The SDSC Storage Resource Broker

Chaitanya Baru, Reagan Moore, Arcot Rajasekar, Michael Wan
(*baru,moore,sekar,mwan@sdsc.edu*)

San Diego Supercomputer Center
9500 Gilman Drive, Bldg. 109, La Jolla, CA 92093-0505

Abstract

This paper describes the architecture of the SDSC Storage Resource Broker (SRB). The SRB is middleware that provides applications a uniform API to access heterogeneous distributed storage resources including, filesystems, database systems, and archival storage systems. The SRB utilizes a metadata catalog service, MCAT, to provide a "collection"-oriented view of data. Thus, data items that belong to a single collection may, in fact, be stored on heterogeneous storage systems. The SRB infrastructure is being used to support digital library projects at SDSC. This paper describes the architecture and various features of the SDSC SRB.

1 Introduction

The San Diego Supercomputer Center (SDSC) is involved in developing infrastructure for a high performance distributed computing environment as part of its National Partnership for Advanced Computational Infrastructure (NPACI) project funded by the NSF. The NSF program in Partnerships for Advanced Computational Infrastructure (PACI), which funds NPACI, has the goal of providing high-end computing infrastructure for the academic scientific and engineering community. SDSC is the leading-edge site for one of NSF's two PACI grants (NCSA in Illinois is the leading-edge site for the other grant). The PACI projects are partnerships of multiple institutions and, by definition, give rise to distributed computing environments.

The NPACI computing environment consists of compute and storage resources distributed across the United States and connected via high speed data links. It includes five compute sites–at SDSC, California Institute of Technology (Caltech), University of California, Berkeley, University of Texas at Houston, the University of Michigan–and over ten data cache sites, with SDSC being the lead compute and storage resource site. The compute resources at SDSC include a 128-node IBM SP with 256MB memory per node, a 256-node Cray T3E with 128MB memory per node, a 14-processor Cray T90 with 4GB of memory, and a TERA Multi-Threaded Architecture (MTA) system [18]. The storage resources include a High Performance Storage System (HPSS) [13] archival storage system running on a 23-node IBM SP and currently capable of archiving 120TB of data.

A key requirement of the NPACI infrastructure is support for data-intensive computing, which involves both providing high performance I/O to massive data and providing digital library capabilities for storage, search, and retrieval of scientific data [2, 14, 15, 16]. As part of this infrastructure, we have de-

www.manaraa.com

veloped a Storage Resource Broker (SRB) to provide seamless access to data stored on a variety of storage resources, including filesystems, database systems, and archival storage systems. The SRB provides an API which enables applications executing in the distributed NPACI environment to access data stored at any of the distributed storage sites. The API provides the capability to do information discovery, identify data collections of interest, and select and retrieve data items that may be distributed across a wide area network. These capabilities are supported via the use of a metadata catalog and, in our previous work, we have described issues in designing such a catalog [3]. The SRB system, along with its associated metadata catalog, is currently in use within NPACI. In December 1995, an early version of the system was demonstrated as part of another project at SDSC, called the Distributed Object Computation Testbed (DOCT) project [10, 4]. In general, the SRB provides the type of functionality expected in next-generation, wide-area file systems and digital libraries [6].

As depicted in Figure 1, applications use the SRB middleware to access heterogeneous storage resources in a distributed system. The SRB middleware employs a metadata catalog service, MCAT, which manages *descriptive* as well as *system* metadata associated with data collections and system resources. *Descriptive* metadata describes the contents of entire data collections and/or individual data items, while *system* metadata provides location and access control information, again for collections and/or items. Using the MCAT service, the SRB stores and retrieves metadata about system entities including, data collections, data items, storage resources, and users. Besides providing location transparency, the MCAT service enables attribute-based access to data. This means that applications can be freed from having to provide low-level details to locate data and from the constraint of referencing data by path names. Instead, data items of interest can be dynamically identified and read, based on attributes that are meaningful to the application, thereby enabling the ability to do information discovery and automated processing of data.

Section 2 describes the salient features of the SRB. Section 3 provides an overview of the system architecture as well as details of the SRB process model, and operation of SRB servers. Section 4 provides a summary of on-going work.

