International Journal on Digital Libraries manuscript No.
(will be inserted by the editor)

Scale and Performance in Semantic Storage Management of Data Grids

Stergios V. Anastasiadis[*],[**] **and Syam Gadde**[**] **and Jeffrey S. Chase**[*]

[*]Department of Computer Science, Duke University, Durham, NC, 27708, USA
[**]Duke-UNC Brain Imaging and Analysis Center, Durham, NC, 27708, USA

Received: December 1, 2003 / Revised version: May 1, 2004

Abstract. Data grids are middleware systems that offer secure shared storage of massive scientific datasets over wide area networks. Main challenge in their design is to provide reliable storage, search and transfer of numerous or large files over geographically dispersed heterogeneous platforms. The Storage Resource Broker (SRB) is an example of such a system that has been deployed in multiple high-performance scientific projects during the past few years. In the present paper, we take a detailed look at several of its functional features, and examine its scalability using synthetic and trace-based workloads. Unlike traditional file systems, SRB uses a commodity database to manage both system and user-defined metadata. We quantitatively evaluate this decision, and draw insightful conclusions about its implications to the system architecture and performance characteristics. We find that the bulk transfer facilities of SRB demonstrate good scalability properties, and we identify the bottleneck resources across different data search and transfer tasks. We examine the sensitivity to several configuration parameters, and provide details about how different internal operations contribute to the overall performance.

Key words: data grids, middleware systems, distributed storage systems, semantic web

Correspondence to: Stergios V. Anastasiadis, Department of Computer Science, Duke University, PO Box 90129, Durham, NC 27708, USA, Email: stergios@cs.duke.edu

1 Introduction

Applications in scientific and enterprise data mining currently drive the demand for storing and processing massive amounts of data [28]. Not surprisingly, accelerating rates in data acquisition make semantic querying a problem of crucial importance for the effective operation of data storage systems in both high-performance research and personal computing environments [7]. Arguably the management of user- defined and system metadata becomes the most critical component in the architecture of data sharing services, and finding ways to streamline the metadata operations can have significant implications to the efficiency and scalability of the entire system.

The Storage Resource Broker (SRB) is a data discovery and sharing facility developed at the San Diego Supercomputer Center for serving the storage and computational needs of modern scientific research [4]. It combines multiple heterogeneous storage servers with a decentralized database into a distributed facility that supports secure storage and search for scientific data. The system allows data owners to attach semantic information to datasets, upon which authorized users can base keyword and range queries to search and retrieve data.

During the last five years, several SRB installations have been deployed to support terabytes of data and millions of files in astronomy, physics, medical imaging, and molecular sciences [20]. Unlike traditional file system architectures that are usually based on custom- developed data structures, SRB is a data storage middleware facility that relies on commodity database software for its metadata management. Making a commodity database major component of a large-scale distributed facility adds significant flexibility and extensibility into the features supported by the system. It also creates major challenges in maintaining the efficiency of the system operation and amortizing the cost of database transactions across data accesses.

The semantic description of scientific and commercial datasets creates a building block towards (i) the construction of data discovery facilities with richer interfaces, and (ii) the integration of shared data and applications into distributed computational workflows [7, 24]. As the world wide web infrastructure is enriched with data interpretation capabilities, semantic storage systems will become essential for searching and accessing effectively and securely distributed data resources. Search engines will index complex datasets more precisely, while distributed services more easily will automatically recognize and handle different data formats.

In the present paper, we examine several aspects of the SRB functionality, and experimentally evaluate its performance using both synthetic and trace-based workloads. We identify performance bottlenecks across different supported operations, examine interesting scalability properties in specialized data search and transfer commands, and investigate the sensitivity of the system responsiveness to the namespace organization and the network latency. Additionally, we measure the efficiency of alternative authentication schemes that the system supports, and study the behavior of the system when it is accessed through a regular file system interface. We find that specific relational database queries and tables are used repeatedly, which points into potential targets for further tuning and optimization of the system.

Our main contributions include: (i) clarification of the architectural and performance objectives of semantic data storage systems, (ii) comprehensive bottleneck and sensitivity analysis across different operations and parameters, (iii) identification of repetitive internal tasks whose optimization can improve the performance of such systems, and (iv) better understanding of the advantages and challenges introduced when using commodity software for metadata management. Several of our conclusions can be generalized to a broad range of systems with similar properties, even though our evaluation is based on a specific hardware and software setup.

The rest of the paper is organized as follows. Section 2 outlines the architecture of the SRB, while Section 3 describes the performance metrics and the environment used for the performance evaluation. Section 4 analyzes the performance measurements that we did, and Section 5 summarizes previous related work. We present our conlusions in Section 6.

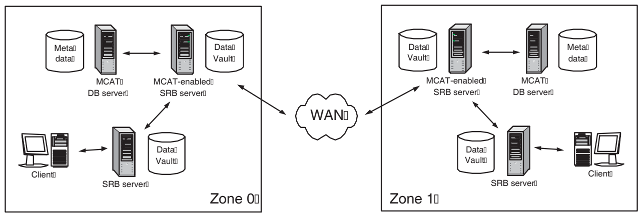

Fig. 1. Example of federated Storage Resource Broker installation consisting of multiple MCATs.

2 System Overview

The Storage Resource Broker (SRB) combines multiple storage servers with a decentralized database into a data discovery and sharing facility deployed over a wide-area network. Different clients can access the entire system by attaching to a specific storage server over the network. A client can submit queries for finding datasets that satisfy certain search criteria, and data transfer requests for accessing the discovered datasets. Overall, the functionality of the SRB can be summarized into the following four components: (i) search engine, (ii) access control, (iii) access auditing, and (iv) data storage. Different subsets of the system operations are accessible through standalone command-line binaries, graphical window environments, application programming interface libraries, and a unix-like file system interface.

2.1 Metadata Search and Management

The *Metadata Catalog (MCAT)* is an information management facility used for managing the metadata of SRB. Both the data model used by the MCAT for the internal structuring of the metadata, and also the exchange format used for the metadata communication with the external world are described by the *Metadata Attribute Presentation Structure (MAPS)* specification. Although, in principle, the metadata information can reside on alternative catalog management systems, the current implementation only stores the metadata as a collection of relational tables managed in a commodity database, such as Oracle or DB2.

