A Prototype Rule-based Distributed Data Management System

Arcot Rajasekar
Mike Wan
Reagan Moore
Wayne Schroeder
San Diego Supercomputer Center
University of California, San Diego
{sekar,mwan,moore,schroede}@sdsc.edu

Abstract

Current distributed data management systems implement internal consistency constraints directly within the software. If these constraints change over time, the software must be rewritten. A new approach towards data management system based on the dynamic execution of rules is discussed. By building upon the experiences gained with the Storage Resource Broker data grid, a scalable rule-based data management system can be designed.

1. Introduction

Data Grids support data virtualization, the ability to manage the properties of and access to shared collections distributed across multiple storage systems. [1] The data virtualization concept maps the physical aspects of the storage systems to a logical view of the information and data that is provided to the user and applications. Shared collections are used to organize distributed data for data grids (data sharing) [2], digital libraries (data publication) [3], and persistent archives (data preservation) [4].

The current state of the art in data abstraction, as represented by the SDSC Storage Resource Broker (SRB) [5] architecture, relies upon the following multiple levels of virtualization:

Storage access virtualization – this level of virtualization provides a single uniform interface for mapping between access APIs and the protocols used by different types of storage systems (e. g. Unix file systems, Windows FAT32, and other custom-created file systems such as SamQFS). Distributed file systems usually support a limited number of operating system types: IBM's GPFS on Unix [6], Gfarm on Linux systems [7]. Data grids provide uniform access to data stored in file systems, databases, archival storage systems, file transfer protocols, sensor data loggers, and object ring buffers [8].

Naming virtualization – this level provides naming transparency for shared collections including logical data set naming, logical resource naming, and logical user naming. It provides for access to and organization of data independent of their location, as well as authentication and authorization controls that are not dependent upon the storage repositories. The Grid File System working group of the Global Grid Forum is exploring virtualization of file names for file systems [9]. The SRB virtualizes naming across all types of storage systems [10].

Process and workflow virtualization – this level provides a virtualization of operations performed on datasets either remotely or through the use of grid computing services to execute workflows across multiple compute platforms. It automates scheduling, data movement and application execution on remote, and possibly multiple computational platforms [11]. The Kepler workflow system uses actors to define allowed operations, and controls interactions between actors [12]. Kepler actors have been developed that manage access to SRB data collections.

Information virtualization – this level provides a way to associate metadata with the data. Multiple types of metadata may be specified from simple attribute-value-unit triplets to semi-structured metadata. The gLite Metadata Catalog associates metadata with entities that are defined by globally unique identifiers [13]. The SRB system associates metadata directly with the logical file name space [14].

The data virtualization/abstraction as discussed above still does not provide complete transparency for data and information management. The additional level that is needed is "policy/constraint virtualization" which provides virtualization at the data management level as opposed to data and application (workflow) levels. The development of a constraint-based virtualization system for data management can be viewed as a new level of virtualization of data and information management.

When a user or organization stores data with associated metadata in a data grid, they apply policies to ensure that the resulting collection will meet their goals. Such policies include disaster recovery (syntactic replication) [15], persistent preservation for the long term (temporal replication) [16], caching on disk for ease of access, load balancing across resources, access by bulk operations for managing latency of wide area networks, automated metadata extraction and registration, work-flow launching, derived product generation, transformative migration of the encoding format (semantic replication) [17], etc. These policies can be applied at the level of the entire collection, or at a sub-collection level, or at the level of an individual dataset, or for a specific user and/or for specific logical resources. For example in the same data grid, one may have different policies for replication of files depending upon the importance of the sub-collection.

A constraint-based knowledge system virtualizes constraints. The rules and constraints that are used to manage state transitions within a data grid (changes to state information resulting from operations within the data grid) are explicitly defined. By naming the rules and constraints, organizing them in rule sets, and choosing the level of granularity across which the rules will be applied, we expect to generalize data management systems. The goal is to be able to change the sets of rules and add new rules without having to rewrite code. In effect, this is the virtualization of the SRB data grid control mechanisms. Constraint virtualization is needed across all levels of the system, including user access, global consistency properties, and state transitions during data grid operations.