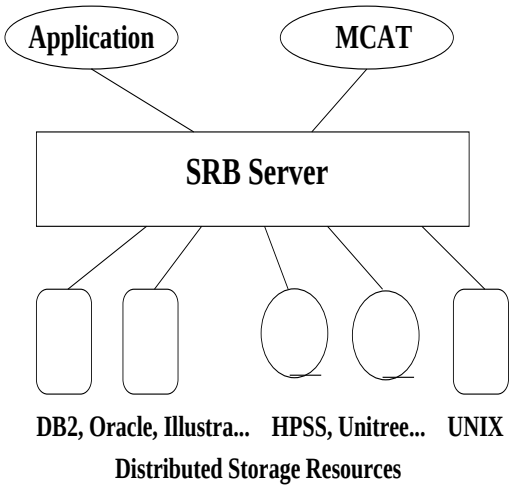

Figure 1: The SRB middleware.

2 SRB Features

The salient features of the SRB system include an API for accessing data, a metadata catalog for organizing data collection attributes, and a data handling system for supporting remote access of data.

2.1 Uniform storage interface

The SRB implements well-defined storage interfaces to a variety of heterogeneous storage resources. Currently, these interfaces include a UNIX-like file I/O interface, and an interface that supports *get* and *put* operations on storage objects. The storage resources include filesystems, database systems, and hierarchical storage management system. The SRB middleware provides a mapping from the defined storage interfaces to the native interfaces supported by each, underlying storage resource. This is done via resource-specific drivers which implement each interface for each resource.

www.manaraa.com

2.2 Metadata catalog

To support attribute-based access to data collections and items, and other system resources, the SRB employs a metadata catalog service called, MCAT, which provides a set of APIs for querying and updating the metadata catalog. The items stored via SRB are associated with *descriptive* as well as *system* metadata. Currently, a standard schema is provided for the descriptive metadata, which includes attributes similar to those specified in the Dublin Core [7]. We are currently implementing an extensible framework that will allow users to define and register schemas, since descriptive metadata may be specific to collections and/or items. The system metadata includes information used to locate and control access to data. For some metadata, e.g. location information, the actual set of attributes that are stored in the catalog may vary, depending on the type of resource.

2.3 The *Collection* Hierarchy

Data stored via the SRB is organized along a hierarchy of collections and sub-collections, defined as follows:

- A collection contains zero or more data items and zero or more sub-collections.
- A sub-collection contains zero or more data items and zero or more other sub-collections.
- A data item is a file or binary large object (BLOB).

The SRB storage model allows data items belonging to the same collection/sub-collection, to be stored in physically distributed, heterogeneous storage resources.

2.4 Hierarchical Access Control

Access to collections, sub-collections, and data items, is based on the collection hierarchy. Users may control the propagation of various privileges along this hierarchy. The set of privileges itself is extensible [5].

2.5 Tickets

A flexible mechanism for controlling read access to data is provided via the concept of *tickets*. Users with the appropriate privilege on collections/items, may issue tickets on those objects to other users. These tickets are valid either for a fixed amount of time or for a fixed number of uses. The ticket mechanism thus provides the ability to issue constrained read access to data, for any subset of users in the system. It works as follows:

- The user who has *control* privilege on a data item or collection (e.g. the creator of the object) [5], may issue tickets to a user, or group of users, on that data item or collection.
- The ticket itself is implemented as a 10-character alphanumeric string that is passed from the ticket issuer to the ticket users. A ticket user can then use this ticket to open and read data objects.

The MCAT catalog distinguishes between *registered* versus *unregistered* users. Registered users are those for whom the necessary metadata is already available in MCAT. Unregistered users are anonymous users of the SRB/MCAT system. Tickets can be issued to registered as well as unregistered users. To use tickets, unregistered users use a special call to connect to the SRB server. Such users may only read data for which they have read authorization. They are not allowed to perform any other function.

2.6 Physical Storage Resources (PSRs)

The SRB middleware manages a set of *physical storage resources (PSRs)*. A PSR is defined as follows:

- For storage resources with file system interfaces: a PSR is the (*hostname, pathname*) combination, indicating a particular directory path on a particular host.
- For storage resources with database system interfaces, a PSR is the (*hostname, database_id, table_id*) triple, indicating

www.manaraa.com

a host, a particular database on that host, and a particular table within that database. The set of parameters employed to identify a database and a table may vary, depending on the specific vendor DBMS.