The metadata information managed by the MCAT contains four different types of entities (i) users, (ii) resources, (iii) data objects, and (iv) methods. Each user entity is associated with metadata that include a unique identifier and the user contact information. The user namespace is partitioned across tree- structured directories, called *domains*. Also, each user can arbitrarily belong to at least one, or multiple groups. The *storage vault* is the pair of a host address and a directory path where data can be stored in a storage server. The *physical resource* associates a name with a particular storage vault. Multiple physical resources can be grouped into a *logical resource* with abstract properties related to data replication or striping, and write-once or read-only access permissions.

Each data object corresponds to an actual data entity. Supported data objects include files accessible to Unix and Windows file servers, FTP and HTTP servers, or database objects. Object system attributes include a unique identifier, the object name, the owner, the physical resource, and the access location of the data. Other related metadata include the data size, the access timestamps, and the replica number for the case a data object is replicated. The data objects are grouped into tree- structured directories called *collections*. Each collection can contain collections that reside on different physical resources. Methods provide application- specific computation through well-defined input and output interfaces. Invocation of remote execution is possible through special calling interfaces.

Each collection and data object can be associated with a user-defined set of attributes. The system currently supports only string and integer attributes, whose number is only limited by the underlying implementation. A metadata query can return the attribute values of a particular object. Alternatively, a query can request the identifiers of all the objects that satisfy specific conditions on the attribute values.

2.2 Storage Management

The SRB middleware provides unified access to multiple distributed storage servers connected in peer-to-peer fashion. Both the SRB storage servers and the metadata are partitioned into *zones* with each zone managed by a single Metadata Catalog (MCAT). Each MCAT is accessible by a single MCAT- enabled storage server, and is capable of caching metadata of other MCATs. A client binds to a single storage server through which it accesses the data of the entire SRB installation. Each storage server resolves access requests by submitting queries to the MCAT of its zone, and directly communicating with the SRB server where the requested data reside (Figure 1).

A *high-level interface* provides communication between a client and the MCAT through one or multiple SRB servers. It allows creation and deletion of datasets, access to data, and query handling related to the entities supported by the system. A *low-level interface* provides direct access to the remote data storage resources by using metadata information provided by the high-level interface. In a typical sequence that creates a dataset, the server submits an MCAT query to check if the client is allowed the specific operation. If successful, the query returns the host name and path location of the collection where the dataset will be created. A low-level request creates the file in the appropriate location, while a high-level request registers the file to the MCAT and passes all the necessary metadata. The successful registration request returns to the client the file handle for the low-level write calls that will transfer the data to the new data object.

When a large number of data objects have to be created or accessed on a remote storage server, data communication can be reduced by using the *container* entity. Each container is associated with one or multiple storage servers. Data access requests can be handled by the local server thus reducing the communication overhead. Special calls can be used for explicitly synchronizing the local storage server of the container with the remote server. All the metadata involved in translating data object requests to container requests are still maintained by the MCAT. There is no data caching functionality transparent to the users unlike most existing storage systems. However, metadata caching is supported in some cases depending on the client software used for connecting to the server.

2.3 Access Control

In this section, we give an overview of the SRB security mechanisms that we evaluate experimentally in the present paper. Additional details about the security architecture of SRB can be found in previous published literature [4]. Each data object and collection is associated with an access control list that specifies the access permissions granted to individual users or groups: (i) read, (ii) write, (ii) control, and (iv) grantTicket. The system verifies the identity of a user before it will establish a secure connection between the user and a storage server. One simple authentication method supported by SRB uses a plain-text or encrypted password stored on the home directory of the user client host. Each time a user submits a request to the system, the server will use the user's password to verify one's identify.

An alternative more secure method relies on the Grid Security Infrastructure (GSI) for mutual authentication between the user and the system [14, 27]. A user generates locally a public-private key pair, and has the public key signed into a certificate by a trusted authority. Similarly, a public-private key pair and a certificate are generated for the SRB itself. When a user attempts to access the SRB service, both the system and the user verify the identity of the authority who signed the certificate of the other. Then, based on their certificates and public keys they can establish mutual trust and proceed into a secure connection. Users can also generate proxies, or additional public-private key pairs and certificates signed by themselves rather than by a certificate authority.

A different method for authorizing data accesses within SRB makes use of ten-character strings called *tickets*. A ticket can be created by an object owner and be passed to a registered or unregistered user (or user group) to provide an action permit (e.g. read) on a data object or collection. A ticket provider can impose access restrictions, such as who can use the ticket, and the time period during which the ticket can be applied. The ticket is used subsequently for opening a connection to a storage server and allowing a specified operation to be completed.

2.4 Access Auditing

Each action on a data object can be logged and the action success or failure can be noted in an audit trail. The owner or anyone with control permission on the object can impose auditing of the actions on a data object from specific users. Then whenever those users access the dataset the action is logged in the audit trail. The users can also impose a restriction on the audit that prevents their actions from being audited in which case an anonymous user audit trail is written. The audit trail consists of five-tuples that include identifiers for the accessed object, the accessing user, the audited operation, a timestamp and a comment.

3 Evaluation Methodology

The SRB version that we evaluate stores all the system and user metadata onto a relational database, called Metadata Catalog (MCAT), and uses SQL statements for handling all the metadata search and update operations involved in data accesses. In fact, three out of the four components that we identified previously, namely searching, access control and auditing are fully implemented using select and insert/update SQL queries. Only the data storage management is implemented using standard operating system calls outside the database. All the metadata traffic has to pass through a specific SRB server that communicates directly with the MCAT, called MCAT- enabled server.

3.1 Goals and Metrics

We use system performance and resource utilization metrics in order to explore potential limits in the system scalability and identify software and hardware bottleneck resources. As a result of our measurements, we point out that some particular functions can overload the system, while others have negligible system performance implications.

The *command throughput* measures the number of SRB commands the system completes in a time unit. Since most of the relevant commands are operating on data files, we normalize this metric across the different commands by measuring the number of operations per individual file completed in the time unit (files/s). Instead, the *data throughput* measures the amount of bytes transferred into and out of the system in a time unit (bytes/s). The data throughput cannot exceed the aggregate network link capacity connecting the clients to the SRB storage servers. In the case of data uploading operations, the data throughput is also limited by the aggregate capacity of the disk channels connecting the storage devices to the SRB storage servers. On the other hand, download data transfer throughput can exceed the disk channel bandwidth limitation, if the requested data are served from the buffer cache rather than directly from the disks.