2. Rule-based Middleware

The current architecture design for rule-based middleware is shown in Figure 1. The architecture differentiates between the administrative commands needed to manage the rules, and the rules that invoke data management modules. When a user invokes a service, it fires a rule that uses the information from the rule base, status, and metadata catalog to invoke micro-services. The micro-services either change the metadata catalog or change the resource (read/write/create/etc).

The management of rule-based middleware is being investigated from multiple perspectives. The first approach we are trying is to look at the constraints

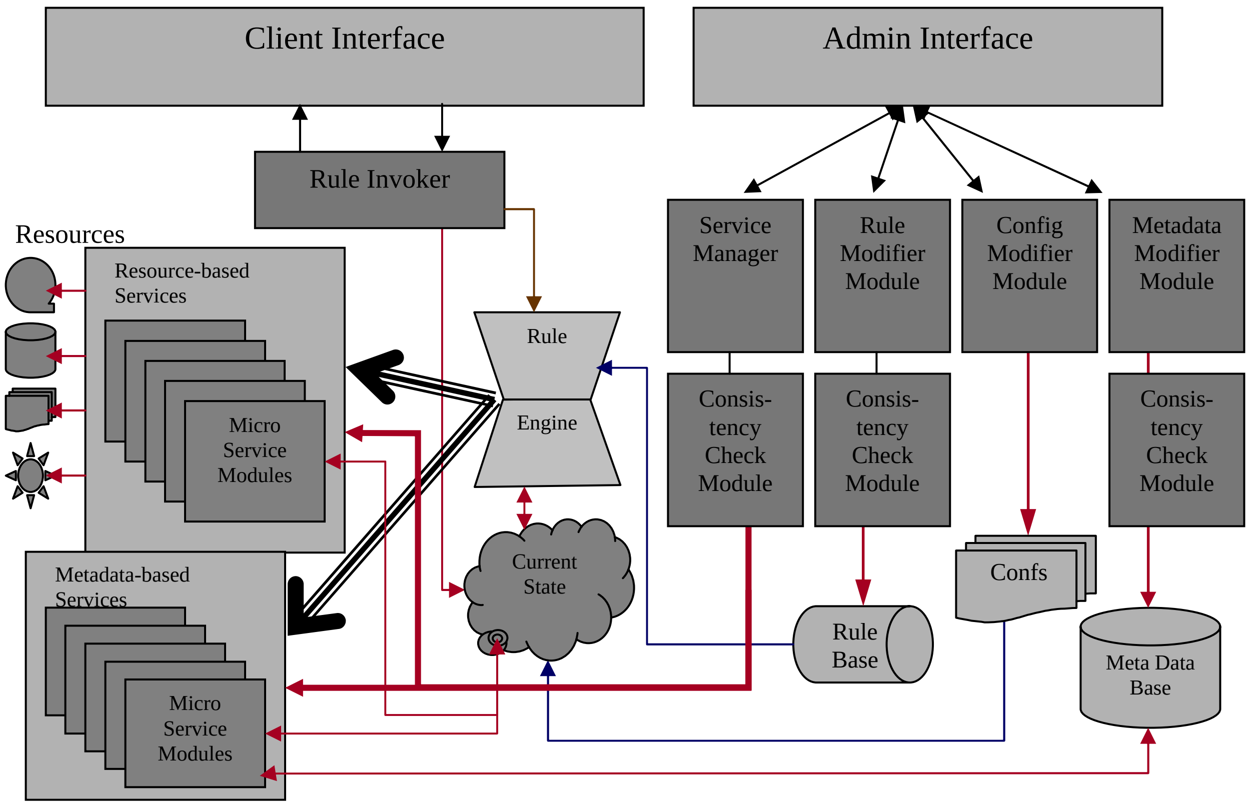

Figure 1. Rule-based middleware framework

used within the SRB to understand how they can be implemented using rule-based concepts [18]. This has required the assessment of the multiple pieces of state information in the SRB (there are more than 100) to define the associated rules and consistency constraints, and to see how they can be organized into classes of rules. The goal is to see how these classes of rules can be integrated into a data management framework without losing any functionality. This study is underway. Our aim is to use these classes as the fundamental blocks for organizing the rules. Related work is the characterization of rules used in active databases [19].