2.7 Logical Storage Resources (LSRs) and Replication

A set of one or more PSRs can be combined into a single *logical storage resource (LSR)*. Thus, an LSR may contain one PSR which is a table in an Oracle database, another which is a directory path in HPSS, and a third which is a directory path in an AIX filesystem. Client APIs typically refer to LSRs, and not PSRs. Collections are implemented using LSRs, while data items are added to collections. A data item is replicated across all PSRs contained in the LSR associated with the corresponding collection. For read operations, a data item can be read from any one of the associated PSRs.

2.8 Proxy operations

The SRB provides several data handling operations that are performed without involving the client, i.e. without moving data from the server to the client (and back). Such operations are referred to as *proxy* operations, since they are performed by the SRB on behalf of the client application. Examples of such proxy operations are, *move* and *copy*, which allow users to move data directly from a source to a target resource, without involving the client.

2.9 Federated operation

Access to distributed storage resources is provided via a *federation* of SRB servers. Each SRB server controls a distinct set of PSRs. The federated SRB scheme allows one SRB server to act as a client to another SRB server. This enables a client application to access data stored anywhere in the distributed system, even if the application is not directly connected to the controlling SRB server.

2.10 Authentication and encryption

The SRB client-server communications protocol supports a variety of authentication systems. The simplest is a standard, "clear text" password-based authentication scheme. Password information is maintained by the MCAT metadata service. In addition, the SRB supports user authentication and data encryption based on the SEA system [19], which employs a public-private key mechanisms (RSA), a symmetric key encryption algorithm (RC5), and provides simple key management capabilities. Using SEA, clients may optionally request encryption on the sockets used for client-server communications. The SEA library is also used to authenticate and encrypt all inter-SRB communications during the federated SRB operation.

2.11 Activity Logs

Storage and metadata update operations performed via the SRB can be logged by the system using MCAT. When creating a data collection or data item, the user may specify whether operations performed on that object should be logged by the system. However, logging of such information is viewed as being voluntary. Therefore, a user accessing an object can turn off such logging, even if the creator of the object had requested activity logging.

2.12 Types of Applications

The goal of the SRB middleware is to improve the operating environment for scientific applications that have the requirement to access and process many scientific data sets. Currently, such applications deal primarily with a file model, where each individual application is required to keep information of the set of files of interest and store low level information, such as directory path and file name, for each file. If the data are stored in different storage systems, each application must have the interface to each storage system. Also, applications are typically able to access only local data, and not data that is distributed across a wide area.

By using the SRB middleware, scientific applications can now issue *ad hoc* queries on the

www.manaraa.com

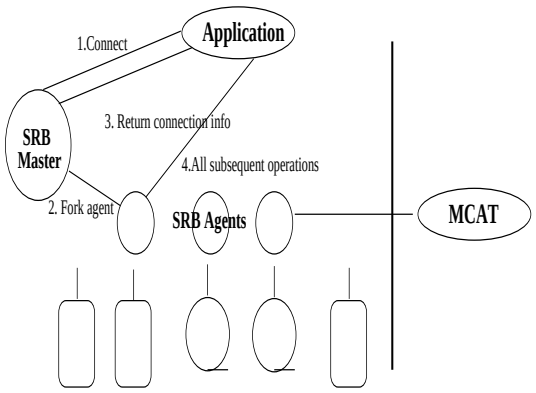

Figure 2: The SRB Process Model.

metadata to identify data of interest. They can then use the SRB API's to access data stored in distributed, heterogeneous storage resources.

3 System Architecture

The SRB middleware consists of one or more SRB Master daemon processes and SRB Agent processes associated with each Master. Each SRB Master is uniquely identified by its (*host-name, port number*). Each Master also controls a distinct set of PSRs. Conversely, each PSR is controlled by a single SRB Master.

The Master monitors its well-known port for connection requests from clients. Clients and servers communicate via sockets, using TCP/IP connections. Figure 2 shows the steps involved in connection processing, which are:

- *Step 1:* Client issues a connect request to the SRB Master, which first authenticates the client, possibly using the SEA library.
- *Steps 2,3:* The SRB Master forks a SRB Agent to service the authenticated connection and returns the connection handle to the client. Each client connection is serviced by a distinct SRB Agent.
- *Step 4:* The client uses the SRB Agent for all subsequent communications.

A client may establish only a single connection to a given SRB Master. However, it may establish concurrent connections to multiple, distinct SRB Masters. A single connection may be used to access multiple data items, possibly stored on different PSRs.