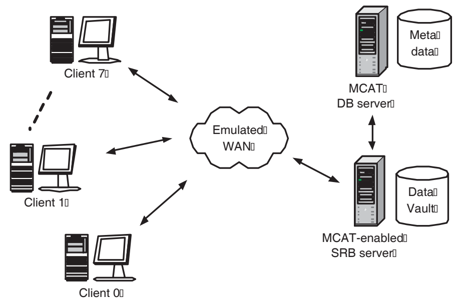

Fig. 2. The testbed configuration of SRB that we used during our experiments.

The command throughput tracks the metadata management efficiency of the system, while the data throughput is more dependent on the underlying networking and storage devices used for actually serving the data transfers. Depending on the size of the datasets transferred and the granularity of the individual requests, either the command throughput or the data throughput can become the prevalent performance metric that specifies the number of operations completed per second.

Additionally, we use the *turnaround time* metric to measure the elapsed time from the point a user submits a command until the command completes. Since each command can handle different numbers of files, we normalize the turnaround time by measuring it in time units to complete an operation per individual file (s/file). During each experiment, we also accumulate statistics of the processor utilization on the hosts running the SRB server and the database server, respectively, and also the average amount of bytes passing through the channel of each disk and the network interface card of each host.

3.2 Experimentation Environment

In all our experiments, we used SRB version 2.1.1 with MCAT version 2.0 running on an Oracle 9.2.0 database. The server systems that we used were Intel-based workstations running Debian Linux kernel 2.4.21. We ran the Oracle database on a PIII/ 1.4GHz system with 2GB memory, 1 Gbit/s Ethernet network card, and Ultra160 SCSI disk adapters. We stored the database datafiles on a Seagate Cheetah 10K RPM disk of 50 MB/s minimum internal transfer rate. The MCAT- enabled SRB server was hosted on a system with PIII/ 866MHz equipped with 1GB memory, 1 Gbit/s Ethernet network card, Ultra160 SCSI adapter and Seagate Cheetah disk of 10K RPM and 26 MByte/s minimum internal transfer rate. The majority of our experiments have the clients accessing the MCAT- enabled SRB directly (Figure 2). Nevertheless, an additional SRB server has been used in some experiments for measuring the impact of multiple SRB servers between the client and the metadata database server. The request workload was generated by up to eight client machines with Intel PIII/ 1GHz running Debian Linux 2.4.20 and using 256 MB and 100 Mbit/s network cards. We relied on the NIST Net tool to emulate alternative network throughput and latency values between the system nodes [11] (Emulated WAN in Figure 2).

4 Experimental Results

We start our evaluation by focusing on specific SRB commands that we expect to be frequently used: i) `Sput` for uploading a fileset to the server, ii) `Sget` for downloading a fileset from the server, iii) `Sbload` for uploading in bulk a fileset, iv) `Sbunload` for downloading in bulk a fileset to a local directory, v) `Sregister` for registering a fileset to the server without physically copying the data to SRB, and vi) `Sls` for listing the contents of an SRB directory. Since the performance of `Sput`, `Sget` and `Sregister` is similar, assuming that the data transfer throughput is not the bottleneck, in several cases we only evaluate the `Sget` command instead of all three of them. `Sbload` and `Sbunload` operate by default on containers and use multiple threads to transfer data in large blocks of 8 MByte each. This improves significantly the system efficiency especially when handling large numbers of small files.

In our experiments, each reported number is average over the number of runs (up to 10) that were necessary to achieve half-length of the 95% confidence interval of the turnaround time less than 5% of the average turnaround time. Each run lasts three minutes, and we discard the statistics gathered during the first minute. Except for Section 4.5 where we examine alternative security schemes, we only use password-based authentication without encryption in the rest of our experiments.

4.1 Baseline Performance

Our goal here is to evaluate the turnaround time across different operations as a function of the system throughput. The design of each command allows them to scale up to a number of operations per second with reasonable turnaround time. Trying to further increase the throughput results in arbitrarily high turnaround time due to saturation of some particular hardware or software system resource. We assume a single SRB storage server with a collection of 1000 empty files. We show later in detail how sensitive the system is to the numbers of files it contains.

4.1.1 File Accesses

In the first set of experiments (Figure 3(a)) we evaluate the performance of `Sput`, `Sget`, and `Sregister` when used for individual files rather than file sets. We measure the time that is required to complete a single file operation across different values of system throughput. We only use files of size 1 KByte for this experiment so that we can isolate the metadata management performance of the system. We observe that `Sregister` maintains less than 300ms turnaround time for throughput up to 9 files/s, while `Sget` and `Sput` increase exponentially the turnaround time when the system throughput exceeds about 8 files/s. We should keep in mind that `Sregister` involves no file data transfer but does update the metadata database, `Sget` mainly only reads data and metadata, and `Sput` both transfers data and updates metadata.

In Figure 3(b), we repeat the above experiment using `Sget` and `Sput` over entire directories rather than individual files. We observe that as the number of files increases, turnaround time tends to drop but starts to exponentially increase again as throughput exceeds 8 files/s. In comparison to Figure 3(a), `Sput` and `Sget` manage to maintain significatly lower turnaround time, by aggregating the update overhead over more than one files.

In Figure 3(c), we focus on the `Sbload` and `Sbunload` commands. Both these operations leverage the container mechanism in order to handle a large number of files efficiently. We notice that the throughput achieved by `Sbload` is an order of magnitude lower than that of `Sbunload`. We applied the `Sbunload` repeatedly on the same container, and we measure the best possible performance assuming that file

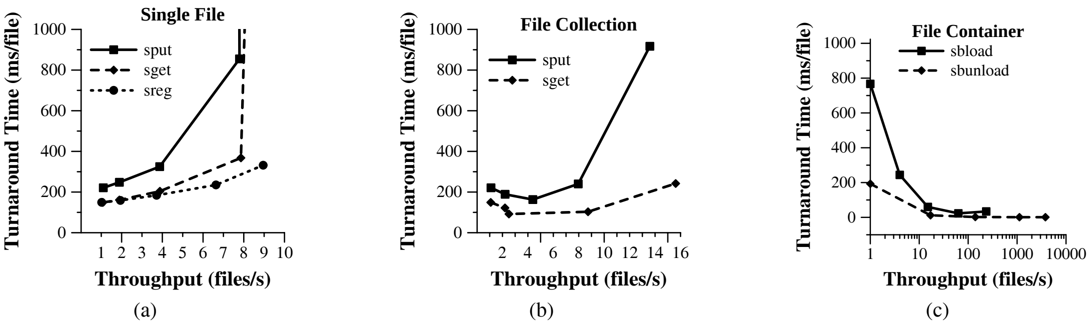

Fig. 3. (a). We measure the turnaround time of `Sput`, `Sget`, and `Sregister` as a function of the achieved throughput. Each command invocation transfers or registers a single file of 1 KByte. We notice that the turnaround time of `Sput` and `Sget` increases exponentially as the throughput grows, unlike `Sregister` that reacts more gracefully to higher loads. (b). When `Sput` and `Sget` manipulate collections of multiple files rather than individual files, we observe a noticable drop in the turnaround time. (c). We explore the maximum throughput and the corresponding turnaround time of `Sbload` and `Sbunload`. The independent variable is the number of files contained in the transferred container. `Sbunload` can handle an order of magnitude more files in each invocation than `Sbload`, while requiring lower time to complete for the same throughput.