The second approach we are trying is specification of a language for consistency constraints [20]. This is an on-going activity to verify the syntax and semantics required for a language that describes the rules performed within the SRB. The research is being conducted jointly by SDSC staff, Alin Deutsch (UCSD Professor), and his graduate student Dayou Zhou.

The third approach we are trying is from the viewpoint of the collection itself. The main concept here is to consider the state information that are needed to describe manipulation of a collection in a digital library. This will be helpful in deriving rules for collection usage and access. This work is being done by Ioannis Katsis, a Graduate Student Research Assistant working with Yannis Papakonstantinou (UCSD Professor).

The fourth approach we are trying is from the perspective of a persistent archive. In this area, our goal is to define the abstractions and workflows that are needed to perform archival processes. The long-term goal is to be able to characterize the management policies needed for assuring the authenticity and integrity of records within a collection.

3. Design

In addition to the specification of requirements and development of an appropriate rule expression language, we have also begun research on the design constraints required to build a scalable system. Multiple rule-based systems are now under development within academia, including Fedora (Cornell) [21], Presto document system [22], Lifestreams storage model [23], and Haystack semantic web authoring [24]. The systems examine the use of rules to control a particular aspect of their data management system. In a separate project, we are integrating the current SRB data management system as the distributed data support infrastructure for the Fedora digital library. The integrated system will allow Fedora to assert relationships about the metadata attributes associated with the digital entities stored in the SRB. Note that this integrated system, while a form of constraint management, will not manage the constraints that the SRB must implement to guarantee consistent data grid state information. Instead, relationships between the descriptive metadata attributes about SRB digital entities will be managed independently of the SRB software by Fedora.

We are seeking a generic solution that will support rules for data placement, rules for controlling access, rules for presentation, rules for federation, and rules for data ingestion. Since the SRB already supports over 630 TBs of data with more than 100 million files at SDSC, the system must apply rule sets to massive collections.

The development of a scalable system requires the following mechanisms:

- Reification of constraints into metadata attributes. It is easier to do a join on a metadata table than it is to apply constraints while reading every object in the collection.
- Granularity. The rules may apply to a subset of a name space. Examples are use of groups to represent multiple users, use of sub-collections to represent multiple files, use of logical schema to represent a set of descriptive metadata.
- Constraint state information. We will need to maintain status information about the application of each constraint on each digital entity in the shared collection. The status information allows constraints to be dynamically changed and lazily updated.
- Collective constraints. These are assertions about levels of granularity and typically apply within a name space. An example is checking the integrity of a collection. The associated integrity rule is the periodic application of a process to validate a checksum for each file. The associated constraint state information is the date that the checksum was last validated.
- Procedural constraints. These are processes that typically require evaluation of rules that go across multiple name spaces (logical resource, logical file, distinguished user name). An example is the allowed view on metadata by a given user.

The expected problems in developing a constraint-based knowledge system will be:

- Scalability. Application of dynamic constraints on collections with a billion records.

- Deadlock. Ordering of the application of constraints to ensure that the process is computable.
- Complexity. Identification of the generic constraints that are sufficient for handling expected procedures and integrative constraints needed for data management.
- Size. Management of the constraint space itself should not take more space than the collection metadata. Since we currently support 500 attributes per digital entity, will we be able to support 1000 constraints and keep the database size manageable?