The SRB Agent uses the MCAT metadata service to obtain the necessary system metadata needed for processing client storage requests. For example, the client may issue a request to *open* a data item identified by the name of the collection and the name of the item. This requires the SRB to access MCAT to obtain the corresponding LSR and PSR information. For example, if the data item was stored in an HPSS PSR, MCAT would contain the neccesary information for connecting to the particular HPSS server and for locating the file within that server. If the PSR is not under control of the current SRB Master (i.e. the Master through which the client has connected), then the SRB federation mechanism is invoked to obtain access to the appropriate PSR containing the data item. This is further described in Section 3.2.

3.1 The Metadata Catalog Service, MCAT

Data items managed by the SRB system are referenced by the collection/sub-collection under which they reside, and by their name. This is similar to a directory name and file name in a file system. However, in the file system paradigm, the directory/file name also implies the physical location of the file. In contrast, the physical location of a data item stored via the SRB is distinct from the logical collection hierarchy under which it resides and the logical name of the file. Data items in the same collection may, in fact, reside on different storage systems. Thus, a given collection may contain several data items with one residing in a local UNIX filesystem, another residing in a remote database system, and a third residing in a remote archival storage system.

The metadata catalog is used to record location information for storage resources (PSRs) as well as for data items. The catalog also contains metadata that is used for implementing hierarchical access control, the collection/sub-collection hierarchy, and the ticket mechanism. The catalog metadata also describes the contents of collections and/or data items. This enables attribute-based querying and identifi-

www.manaraa.com

cation of data. We are currently implementing mechanisms for registering new metadata schemas and querying/updating their contents.

The MCAT catalog has been implemented in DB2 UDB and Oracle. In the current implementation, the catalog may be used either as a a central resource or as a local resource. In the former case, all metadata requests are directed to a single, central catalog. While the access to the data item itself does not require accessing the catalog, the central catalog may still cause undue communication overheads and create a resource bottleneck for metadata access. While we have not yet reached that stage in our deployed systems, we are investigating the possiblity of replicating MCAT itself (for example, using the built-in replication capabilities provided by DB2 UDB or using the Oracle Replication Server).

In the case where MCAT is used as a local resource in a distributed system, different applications could use different MCAT catalogs depending on the SRB Master to which they connect. However, each catalog can only be accessed individually, i.e. we do not yet support federation of MCAT catalogs.

3.2 Federated Servers

In a distributed system, one may choose to control different storage resources using different SRB Masters, due to a variety of technical and administrative reasons. For example,

1. different storage resources may be under different administrative domains, each of which wishes to maintain its own SRB environment and, consequently, runs its own SRB Master

2. system performance may be improved by running multiple SRB Masters and allowing clients to connect to different Masters, in order to access different storage resources

3. system availability may be improved by having different SRB Masters control PSRs on different hosts, and by replicating data across such PSRs. Synchronization among replicas is handled as follows. When an application creates a replicated data item and writes to it, the SRB system ensures that this data is written to all replicas. If there is a failure during the creation or writing of any of the replicas, then an "inconsistent" flag is set for the failed replica(s). If the application so chooses, it may continue writing into the other replica(s). When an application opens a data item for read, it can choose to open any one of the associated replicas. However, if a replica has its "inconsistent" flag set, then the system will not allow that replica to be opened.

   Inconsistent replicas can be "synchronized" with consistent replicas via a *replicate* operation which copies data from the most up-to-date replica into all other replicas.

4. some storage systems may not offer efficient support for client-sever operation, thus requiring a SRB Master to run on the same host as the storage system itself

5. in systems such as HPSS, the SRB Master may be required to run under the same DCE cell as the HPSS system. Thus, the existence of multiple HPSS systems may require running multiple SRB Masters.

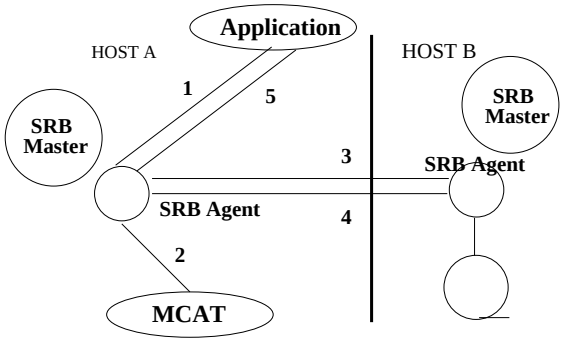

Figure 3: Federated SRB Operation.