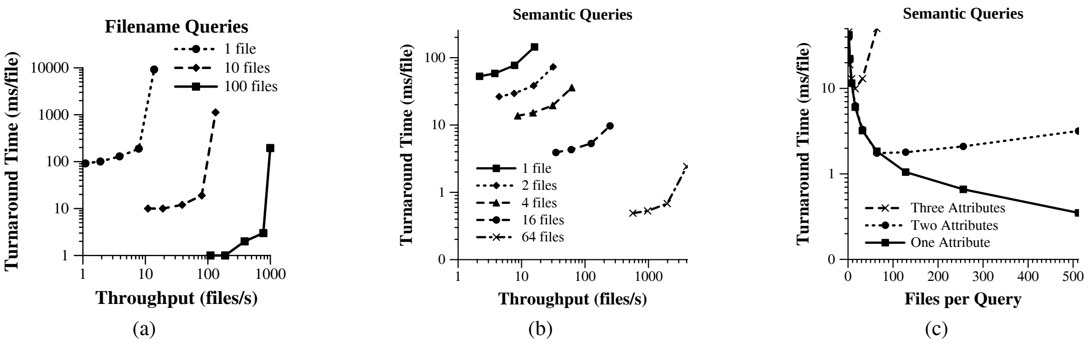

Fig. 4. (a) We consider filename queries, and vary the number of files returned by `Sls` in a collection of 1,000 empty files. We notice that as the number of returned files increases, the turnaround time per file drops exponentially, while the system consistenly saturates at roughly ten queries per second. (b) We study the performance of semantic queries based on user-defined attributes. As the number of files returned by a query increases, the completion time per file drops, while higher throughput makes the reported turnaround time to increase exponentially. (c) We focus on semantic queries, and vary between one and three the number of metadata attributes involved in a query. We notice that, with two and three attributes, the search time per file increases exponentially, when the number of returned files exceeds some threshold. During this experiment, we maintained a very low average request rate of 0.1 queries/s.

data are made available from SRB buffer cache, and metadata from the database buffer and library cache. More importantly, we observe that the completion time per file drops exponentially as the throughput increases, which demonstrates the scalability properties of these two particular commands. We should point out that the throughput can only be controlled indirectly by the rate at which requests are submitted to the system. By further increasing the request rate, we didn't notice additional improvement in the measured performance without negative effect to the reliable completion of the commands.

4.1.2 Metadata Search

In the implementation of SRB, system and user metadata accesses are handled by running SQL queries against relational database tables. In order to evaluate the related performance, we first experiment with the `Sls` command. It returns the names (and optionally the system attributes) of the files that satisfy a search pattern of plain and wildcard characters. We used a small collection of 1000 files, and ran `Sls` commands that would return a random subset of 1, 10, or 100 filenames. We measure the time that it takes per file to complete the operation across different achieved system throughput values. As before, the system throughput is controlled by changing the rate at which operations are submitted to the system.

As we see in Figure 4(a), the number of files returned by each `Sls` invocation affects significantly the turnaround time per file. This time is less than 1ms per file when `Sls` requests 100 files, about 10ms per file for requests of 10 files, and about 100ms per file for requests of 1 file. Furthermore, as the system throughput increases, the turnaround time grows quickly to more than 100ms, 1s and 10s, respectively. We conclude that the more files an `Sls` call returns, the better the performance at higher system throughput. This is reasonable given the significant database overhead incurred by the relational table searches, which is demonstrated by the increased processor utilization on the database server.

Subsequently, we attach ten attribute-value pairs, $a_0 = v_0, a_1 = v_1, ..., a_9 = v_9$, to each file in a collection of 1024 files. Uniformly across the different files, the first attribute takes values between 1 and 1024, the second between 1 and 512, and so on, while the last attribute only takes values 0 and 1. We use the `Sufmeta` command of SRB to search the files with a specific attribute value. In queries of one attribute, we search for files that satisfy the constraint $p_i = v_i$, with $i = 0, .., 9$. Note that the query $p_i = v_i$ returns 2^i files. For example, $p_0 = v_0$ returns one file, while $p_9 = v_9$ returns 512 files. In queries of two attributes, we use the conjunction $p_{i+1} = v_{i+1}\ AND\ p_i = v_{i+1}$. The equality $p_{i+1} = v_{i+1}$ finds 2^{i+1} files which are subsequently filtered into 2^i files by the equality $p_i = v_{i+1}$. This structure can be generalized to three or more attributes in a similar way. Essentially, as the number of attributes in our query increases, an exponentially larger number of files is filtered down to a specified return size. This allows us to examine whether the search time depends on the return size only, or other parameters as well.

In Figure 4(b), we measure the turnaround time per returned file spent using single-attribute queries. As the query throughput increases, the turnaround time grows exponentially. Also, queries that return more files require less time per file to complete. This behavior is consistent with filename queries that we examined in Figure 4(a). It is also something we expected since both types of queries are handled through SQL statements running on the MCAT. In Figure 4(c), we consider queries of one and multiple attributes, while we vary the number of returned files between 1 and 512. We observe that in queries of one attribute, the turnaround time to search each file drops, as the number of returned files for the query increases. Instead, we notice that in queries of two and three attributes, as the number of returned files exceeds 64 and 16, respectively, the search time per file increases exponentially. Correspondingly, the processor utilization (not shown) of the MCAT database grows from less than 1% to about 20%. We conclude that multi- attribute queries are more likely to overload the system than single-attribute queries, while file queries with user-defined attributes behave similarly to searches using system metadata.

In summary, the turnaround time of search operations depends on both the number of files found during the intermediate steps of the query processing and the number of files returned to the user.