The constraints that are currently supported by the SRB can be classified by the relationships they impose on the five logical name spaces (resource names, user names, file names, metadata names, and access controls). We are investigating ways to characterize the existing constraints in the following classes.

User access rules:

- views of collections (presentation of restricted sets of metadata).
- views of files (presentation of restricted lists of files).
- allowed operations on collections (ability to specify and present only the set of operations that the user is allowed to perform).
- allowed operations on files.
- end-to-end encryption constraints.

Procedural rules:

- rules for data placement, the locations where copies are preferentially deposited.
- rules for aggregation of files into sub-directories.
- rules for access controls on storage systems.
- rules for transformative migrations on data during display.
- rules for consistency checking of metadata.
- rules for integrity checking of files (verification of checksum).
- rules for pinning on disk.
- rules for setting and releasing locks.
- rules for version creation.
- rules for aggregation into physical containers.
- rules for use of parallel I/O streams.
- rules for choice of transport protocol to manage firewall interactions.
- rules for interactions between data grids (authentication constraints). [25]
- rules for sharing of name spaces between data grids.
- rules for updating audit trails.

Global consistency properties

- assertions on the synchronization of replicas.
- assertions on the synchronization of name spaces in data grid federations.
- assertions on data organization in a collection hierarchy.
- assertions for application of access constraints.
- assertions on metadata consistency.
- assertions on name-space sharing between data grids.
- assertions on use of storage systems (maximum amount of data stored).

4. Sample Rules

We provide some rules that illustrate the concept of constraint-virtualization.

```
ingestInCollection(S) :- /* store & backup */
       chkCond1(S), ingest(S), register(S)
        findBackUpRsrc(S.Coll, R), replicate(S,R).
```

The above compound rule checks that condition-1 is satisfied, retrieves the metadata from the remote location (ingest), registers the metadata into the state catalog (register), identifies an appropriate storage location for creating a replica of the metadata and performs the replication of the metadata, registering the location of the replica into the state catalog. Each of the component tasks can be implemented as an atomic rule. Note that some of these tasks can be deferred to a later date, such as creation of a replica at a remote site when the remote system becomes available.

```
ingestInCollection(S) :- /*store & check */
      chkCond2(S), computeClntChkSum(S,C1),
      ingest(S), register(S),
      computeSerChkSum(S,C2),
      checkAndRegisterChkSum(C1,C2,S).
```

The above rule checks that condition-2 is satisfied, computes a checksum on the metadata at the remote storage location, retrieves the metadata from the remote location (ingest), computes a second checksum after the transfer, verifies the checksum is correct, registers the metadata into the state catalog (register), and registers the value of the checksum into the state catalog.

```
ingestInCollection(S) :- /store, check, backup &
                               check */
      chkCond3(S),computeClntChkSum(S,C1),
    ingest(S), register(S),
    computeSerChkSum(S,C2),
    checkAndRegisterChkSum(C1,C2,S),
    findBackUpRsrc(S.Coll, R), replicate(S,R)
    computeSerChkSum(S,C3),
    checkAndRegisterChkSum(C2,C3,S).
```

The above rule checks that condition-3 is satisfied, executes the rules from example 2, and then also locates a backup resource for the metadata, replicates the metadata, verifies a checksum on the replicated metadata, and registers the checksum value into the state information.

```
ingestInCollection(S) :- /*store, check,
                  backup & extract metadata */
    chkCond4(S), computeClntChkSum(S,C1),
    ingest(S), register(S),
    computeSerChkSum(S,C2),
    checkAndRegisterChkSum(C1,C2,S),
    findBackUpRsrc(S.Coll,R),
    [replicate(S,R) || extractRegisterMetadata(S)].
```

The above rule checks that condition-4 is satisfied, executes steps from example 3, but also in parallel with the replication of the metadata extracts additional metadata and registers the values.