Figure 3 shows the operation of a federated SRB system consisting of two hosts, A and B, each of which runs a SRB Master. For example, a request to *open* a data item, D, on Host A results in the following sequence of events. The client on Host A connects to the SRB system. After successful connection:

www.manaraa.com

1. The SRB client issues its request to open the data item, D, for reading.

2. The SRB Agent on Host A refers to the MCAT catalog service and determines that D is in a PSR controlled by the SRB Master running on Host B.

3. This agent then issues a connect request to the SRB Master on Host B and passes the original open request. The Master on Host B authenticates its client (SRB agent) and then services this open request. A file handle is returned to the agent on Host A.

4. The agent on Host A returns this file handle information to its client.

5. The client application on Host A may now access the data item in Host B, using the returned file handle.

In the above scenario, an option for the client application on Host A is to determine, at the outset, the address of the SRB Master which controls the data item, D. In the case of replicated data, there could be multiple masters. The client can then directly connect to one of the masters and thereby by-pass the federated SRB mechanism, if desired.

3.3 The SRB Agent Design Details

The major internal software components of the SRB Agent are shown in Figure 4. The *Dispatcher* module monitors incoming client requests and dispatches requests to the *High-Level Request Handler* or the *Low-Level Request Handler.* The Dispatcher is also responsible for returning results to clients. The difference between the two request handlers is that the high-level handler maps user names and data item names to access control and location information using MCAT, whereas, the low-level handler expects the caller to provide the detailed parametric information necessary for executing a particular storage request. The typical parameters that are passed at this level are *hostname*, *storage system type*, and *physical location* information. Only privileged SRB users are allowed to directly invoke the APIs associated with the low-level handler.

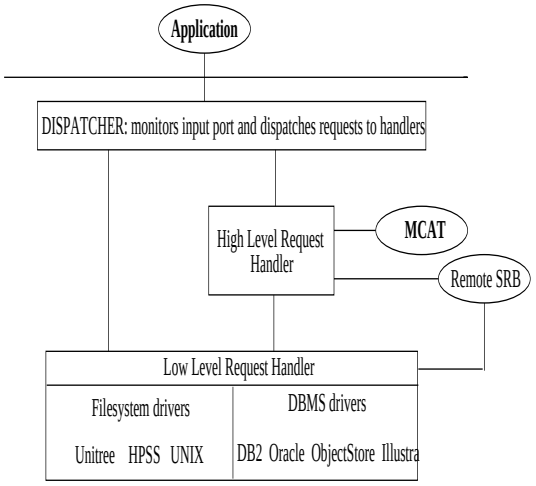

Figure 4: The SRB Agent Details.

Based on the native interfaces supported by storage resources, the correpsonding SRB drivers are grouped into two types, *filesystem drivers* and *DBMS drivers.* Storage resources that have filesystem drivers include UNIX (and its variants), HPSS, ADSM, and UniTree. DBMS drivers are used for resources that provide a large-object (or, BLOB) interface to storage, e.g. DB2, Oracle, Illustra, and Objectstore.

3.4 Client API's

Several sets of client APIs are provided to allow SRB clients to perform a number of different types of functions, which are discussed briefly in the following subsections.

3.4.1 Query/update of metadata

A set of APIs are provided for querying and updating the information in the MCAT catalog. The sample subset is shown in Appendix A.3. For example, the *srbGetDataItemInfo* API can be used to query the metadata attributes of data items. This set of APIs allows applications to manage metadata associated with data collections, data items, users, user groups, and storage resources. As mentioned above, the high-level request handler in the SRB middleware is itself a client to the MCAT catalog and uses these APIs.

www.manaraa.com

3.4.2 Connecting to the server

Appendix A.1 shows the set of API calls related to establishing and using client/server connections. These include the usual connect and disconnect (*clFinish*) calls as well as a special type of connect call used by clients holding SRB tickets. For a "ticketed connection," the system does not perform user authentication. Instead, it verifies that the ticket offered by the client is associated with the data collection/item that the client wants to access and encodes the necessary level of permission to perform the requested operation on the specified object.