4.2 Bottleneck Analysis

In Figure 5, we depict the total bidirectional transfer rates measured on the disk channel and network link of the SRB storage server. We transfer files of three different sizes: (i) 1 KByte, (ii) 1 MByte and, (iii) 10 MByte. `Sbload` with 1 MByte or 10 MByte files is disk- I/O intensive and its throughput (shown on the top of the network link bar) is limited by the disk transfer capacity measured at about 22 MByte/sec. For a disk with nominal minimum internal transfer rate of 26 MByte/s this is a reasonable expectation. The network link transfer rate is only slightly higher at 23 MByte/s. In Figure 6 we also notice the `Sbload` and `Sbunload` of 1 KByte files to stress out the database server processor. When the file size increases to 1 MByte or 10 MBytes, however, the database processor utilization drops with a corresponding increase in the SRB processor utilization. We attribute this change to the processing overhead involved in handling concurrent disk accesses and network transfers.

Similarly, `Sget` of 1 KByte and 1 MByte files is mostly limited by the processing power of the database server, while the corresponding throughput hardly exceeds 15 files/s. When downloading files of 10 MByte, however, the system throughput is limited by the SRB server processing power. From our previous experience, we already know that even though the 1 Gbit/s Ethernet links have nominal capacity of $1.25\ 10^6$ Byte/s, their measured transferred hardly exceeds 50 MByte/s when using the typical Ethernet frame size of 1500 bytes, due to the network protocol processing required. In our previous discussion we assume that the downloaded directory is fully cached in the SRB server main memory, and practically no disk transfers are involved.

`Sbunload` of 1 MByte or 10 MByte files (cached in the SRB server) stresses out both the processor and the network links of the SRB server. Instead, `Sbload` of 1 KByte files incurs significant database and SRB server processing load at relatively low network data throughput. Finally, we experiment with `Sls` across 1, 10 and 100 returned files. We observe that `Sls` saturates completely the SRB server processor, while incurring high database processing overhead

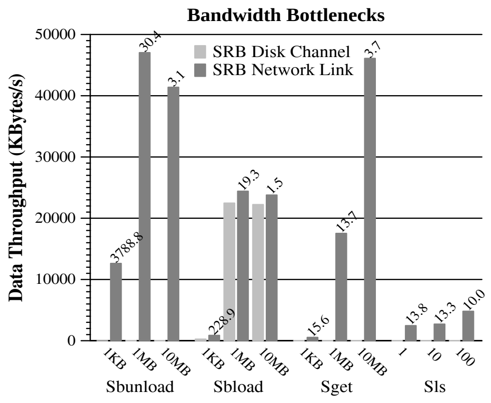

Fig. 5. We depict the measured data throughput at different file sizes (`Sbload`, `Sbunload` and `Sget`) or numbers of returned files (`Sls`). We show that when manipulating large files, `Sbload` saturates the disk channel capacity, while `Sbunload` and `Sget` are stressing the network transfer capacity. The number at the top of the bars shows the measured command throughput (files/s).

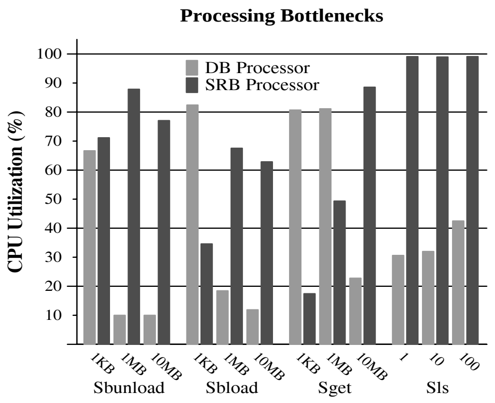

Fig. 6. We measure the processor utilization of the SRB server and the database server when varying the file sizes transferred by `Sbload`, `Sbunload` and `Sget` or the number of files returned by `Sls`. We notice that high command throughput involves high database and SRB server processing overhead, while high data throughput saturates the SRB server processor.

too. Both the network and disk data throughput are negligible, though.

From the above we conclude that high command throughput operations are limited by the processing power of either the database server, or the SRB server. On the other hand, high data throughput operations are limited by the processing power of the SRB server and the corresponding network transfer capacity. In general, we expect the SRB processing utilization at high data throughput to vary depending on how much the I/O device controller depends on the host processor during the data transfers.

4.3 Namespace Organization

The number of distinct files managed efficiently by SRB is one measure of scalability that we used. We expect that as the system is more widely used, the number of uploaded datasets will increase. We examine the effect that such increase has to the achieved throughput and the user-perceived turnaround time. We experimented with three different collection configurations: (i) a flat collection of 1000 empty files (1Kx1), (ii) a flat collection of 100,000 empty files (100Kx1), and (iii) a single collection of 10,000 collections each containing 100 empty files (10Kx100). We generated the first two organizations using a separate `Sput` for each file, and the last organization using `Sbload` for each collection of 100 files. In all our experiments, the initial file configuration remains unaffected by the files additionally inserted for the evaluation of particular commands. We checkpointed and restored the database after the creation of each organization, in order to prevent performance degradation from tablespace fragmentation. During our experiments, we also verified 100% library and buffer cache hit ratio in the database server, as a result of aggressive buffer caching and query cursor caching.

In Figure 7 we show the turnaround time (per file) across the three collection organizations. We notice a significant slowdown of `Sget` when the collection size increases from 1K files to 100K files. From Figure 8 we find a strong correlation between the turnaround time and the database processor utilization. Ideally, we would expect the system performance to be independent of the number of files managed, especially when only a small subset of them is accessed at a time during the system operation. Given the significant database latching and table scanning activity reported by the Oracle diagnostics tools, we infer that the system would most likely benefit from additional indexing over the relational tables. It is surprising that the turnaround time drops with the 10Kx100 organization, even though we would expect the opposite due to the larger total number of files in comparison to the other two cases. We don't have a definite explanation for that behavior, but we could attribute it to the cost optimization decisions of the database engine itself.

In Figure 8 we also observe high database processing load with `Sbunload` in the 10Kx100 organization. In fact, we had to reduce the rate of the requests from 1 req/s to 0.2 req/s in order to get the experiment completed. As a result, the measured command throughput (shown at the top of the DB bars) is an order of magnitude lower in comparison to the other two organizations. Nevertheless, we hardly notice any major increase to the turnaround time (per file) due to the amortization of the cost over a relatively large number

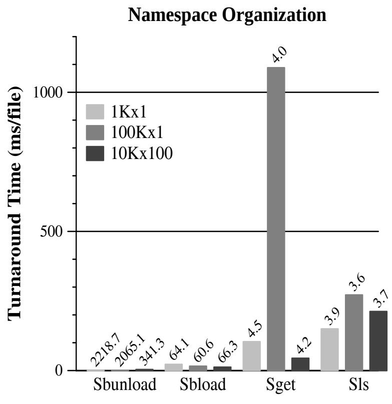

Fig. 7. We illustrate the turnaround time (per file) across three different namespace organization schemes. `Sget` is slowed down in the case of a single collection with 100,000 files. The measured command throughput is shown at the top of the bars.