The example conditions are as follows:
chkCond1(S) :- user(S) == ‘adil@cclrc’, i.e. perform the rule for only the specified user ‘adil@cclrc”
chkCond1(S) :- coll(S) like ‘*/scec.sdsc/img/*’, i.e. perform the rule for all collections that include the collection hierarchy ‘/scec.sdsc/img/ in the pathname.
chkCond2(S) :- user(S) == ‘*@nara’, i.e. perform the rule for all users within the ‘nara’ project.
chkCond3(S) :- user(S) == ‘@salk’, i.e. perform the rule for all users within the ‘salk’ project.
chkCond4(S) :- user(S) == ‘@birn’, datatype(S) == ‘DICOM’, i.e. perform the rule for all users within the ‘birn’ project for all data of type ‘DICOM’.
[OprList] implies delay execution to a later time or queue execution in a CronJobManager
Opr||Opr implies do the rules in parallel
Opr, Opr implies do the rules serially

The application of user access rules can be governed by metadata kept for each user and each record in the data grid. The application of global consistency properties can be done through workflow systems that use atomic transactions on the data grid to check the status of the records and update as necessary. The application of the procedural rules is much more difficult. It is straightforward to define the constraints that need to be checked when applying a defined data or metadata operation. The inverse is much more difficult, namely the identification of the set of constraints that might cause a particular piece of state information to change. This is needed to ensure that the application of the constraints results in a deterministic system. An example is that the state information managed by the SRB data grid depends upon the order in which the rules are applied. At the moment, these rules are hard-coded in software, so the same result occurs for a given operation. When the rules can be changed dynamically, it is not obvious that the result is well-defined, unless the interactions between the changed rule and all other rules are known. This can be reduced to a characterization of the order in which the rules must be applied for a deterministic solution. In effect, an algebra is needed on allowed composition of constraints when executing a data grid operation.

The approach that is planned for the implementation of state information virtualization is based on the phased porting of existing Storage Resource Broker modules to the new information management infrastructure. This implies that a hybrid system may be initially developed, in which some constraints are hard-coded in software, and other constraints are implemented by checking a rule base. This approach has the advantage of identifying sets of constraint rules associated with a given data manipulation operation, identifying impact of applying dynamic constraints on performance, and enabling modular development. The goal is an open-source version that has no restrictions on distribution.

5. Deferred Rule Execution

The creation of a scalable rule-oriented data system depends upon the ability to do deferred execution of rules. In the examples listed above, some of the operations can be deferred to a later time, such as the creation of a replica. The approach that has been followed in the SRB data grid has been to define state information for each object for which a deferred operation must eventually be executed. This is essential when dealing with wide-area-network latencies. Examples of deferred execution include:

- Synchronization of replicas after changing one of the copies. A “dirty bit” is turned on to indicate that all future reads and writes should be done to the modified copy. A synchronization command can be executed at a future time to update all of the replicas.

- Validation of checksums. Even after a file has been stored successfully, it may be become corrupted. Thus checksums must be continually revalidated to ensure integrity, preferably at a frequency that is four times faster than the expected degradation rate. A time stamp is needed for each object for when the checksum was last validated.
- Placement of files within a distributed storage environment. A logical resource name can be used to identify the locations of the multiple storage systems where the copies will reside. If a new physical resource is added to the logical resource name, then a copy would need to be created at the new location. A change flag is needed to denote that the replication operation should be executed.

This approach implies that the state information used to characterize the rules must be periodically checked by the data management system, A set of meta-rules will be needed to control the frequency of checking of state information, and the specification of the name-space granularity over which the rules will be applied. A tuning parameter is needed to choose between ensuring a globally consistent state (all rules are correctly enforced at all times) and an interactive system (deferred consistency).

We therefore require three types of rules within the rule-oriented data system:

- Atomic rules. These are executed immediately when the associated operation is executed.
- Consistency rules. These are executed asynchronously. Rule state information is needed that indicates when a consistency check should be applied, the level of granularity on which the rule is applied, and set of state information that is updated on execution of the rule.
- Compound rules. These are sets of rules that are required to implement an operation. To enable a scalable system, a compound rule is cast as an atomic rule and a set of deferred execution tags for the remaining rules. The goal is to avoid having interactive response wait on remote rule application.