3.4.3 Creation of data items

The API used for creating new objects is listed in Appendix A.2 (*srbObjCreate*). When a client issues a request to create an object, the SRB first obtains access control information from MCAT to verify user permissions. If the user has the privilege to create a data item under the specified collection in the requested storage resource, then the system generates a unique physical path name (e.g., UNIX or HPSS path name, or database LOB id) in that storage resource. Next, a *create* request is issued to the low-level request handler. If the create is successful, then the corresponding metadata is registered in MCAT, including information about the data item name, the associated collection, the corresponding PSR, and the user name. Finally, a data item handle is returned to the client which is used in subsequent read/write calls. If the call to register the data is unsuccessful, a low-level unlink call is issued to remove the data item and an error is returned to the client.

3.4.4 Open/read/write/delete of data items

Appendix A.2 lists some of the APIs for open, close, read, write, and unlink of data items as well as APIs to perform replication and grant tickets.

The open operation requires querying MCAT to obtain the necessary metadata for the data item. The read/write operations are direct storage operations which need not involve the MCAT service (unless activity logging is turned on). Deletion of a data item requires updates to MCAT to "de-register" the data item.

4 Related Work

Work related to the SRB includes the scientific data archive effort at the Pacific Northwest National Labs (PNNL) [1], the AFS filesystem [12, 20], the Synopsis project [6], DB2 Datalinks [8], and the IBM Digital Library product [9]. The PNNL project provides a metadata catalog to query attributes of data sets that are stored in an archival storage system. The system is restricted to accessing data from archives and does not support remote filesystem and/or database systems. AFS provides support for wide area UNIX-like filesystems. Thus, it provides a file I/O interface to distributed files. However, it does not integrate access to database systems (DBMS) and hierarchical storage management (HSM) systems (note that there is a current effort to provide a DFS interface to the HPSS HSM system). In addition, AFS does not provide the ability to query metadata associated with a data item to identify items of interest. The Synopsis system provides an object-oriented filesystem and the ability to query metadata associated with objects. However, it does not integrate access to resources such as DBMSs and HSMs. It does provide certain useful, object-oriented capabilities which are not currently available in the SRB/MCAT system, such as the ability to recognize the type of an object and use this information for appropriate display of an object.

The IBM Datalinks is a feature introduced in the recent version of IBM's DB2 UDB database product which provides the ability to store URL's referring to external file objects within a database, and provides referential integrity and recovery for such files in conjunction with the associated database. This may be used to provide a "tight" linkage between metadata stored in the database and the corresponding files which are stored externally. The external files may be stored on disk or in an HSM (Datalinks currently only supports the ADSM HSM system).

The IBM Digital Library product consists of a set of distributed servers including *Li-*

www.manaraa.com

brary Servers and *Object Servers.* The metadata is stored in the Library Server while the actual data objects are in the Object Server. The object servers have the ability to migrate data to an HSM. The product also supports text and image indexes. Thus, a client may query the metadata stored in the library server, which may result in a SQL query on a relational database, as well as a search request on a text index and/or an image index. The result is a set of object handles which can be used to retrieve the object(s) from the object server. Note that there is the possibility of using the Datalinks feature to provide tight linkage between the Library Server and the Object Server.

The SRB system differs from those mentioned above in that it provides a simplified SQL-like interface to the metadata with a file I/O interface to the actual data item, which may reside in a filesystem, database system, or archival storage system. Thus, the SRB system supports more types of storage resources. It has been implemented for a variety of storage systems and OS platforms, as listed in the next section.

5 Current Status

The SRB system is currently in release 1.2. Both the client library as well as the server have been implemented for AIX, Sun Solaris, SunOS, SGI Irix, DEC OSF, Cray C90, and Cray T3E. The MCAT catalog has been implemented using Oracle as well as DB2. The UC Berkeley ELIB [17] and UCSB Alexandira Digital Library (ADL) [11] are both being mirrored at SDSC. In both cases, the SDSC implementation employs the SRB to provide seamless access to data that are stored in the HPSS hierarchical storage management system. Other applications currently using the SRB technology include the Environmental Sciences digital library (ESADR).

For the next release of the SRB, we are working on items related to support for extensible schemas, application-specific schemas, registration and execution of user-defined proxy operations (UDPOs), and metadata schema for image data. We are also investigating various type-specific operations (e.g. for image data) that can potentially be implemented as *built-in* proxy operations. System performance is also an item for study. We wish to study the impact of a central MCAT catalog. If this is a significant bottleneck, we will investigate catalog replication and/or partitioning as approaches to solving the problem. Also, while the SRB middleware is a relatively "thin" layer, we are interested in studying the impact of its path length on the overall system performance. Another aspect is the impact on performance of having a large number of concurrent users in the distributed SRB system.