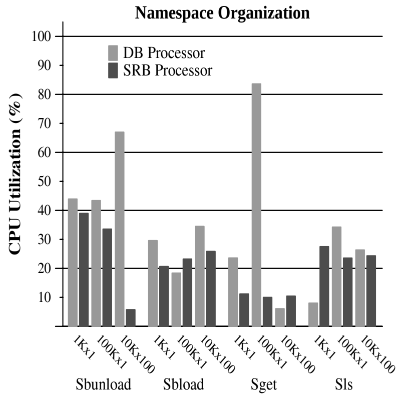

Fig. 8. We examine the SRB and DB processor utilization across different namespace organizations. We find an increased number of files to be leading into higher database processing activity in the case of `Sbunload` and `Sget`.

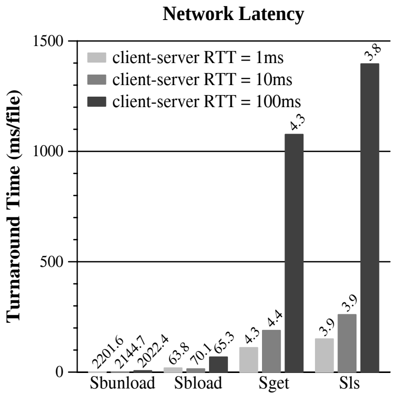

Fig. 9. We experimented with different network latencies between an SRB client and server. We see that `Sls` and `Sget` are both very sensitive to increased latencies, as a result of which they are significantly slowed down.

of files. Finally, we find `Sbload` and `Sls` insensitive to the collection organization, which implies that the necessary table indexes have already been added to the system to make the related metadata accesses fast.

Overall, we find that some particular operations remain sensitive to the namespace organization of the system. This occurs despite the fact that indexing structures have been added to prevent full table scans in several cases. Since table scans are currently not completely eliminated in the system, we anticipate that additional indexing is likely to reduce the observed excessive database processing load.

4.4 Network Latency

In Figure 9, we evaluate the effect of the network latency between the SRB client and the SRB server to the user-perceived completion time when running `Sget`, `Sbload`, `Sbunload` and `Sls`. Even though we only show numbers for throughput at about 1/4 of the maximum possible rate in our system setup, we verified that the relative effects that we report become less prevalent as the throughput increases.

`Sls` proves to be an operation very sensitive to the round-trip delay between the client and the server. We observe that for `Sls` returning just one file, the time to complete grows by an order of magnitude from roughly 150ms to 1.396s as the network latency changes from 1ms to 100ms. `Sget` behaves similarly since its completion time increases from about 111ms to 1.077s under the same conditions. We expect that multiple handshake steps in the protocol implementation of `Sget` and `Sls` to have a multiplier effect on the measured performance as a function of the round-trip delay. Nevertheless, it is remarkable, how the bulk operations `Sbload` and `Sbunload` remain relatively insensitive to the network latency. Additionally, from the measured throughput attached at the top of each bar in Figure 9, we find the throughput to remain relatively insensitive to the network latency at low operation request rates.

Finally, we examined the case that the client is attached to a local SRB storage server that is remotely connected to the MCAT- enabled SRB server. The accessed storage vault was directly attached to the remote SRB server. Repeating the experiments in that setting we didn't find any major difference in comparison to the case that the client is directly attached to the remote MCAT- enabled SRB server.

4.5 Access Control

As we already mentioned in Section 2.3, SRB supports a range of alternative methods for authenticating the users and authorizing their requests. We evaluated three particular authentication methods of SRB: (i) plain-text password which is expected to have the lowest overhead, and (ii) tickets that allow registered or unregistered users to access specific files, and (iii) GSI which enables SRB to participate in large grid installations. We compare the performance of all three methods with the SSH/SCP file copying facility which uses public-private key pairs for verifying the identify of a user. In all the measurements, we use the `Sget` command for SRB, and the `scp` command of OpenSSH version 3.6.1 on Debian Linux for SSH/SCP.

The workload that we used is based on the logged activity of file requests recorded by the FTP server operated by

Trace	Count	Sum(B)	Avg(B)	Max(B)	StdDev(B)	Sum(s)	Avg(s)	Max(s)	StdDev(s)
I	1,689	202,552,865	119,924	31,742,157	887,870	4,944	2.971	398	11.563
II	470	791,038,275	1,683,060	728,334,336	33,592,658	3,558	7.570	2,191	101.383

Table 1. We compare alternative authentication and file transfer schemes using two traces of duration 10 minutes each. The traces have been kindly made available by the administration of the FTP server operated by the Washington University at St. Louis. Trace I was taken on August 18, 2002 at time 20:50-20:59, while Trace II was logged on September 27, 2002 at time 23:50-23:59. In the above table we summarize the file size and transfer time characteristics as recorded in the log file. It is noteable, that Trace I consists of more file transfers, lower average file size, and lower download time than Trace II.

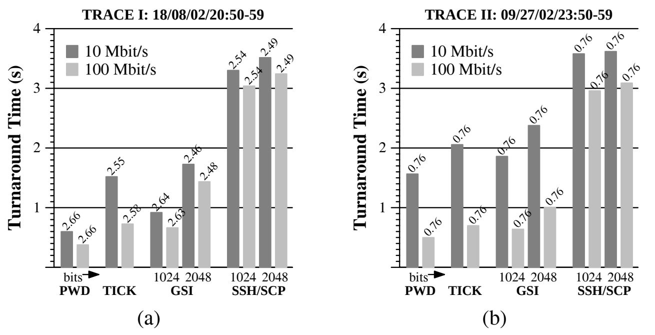

Fig. 10. (a). Password-based SRB is roughly twice as fast as GSI-based SRB. Both are significantly faster than `scp`. Ticket-based SRB is comparable to GSI-based SRB. The bit length of the RSA keys affects more the cost of GSI rather than that of `scp`. (b). When the workload involves few large files, we observe that the completion time of SRB transfers remains lower than `scp`. The bandwidth of the server network link has major impact to the transfer times. The measured throughput is shown at the top of each bar.