6. Latency Management

Scalable rule execution engines that manage hundreds of millions of files require an architecture that hides rule execution latencies. The latency sources include:

- Identification of the rules that need to be executed for a given requested operation.
- Application of the rules on all files within the specified level of granularity within the shared collection.
- Application of the rules on files in a geographically distributed environment

An approach that can mask these latencies establishes state information for controlling the deferred execution of consistency constraints. The rule-oriented data system inherently manages inconsistent state information, but relies upon application of consistency rules to bring the state information back into a globally consistent state. This implies that state information is not only needed about the execution time and granularity of application for each rule on each object, but state information is also required for controlling the execution of the consistency rules.

7. Summary

Existing data grids, such as the Storage Resource Broker, hard-code consistency constraints for data manipulation. A rule-based system is being designed that will allow new consistency constraints to be specified and applied. A simplifying assumption is that the number of types of rules that are needed is limited to only three main execution scenarios:

- Atomic rules
- Deferred rules
- Compound rules which are expressed as atomic and deferred rules.

Whether the rule-oriented data system will function correctly depends upon being able to identify all rules that affect a given digital entity, and being able to identify all digital entities that a given rule can affect. The latter mapping may require an ordering on execution of rules to ensure that a consistent set of state information can be created in the future.

Finally, the reification of rules in terms of the resulting state information that is generated when the rules are applied is necessary. Each rule should be expressible as both the procedure that is applied, or a query on the resulting state information to check whether the rule has already been applied. For a scalable system, the application of a rule should only be done if absolutely required (atomic rule). In preference is the validation of the execution state of the rule by checking the state information, or the setting of flags for the deferred execution of the rule.

There will be operations for which the update of the rule state information must be done to avoid a deadlock situation, in which the global state information cannot be brought back into a globally consistent state. The identification of these situations is based on the experience with the current SRB data management system. For each operation that is performed, the SRB system manages both atomic rules and deferred rules through the creation of rule state metadata. We can build a workable system by following the same strategy as implemented in the Storage Resource Broker for differentiating between the atomic, deferred, and compound rules.

8. Acknowledgement

This work was supported in part by the NSF ITR grant on Constraint-based Knowledge Systems for Grids, Digital Libraries, and Persistent Archives, NSF SCI0438741 (NARA research prototype persistent archive supplement), and the NSF National Science Digital Library/UCAR Subaward S02-36645. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation, the National Archives and Records Administration, or the U.S. government.

