 1000 instances is 0.152 and 0.143 seconds respectively, indicating little performance difference regardless of the number of active services.

Figure 3 shows the performance of add operations when service activation/deactivation is enabled and shows that the time to add a replica does not vary much with the number of active services. The add tests were run on services already activated, so the add times shown do not include re-activation time.

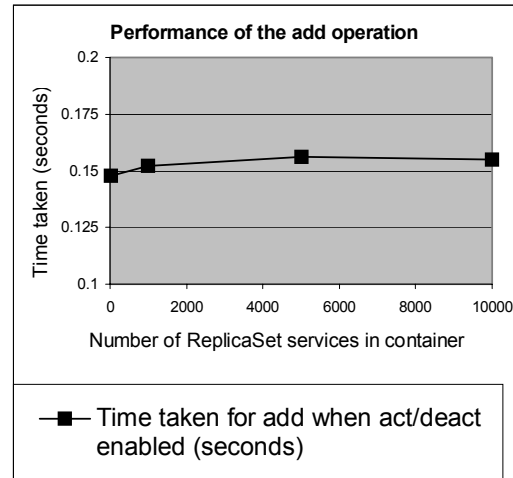


Figure 3: Time to add replicas

When an add operation is performed, our implementation writes the new replica information immediately to the database and does not wait until the service is deactivated to ensure ReplicaSet consistency. We measured the time taken to write to the database during an add operation and found it be approximately 4-10 milliseconds.

#### 5.4 Queries

The query operation we performed retrieved the list of replicas for a particular ReplicaSet instance. The services queried were selected randomly from the set of services in the container. When service activation/deactivation is disabled, the average time taken to query a single ReplicaSet instance when the container has 20 and 1000 instances is 0.506 seconds and 0.568 seconds.

Figure 4 shows the query performance when service activation/deactivation is enabled. In this case, the time taken to query the service includes the service activation time. For comparison, we also measured the time taken to read state from the database during service activation. Figure 4 shows that most of the query time is spent during reactivation in reading the service state from the database. As was the case for create and add

operations, we see that query performance does not vary much when fewer services (20 or 1000) are active, regardless of whether activation/deactivation is enabled. However, when a large number of services are deployed by enabling this feature, query performance suffers.

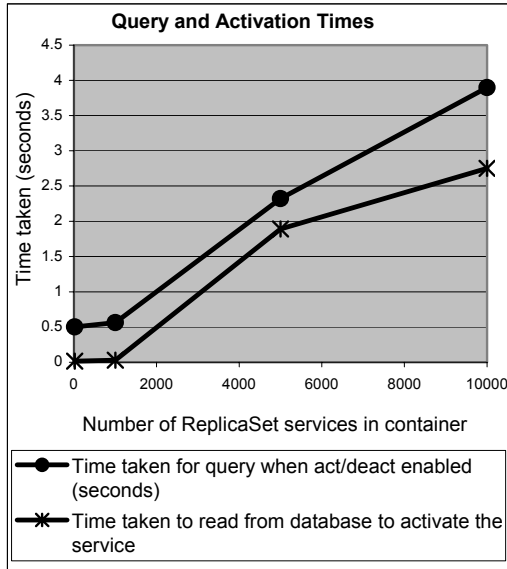


Figure 4: Time to query replicas

## 6. Related Work

In this section, we discuss other systems that perform replication and how they manage their replica location information.

The component that is most similar to our ReplicaSet implementation is the Index Service provided in the Globus Toolkit Version 3 for the Monitoring and Discovery Service [7]. The Index Service provides an interface for accessing, aggregating, generating and querying service data associated with Grid Services. Like the ReplicaSet service, the Index Service's implementation is based on OGSi ServiceGroups. However, the Index Service is more generic in its functionality and does not perform policy enforcement to validate entries added to the service.

As already mentioned, each ReplicaSet service is similar to an entry in a Replica Location Service that associates physical locations of replicas with a logical identifier for a data object. The Globus Toolkit includes an implementation of a Replica Location Service that provides a server front end and a relational database back end to hold mappings between logical names and replica locations [3, 8]. The European DataGrid Project's Reptor replica management system [9] includes a Replica Location Service implementation based on the RLS Framework [3] and implemented using Web services.

Other related Grid systems include the Storage Resource Broker [10] and Grid DataFarm [11] projects that register, discover and maintain consistency among replicas using a metadata service.

Also relevant are replication and data location systems for peer-to-peer systems, including Chord, Freenet, Tapestry and OceanStore. Distributed peer-to-peer hash table systems such as Chord [12] and Freenet [13] perform file location and replication by hashing the logical identifiers into keys. Each node is responsible for a subset of the hashed keys and searches for a requested key within its key space, passing the query to a neighbor node "near" in key-space if the key is not found locally. Tapestry [14] nodes form a peer-to-peer overlay network that deterministically associates each data object with a Tapestry location root; this root is used for location purposes. OceanStore [15] employs a two-part data location mechanism that combines a quick, probabilistic search with a slower, guaranteed traversal of a redundant fault-tolerant backing store.