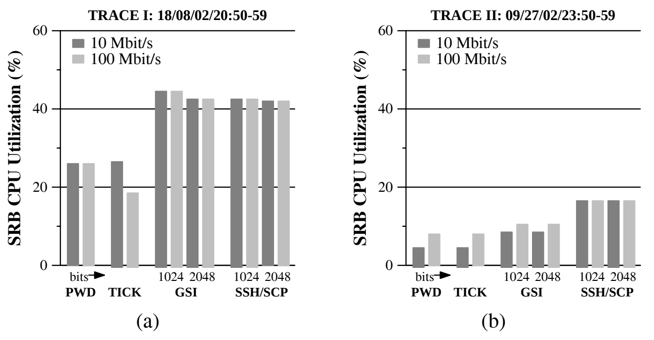

Fig. 11. (a). We notice that the authentication scheme can increase the processor utilization by as high as a factor of two. The cost of `scp` is similar to that of GSI though, while the cost of ticket accesses is comparable to that of password-based authentication. (b) `scp` almost doubles the processing utilization in comparison to GSI, but both the schemes are more costly than tickets and passwords. The smaller number of files transferred leads to lower processing utilization overall.

the Washington University at St Louis. For our experiments we used two 10-minute traces: the Trace I recorded on August 18, 2002 from 20:50 to 20:59, and the Trace II recorded on September 27, 2002 from 23:50 to 23:59. We summarize the features of these traces on Table 1. Trace I demonstrated relatively high activity with 1,689 requests of average size 117 KBytes, and average download time 2.971 seconds. Instead, Trace II consists of 470 requests with average size 1.6 MByte, and average download time 7.570 seconds. In order to emulate the workloads, we uploaded to SRB zero-filled files of the size specified in the traces. For practical reasons, we didn't try to recreate the transfer rates recorded for each file request. Instead, we replayed the traces using two different server network link capacities, 10 Mbit/s and 100 Mbit/s. The node hosting the client was connected to the SRB server node through a gigabit Ethernet switch.

In the case of GSI, we tried RSA keys of both 1024 and 2048 bits. We also generated RSA keys of 1024 and 2048 bits for SSH/SCP and used empty passphrase to run the transfers in batch mode. In Figure 10(a), we notice that Trace I with password authentication takes average file transfer time about 598 msec for 10 Mbit/s, and 377 msec for the 100 Mbit/s server network link. The network capacity doesn't make major difference due to the small average size of the transferred files. Use of GSI certificates with 1024 bits increases the file transfer time to 918 msec and 663 msec, respectively. We believe that this a reasonable cost for the additional security that GSI offers, namely authentication of both the user and the server, and local-only use of the private keys. Increasing the number of bits to 2048 almost doubles the transfer time. We also observe a drop in the achieved throughput shown at the top of each bar. Ticket-based authentication provides to SRB users the flexibility to grant access permission for their data to other users. The corresponding file transfer time lies between those of password and GSI-based authentication.

In the case of SSH we use the inexpensive blowfish cipher for the data transfer encryption. Note that the data traffic in the SRB transfer times that we reported previously was not encrypted at all. From Figure 10(a) we found `scp` to take more than 3 seconds on average. Changing the network link capacity and the number of bits in the RSA keys didn't make any significant difference. Using the -v option with `scp` we verified that most of the time is spent during the handshaking steps for establishing trust between the server and the client, rather than the data transfer itself. From Figure 11(a) we notice that the processor utilization in the SRB server is comparable between GSI and SSH, and roughly twice as high as the plain password authentication.

We repeated the above experiments using the Trace II that has fewer data transfers but larger average file size. Not surprisingly, the bandwidth of the server network link affects significantly the transfer time, especially with SRB transfers (Figure 10(b)). Consistently with Trace I, however, the `scp` requests take more time on average to complete. The difference exceeds a factor of three between GSI and `scp` for 1024 keys. From Figure 11(b) we also find that the processor utilization is much lower in comparison to 11(a), which can be explained by the lower authentication overhead as a result of the fewer file transfers in the case of Trace II.

In summary, we found that GSI and Ticket file transfers using SRB take more time to complete than password-based transfers but significantly less than file transfers using the SSH/SCP facility. The processor utilization on the server is affected more by the number of file requests and less by the actual amount of bytes being transferred.

4.6 Access Auditing

We also experimented with the performance of several SRB commands after enabling the auditing facility at different levels (namely 1, 2, and 3). However, we found no performance degradation as a result of auditing in any of the commands that we tried. Auditing involves appending of log information to a number of tables, but requires minimal search and modification of the stored metadata. Thus, it is not surprising that the incurred overhead is insignificant.

4.7 File System Interface

The Modified Andrew Benchmark (MAB) is a file system benchmark introduced fifteen years ago to emulate software development workloads. It has been widely used to evaluate and compare the performance of several distributed file systems over a multitude of hardware configurations. MAB consists of five phases: (i) create a directory hierarchy, (ii) copy files into the directories, (iii) traverse the hierarchy and list file attributes, (iv) scan the files, and (v) compile the files. Admittedly, these tasks only represent a specific way of using a file system, the one where files are accessed by a single thread of control at a fine granularity of individual data blocks. Therefore, by including MAB at the present study, we only expect to demonstrate the performance of SRB in this mode of operation, and allows us to compare SRB with other systems in this context.

In order to run MAB over SRB, we use an extension of the freeware Linux Userland Filesystem (LUFS) that allows an SRB-managed data collection to appear to a remote client as a mounted filesystem. The LUFS combines a kernel module with a user-space daemon into an extensible file system framework. For our experiments, we used a three-tier system organization where a client, the SRB server, and the database server run on three different machines. They are all interconnected through switched ethernet with average round-trip latencies about 150 microseconds. We introduce no artificial delay between any of the three machines. The database datafiles and the SRB data vaults are stored on disks directly attached to the corresponding hosts. For comparison, we also export the data vault filesystem of the SRB server and mount it over the network to the client through NFS v3.

From table 2 we observe that LUFS-SRB is rougly two orders of magnitude slower than NFSv3 in handling the MAB traffic. In fact, LUFS-SRB is especially slow in handling fine-granularity data accesses (phases ii,iv and v), and more acceptable with tasks involving metadata management only (phase i and iii). We also found indexing to be contributing critically towards improving the performance of the system. We examined the sensitivity of SRB to the number of files it manages by repeating the measurements after adding a thousand empty files. We found this modification to increase the MAB duration by roughly a factor of two, and was accompanied by extra processor utilization in the database server. This outcome clearly indicates that the potential advantages from tuning the database have not been fully tapped in fine-granularity data accesses.