9. References

1. Moore, R., C. Baru, “Virtualization Services for Data Grids”, Book chapter in "Grid Computing: Making the Global Infrastructure a Reality", pp. 409-436, New York, John Wiley & Sons Ltd, 2003.
2. Stockinger, H.,O. Rana, R. Moore, A. Merzky, “Data Management for Grid Environments,” European High Performance Computing and Networks Conference, Amsterdam, Holland, June 2001.
3. Moore, R., A. Rajasekar, M. Wan, “Data Grids, Digital Libraries and Persistent Archives: An Integrated Approach to Publishing, Sharing and Archiving Data”, Special Issue of the Proceedings of the IEEE on Grid Computing, Vol. 93, No.3, pp. 578-588, March 2005.
4. Moore, R., R. Marciano, “Technologies for Preservation”, book chapter in “Managing Electronic Records”, edited by Julie McLeod and Catherine Hare, Facet Publishing, UK, October 2005.
5. Baru, C., R, Moore, A. Rajasekar, M. Wan, "The SDSC Storage Resource Broker,” Proc. CASCON'98 Conference, Nov.30-Dec.3, 1998, Toronto, Canada, p. 5.
6. IBM – General Parallel File System, a high performance cluster file system, http://www-03.ibm.com/servers/eserver/clusters/software/gpfs.html
7. Tatebe, O., N. Soda, Y.Morita, S. Matsuoka, S. Sekiguchi, "Gfarm v2: A Grid file system that supports high-performance distributed and parallel data computing," Proceedings of the 2004 Computing in High Energy and Nuclear Physics (CHEP04), Interlaken, Switzerland, September 2004.
8. Moore, R., A. Rajasekar, M. Wan, “Storage Resource Broker Global Data Grids”, NASA / IEEE MSST2006, Fourteenth NASA Goddard / Twenty-third IEEE Conference on Mass Storage Systems and Technologies, April 2006.
9. Global Grid Forum File System Working Group, https://forge.gridforum.org/projects/gfs-wg
10. Moore, R., M. Wan, A. Rajasekar, “Storage Resource Broker: Generic Software Infrastructure for Managing Globally Distributed Data”, Proceedings of IEEE Conference on Globally Distributed Data, Sardinia, Italy, June 28, 2005.
11. Foster, I., Kesselman, C., “The Grid: Blueprint for a New Computing Infrastructure,” Chapter 5, “Data Intensive Computing,” Morgan Kaufmann, San Francisco, 1999
12. Ludaescher, B., I. Altintas, C. Berkely, D. Higgins, E. Jaeger, M. Jones, E.A. Lee, J. Tao, Y. Zhao, Scientific Workflow Management and the KEPLER System, special issue of Distributed and Parallel Systems, to appear, 2005.
13. Enabling Grids for E-sciencE data catalog, http://egee-jra1-dm.web.cern.ch/egee-jra1-dm/
14. Rajasekar, A.,R. Moore, "Data and Metadata Collections for Scientific Applications", High Performance Computing and Networking (HPCN 2001), Amsterdam, Holland, June 2001, pp. 72-80.
15. Moore, R., J. JaJa, R. Chadduck, “Mitigating Risk of Data Loss in Preservation Environments”, NASA / IEEE MSST2005, Thirteenth NASA Goddard / Twenty-second IEEE Conference on Mass Storage Systems and Technologies, April 2005, pp. 39-48.
16. Moore, R., J. JaJa, A. Rajasekar, “Storage Resource Broker Data Grid Preservation Assessment”, SDSC Technical Report TR-2006.3, Feb 2006.
17. Rajasekar, A., R. Moore, F. Berman, B. Schottlaender, “Digital Preservation Lifecycle Management for Multi-media Collections, Lecture Notes in Computer Science, vol. 3815/2005, pp. 380-384, November 2005.
18. Rajasekar, A.,M. Wan, R. Moore, “mySRB and SRB, Components of a Data Grid”, 11th High Performance Distributed Computing conference, Edinburgh, Scotland, July 2002.
19. Paton, Norman W. (Ed.), *Active Rules in Database Systems, Series: Monographs in Computer Science*, Springer, New York, 1999.
20. Deutsch, A., L. Sui, V. Vianu, D. Zhou, “A System for Specification and Verification of Interactive, Data-driven Web Applications,” SIGMOD 2006 Demo.
21. Fedora digital object repository system, http://www.fedora.info/
22. Dourish, P., W. K. Edwards, A. LaMarca, M. Salisbury, “Presto: An Experimental Architecture for Fluid Interactive Document Spaces”, ACM Transactions on Computer-Human Interaction, 6(2), 1999.

23. Freeman, E., Gelernter, D., “LifeStreams: A storage model for personal data”, ACM SIGMOD Bulletin 25, 1, (March 1996), pp. 80-86.
24. Karger, David R., K. Bakshi, D. Huynh, D. Quan, V. Sinha, “Haystack: A General Purpose Information Management Tool for End Users of Semistructured Data”, CIDR 2005.
25. Rajasekar, A., M. Wan, R. Moore, W. Schroeder, “Data Grid Federation”, PDPTA 2004 - Special Session on New Trends in Distributed Data Access, June 2004.