## 7. Future Work

The OREP specification and the ReplicaSet service design will evolve to accommodate the WS-Resource Framework [16]. In the WS-RF, the data services that we have used as the basis of our ReplicaSet service design are stateful WS-Resources. The ServiceGroup mechanism supported in WS-RF will be similar to the one we have been using in our implementation, providing a grouping of WS-Resources. Thus, we expect that the changes to our design will require some refactoring and renaming but will not require fundamental logical changes to our model of replica location services.

Using a WS-RF version of the ReplicaSet, we will examine a variety of replica semantics and enforcement policies, including an assertion-based ReplicaSet in which a trusted client is allowed to assert a checksum associated with a data item. This would eliminate the need to transfer a file to local storage and calculate its checksum. We will also evaluate wide area performance and evaluate the scalability of the ReplicaSet service at higher request loads, especially when service activation/deactivation is enabled.

One aspect of the OREP specification that we have not yet implemented is a higher-level aggregating index [1] that aggregates information about multiple ReplicaSets and their members. Providing such indexes could improve availability by answering queries about replicaSets and their members even if a particular replicaSet service itself is unavailable. There may also be a performance advantage to aggregating replicaSet membership information and allowing bulk query operations.

## 8. Summary

We have presented a Grid service implementation of the Replica Location Service design being standardized in the OREP Working Group of the Global Grid Forum. Our implementation supports a particular set of replication semantics and enforcement policies: either no enforcement of replica semantics or enforcement that new members of a ReplicaSet are verified at the time of addition to be files with the same checksum as the master copy associated with a ReplicaSet. We presented initial performance results for our implementation. In our future work, we will experiment with a variety of additional semantic and enforcement policies and further evaluate the performance of our implementation. We will also implement a variation of our service for the Web Services Resource Framework.

## 9. Acknowledgments

This research was supported in part by the DOE Cooperative Agreements: DE-FC02-01ER25449 (SciDAC-DATA) and DE-FC02-01ER25453 (SciDAC-ESG).

## 10. References

- [1] A. Chervenak, K. Czajkowski, "OGSA Replica Location Services," Global Grid Forum OREP Working Group September 19, 2003.
- [2] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, D. Snelling, and P. Vanderbilt, "Open Grid Services Infrastructure (OGSI) Version 1.0," Global Grid Forum, Proposed Recommendation GFD-R-P.15, June 27, 2003.
- [3] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, B. Tierney, "Giggle: A Framework for Constructing Scalable Replica Location Services," presented at SC2002 Conference, Baltimore, MD, November 2002.
- [4] I. Foster, S. Tuecke, J. Unger, "OGSA Data Services," Global Grid Forum Data Access and Integration Services (DAIS) Working Group, August 14, 2003.
- [5] "The Reliable File Transfer Service." [http://www-unix.globus.org/toolkit/reliable\\_transfer.html](http://www-unix.globus.org/toolkit/reliable_transfer.html).
- [6] Globus Project, "The GridFTP Protocol and Software." <http://www.globus.org/datagrid/gridftp.html>.
- [7] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, "Grid Information Services for Distributed Resource Sharing," presented at Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), August 2001.
- [8] A. L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, R. Schwartzkopf, "Performance and Scalability of a Replica Location Service," presented at High Performance Distributed Computing Conference (HPDC-13), Honolulu, HI, June 2004.
- [9] P. Kunszt, Erwin Laure, Heinz Stockinger, Kurt Stockinger, "Advanced Replica Management with Reptor," presented at 5th International Conference on Parallel Processing and Applied Mathematics, Czestochowa, Poland, September 7-10, 2003.
- [10] A. Rajasekar, et al., "Storage Resource Broker - Managing Distributed Data in a Grid," Computer Society of India Journal, Special Issue on SAN, 2003.
- [11] O. Tatebe, et al., "Worldwide Fast File Replication on Grid Datafarm," presented at 2003 Computing in High Energy and Nuclear Physics (CHEP03), March 2003.
- [12] I. Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," presented at ACM SIGCOMM, 2001.
- [13] I. Clarke, et al., "Protecting Free Expression Online with Freenet," IEEE Internet Computing Journal, vol. Vol. 6, No. 1, pp. 40-49, 2002.
- [14] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-resilient wide-area location and routing," U.C. Berkeley, Berkeley Technical Report UCB-CSD-01-1141, April 2001.
- [15] J. Kubiatowicz, et al., "OceanStore: An Architecture for Global-Scale Persistent Storage," presented at 9th Int'l. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), November 2000.
- [16] K. Czajkowski, et al., "The WS-Resource Framework Version 1.0," March 5, 2004. <http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf>.