Acknowledgements

The authors are members of a larger group that has contributed to the design, development, and deployment of the SRB system. At SDSC, Richard Frost helped develop several of the initial concepts of SRB and MCAT. Subsequently, he also helped develop the test suites and installation scripts for the SRB/MCAT software. Wayne Schroeder is the architect and implementor of the SEA system, and also implemented the SRB drivers for Illustra and ftp. Richard Marciano helped with the design of MCAT catalog tables and also did some early experiments with a prototype, LDAP implementation of the catalogs. The SRB Oracle driver was implemented by Randall Sharpe and Robert Templeton of NCSA, Illinois. The ObjectStore driver was implemented by David Wade of SAIC. Interfacing the SRB to the ADSM system at the University of Michigan was done by Thomas Hacker at Michigan.

This work is supported by grants from the NSF, under its PACI program, and by DARPA Project F19628-95-C-0194 (MDAS) and Project F19628-96-C-0020 (DOCT).

About the Authors

Chaitanya K. Baru: is Senior Principal Scientist and Manager of the Data Intensive Computing Environments (DICE) group at SDSC. He leads the development activities within the DICE group and is one of the designers of the SRB/MCAT system. His research interests

www.manaraa.com

are in database systems, semistructured data, mediation-based systems, digital libraries, and archival storage systems. Prior to joining SDSC in 1996, he was one of the team leaders of IBM's DB2 Parallel Edition Version 1, at the IBM Toronto Labs. He received his Ph.D. in Electrical Engineering from the University of Florida, 1983.

Reagan W. Moore: is an Adjunct Professor in the CSE department at the University of California San Diego, and the Associate Director for Enabling Technologies at the San Diego Supercomputer Center. His areas of interest include Information Based Computing and the development of infrastructure needed to support scientific data collections. He has a PhD in Plasma Physics from UCSD, 1978.

Arcot Rajasekar: is a Staff Scientist in the Data Intensive Computing Environments group at SDSC, and a member of the SRB development team. He leads the development of the MCAT Metadata Catalog. His research interests include digital library systems, data mining and database-oriented reasoning, and logic programming and default reasoning. He received his Ph.D. in Computer Science from University of Maryland, 1988.

Michael Wan: is a Senior Staff Scientist in the Data Intensive Computing Environments group, and member of the SRB development team. He is responsible for the overall architecture and engineering of the SRB software and has implemented most of the data handling functionality of the SRB client and server software. He has previously worked on a variety of kernel/system-level projects including, kernel level memory and job schedulers, an application level job mix scheduler, network device driver, and trouble-shooting of the Intel Paragon OSF-based kernel. He received his MS in Nuclear Engineering from Georgia Institute of Technology in 1972.

References

[1] D. Adams, D. Hansen, K. Walker, and J. Gash. Scientific data archive at the Environmental Molecular Sciences Laboratory. In *Procs. of Sixth Goddard Conference on Mass Storage Systems and Technologies, p.409*, Silver Spring, MD, 1998.

[2] C. Baru. Archiving Metadata. In *Procs. of European Conf. on Digital Libraries (EuroDL*, Crete, Greece, 1998.

[3] C. Baru, R. Frost, J. Lopez, R. Marciano, R. Moore, A. Rajasekar, and M. Wan. Metadata design for a massive data analysis system. In *Procs. of CASCON'96*, Toronto, CA, 1996.

[4] C. Baru, R. Moore, A. Rajasekar, W. Schroeder, M. Wan, R. Klobuchar, D. Wade, R. Sharpe, and J. Terstriep. A data handling architecture for a prototype federal application. In *Procs. of the 6th Goddard Conference on Mass Storage Systems and Technologies*, Silver Spring, Maryland, 1998.

[5] C. Baru and A. Rajasekar. A Hierarchical Access Control Scheme for Digital Libraries. In *Procs. of the ACM Conf. on Digital Libraries*, Pittsburgh, PA, 1998.

[6] M. Bowman and B. Camargo. Digital Libraries: The Next Generation in File System Technology. *D-Lib Magazine, February*, 1998.

[7] Dublin Core. Dublin Core Metadata. In *http://purl.org/metadata/dublin_core*, 1997.

[8] J. Davis. Datalinks: Managing external data with DB2 Universal Database. In *http://www.software.ibm.com/data/pubs/papers/datalink.html*, IBM Corporation, 1998.

[9] IBM DL. IBM Digital Library. In *http://www.software.ibm.com/is/dig-lib*, IBM Corporation, 1998.

[10] DOCT. The Distributed Object Computation Testbed Project. In *http://www.*

www.manaraa.com

sdsc.edu/DOCT, San Diego Supercomputer Center, La Jolla, CA, 1998.

[11] J. Frew, M. Freeston, R. Kemp, J. Simpson, T. Smith, A. Wells, and Q. Zheng. The Alexandria Digital Library testbed. *D-Lib Magazine, July/August*, 1996.

[12] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayana, N. Sidebotham, and M.J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems, 6(1)*, 1996.

[13] HPSS. High Performance Storage System. In *http://www.sdsc.edu/HPSS*, San Diego Supercomputer Center, La Jolla, CA, 1997.

[14] R. Moore. Enabling petabyte computing, The Unpredictable Certainty, Information Infrastructure through 2000. *National Academy Press*, 1997.

[15] R. Moore, C. Baru, S. Karin, and A. Rajasekar. Information based computing. In *Procs. of the Workshop on Research and Development Opportunities in Federal Information Services*, Washington, D.C., 1997.

[16] R. Moore, R. Marciano, M. Wan, T. Sherwin, and R. Frost. Towards the interoperability of Web, database, and mass storage technologies for petabyte archives. In *Procs. of the 5th Conference on Mass Storage Systems and Technologies*, Silver Spring, MD, 1996.

[17] V. Ogle and R. Wilensky. Testbed Development for the Berkeley Digital Library Project. *D-Lib Magazine, July/August*, 1996.

[18] W. Pfeiffer. Evaluation of a Multithreaded Architecture for Defense Applications. In *http://www.sdsc.edu/mta_eval/darpa*, San Diego Supercomputer Center, La Jolla, CA, 1997.

[19] W. Schroeder. The SDSC Encryption and Authentication (SEA) System. *Concurrency Practice and Experience, April*, 1998.

[20] A. Spector and M. Kazar. Wide-area file service and the AFS experience. *UNIX Review, 7(3)*, 1989.

APPENDIX A

A.1 Examples of client/server connection APIs

This set of APIs handles opening and closing a connection to a SRB server.

clConnect - Initiate a connection to a SRB Master.
tiUserConnect - Initiate a connection to a SRB server by a user holding a "ticket" token. Normal user authentication will be bypassed for such users.
clFinish - Close an existing SRB connection.
clErrorMessage - Return server error message, if any, associated with the most recent client call.

A.2 Examples of APIs associated with data collections/items

The following set of API's are used for performing standard storage operations.

srbCreateCollect - Create a data collection.
srbListCollect - List a data collection.
srbModifyCollect - Update attributes of a data collection.
srbObjCreate - Create a data item.
srbObjOpen - Open a data item.
srbObjOpenWithTicket - Open a data item using a ticket.
srbObjClose - Close an opened data item.
srbObjUnlink - Unlink a data item.
srbObjRead - Read a block of data from a data item into buffer.
srbObjWrite - Write the content of a buffer to a data item.
srbObjSeek - Change the current read or write location for a data item.
srbObjProxyOpr - Execute specified proxy operation.
srbObjReplicate - Replicate data item.
srbObjMove - Move a data item to a new location.

www.manaraa.com

srbIssueTicket - Issue a ticket on a data item/collection.
srbRemoveTicket - Cancel a ticket on a data item/collection.

A.3 Example of MCAT APIs
This is a sample set of API's for querying and updating information in the MCAT catalogs.

srbGetDataItemInfo - Get values of attributes associated with a SRB data item.
srbGetDataDirInfo - Get values of attributes associated with collections.
srbModifyDataItem - Modify attributes of a data item.
srbSetAuditTrail - Set/reset activity logging.
srbChkMdasAuth - Authenticate a username/passwd.
srbChkMdasSysAuth - Authenticate a user-Name/password for system administrator-level access.
srbRegisterUserGrp - Register a new user group in the system.
srbRegisterUser - Register a new user in the system.
srbModifyUser - Modify attributes of a user info.
srbGetPrivUsers - Get the current set of privileged users.
srbGetMoreRows - A cursor-like operation to get further rows from a result set created by a previous operation such as, srbGetDataset-Info, srbGetDataDirInfo, srbListCollect or srbGetPrivUsers call.

www.manaraa.com