We repeated the execution of the Modified Andrew Benchmark while logging the database operations using the SRB debugging facility. In Table 3, we break down the database operations categorized into five groups. *Select Query* corresponds to running SQL queries that retrieve metadata, and *Modify Query* describes queries that insert or update metadata in the database. *Bind Row* and *Unbind Row* operations describe the allocation and deallocation of variables for holding database table rows, respectively. *Get Row* refers to transfer of rows between the database and the SRB server, while we report separately the database updates under *Commits*.

The duration of the MAB execution is slightly extended due to the logging overhead. We believe that this does not affect the relevance of our conclusions given that we only use the measurements to compare the relative significance of the different database access components. The most important operation in terms of time overhead is the execution of SQL select statements. It is closely followed by the transfer of rows between the database and the server. By enabling indexing of the frequently used tables, we reduce consid-

	NFS	LUFS-SRB			
Mode	**ver.3**	**Index**		**Scan**	
Files	**0/1K**	**0**	**1K**	**0**	**1K**
Phase I	0.035	1.097	1.108	2.497	4.960
Phase II	0.317	53.495	95.546	137.536	220.569
Phase III	0.556	2.336	2.351	8.053	16.286
Phase IV	0.595	8.434	19.190	20.365	37.830
Phase V	1.783	66.339	142.342	180.642	306.856
Total	3.284	131.701	260.537	349.094	586.501

Table 2. We measure the duration (seconds) of the five phases of the Modified Andrew Benchmark over two different file systems, NFSv3 and LUFS-SRB. In the LUFS-SRB case, we (i) consider relational tables that are either fully scanned (scan), or indexed with B-trees (index), and (ii) examine two different configurations with zero (0) and a thousand (1K) extra empty files stored on SRB (in addition to those of the MAB itself). We observe that MAB runs two orders of magnitude faster over NFSv3 than over LUFS-SRB.

Operation	**Index**		**Scan**	
Description	Delay	Count	Delay	Count
Select Query	39.713	19536	41.385	19858
Modify Query	4.093	1893	4.261	1890
Bind Row	9.746	19536	9.960	19858
Unbind Row	6.448	19536	6.966	19858
Get Row	42.458	28340	249.334	28860
Commit	8.316	4448	8.483	4446
DB Total	115.370	95174	324.147	96668
MAB Total	145.906		359.447	

Table 3. We show a breakdown (seconds) of the SRB operations when running the Modified Andrew Benchmark. In both full-scan and indexed table accesses, the DBMS overhead is dominated by select query executions, and transfer of rows between the SRB server and the database.

Select					
Table	**Index**	**Scan**	**Table**	**Index**	**Scan**
ad_repl	13880	14101	td_rsrc_2typ	2789	2789
cd_user	10911	11226	td_data_typ	2786	2786
td_data_grp	9645	9870	td_rsrc_class	2786	2786
td_domn	8309	8624	cd_rsrc	1620	1620
au_domn	8217	8532	ad_grp_accs	479	574
td_ds_accs	7959	8181	td_data_2grp	367	461
au_group	7939	8160	td_grp_accs	367	461
ad_accs	7735	7955	ar_repl	95	95
ar_physical	6302	6522	ad_collcont	92	92
adc_repl	6115	6335	ar_accs	92	92
td_container	4314	4314	td_collcont	92	92
au_owner_domn	3421	3641	td_rsrc_accs	92	92
cd_owner_user	3421	3641	td_user_typ	2	2
td_owner_domn	3421	3641	au_auth_key	1	1
td_locn	2789	2789			

Table 4. Total number of table access occurrences in SELECT SQL queries when running the Modified Andrew Benchmark. We consider both full scan and indexed table operations. SELECT SQL queries are typically used for access authorization and data location discovery.

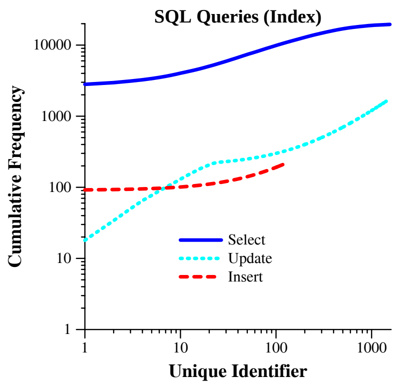

Fig. 12. We depict the cumulative frequency of select, insert, and update SQL statements during the MAB execution over an indexed database. We find a few SQL statements to be responsible for a considerable part of the select, update and insert query traffic, respectively. Note that the unique identifiers of the x-axis correspond to different statements across the three curves.

Insert			**Update**		
Table	**Index**	**Scan**	**Table**	**Index**	**Scan**
ad_repl	92	92	ad_repl	1620	1620
td_data_grp	20	20			
ad_accs	92	92			
ad_grp_accs	20	20			

Table 5. Number of table occurrences in INSERT and UPDATE SQL statements invoked when accessing or creating files.

erably the time spent on both these categories of database operations, while leaving almost unaffected the other three operation groups. As expected, the non-DB component of the MAB duration remains about the same across the full scan and the indexed execution.

The accessed dataset is less than 10 MByte large and fully fits into the system main memory, thus keeping the disk access delays to the minimum necessary to warm up the database and system cache. Given that the number of update operations is an order of magnitude smaller than the number of select queries, it is not unreasonable to observe a similar order of magnitude difference in the time that these different operations occupy. On the other hand, we get an indication that aggressive caching of the requested metadata on the client or even on the server outside the database could greatly benefit the measured system performance.

By looking closer at the individual SQL queries, we found that only 27 out of a total 83 tables within the database participate in most of the SELECT queries involved in the benchmark(Table 4). In addition, only four tables are updated, when new files are created or existing ones are modified (Table 5). Overall, we counted 1542 unique SQL statements to account for all the SELECT queries handled by the database during the benchmark. In our counting, each query with different SQL statement text qualifies as different from the others. The single most frequent SELECT statement accounts for 15% out of a total 19,585 queries that were recorded in the SRB log file. Also, the first 155 most frequent unique statements (about 10%) covered more than 50% of the database entire query traffic (Table 12). Therefore, only few tables are most frequently utilized by only a small number of queries that are used mostly often. This evidence substantiates the argument that aggresive query optimization and caching is likely to improve the performance of the system.

Due to implementation constraints, the LUFS-SRB does not take advantage of bulk transfer operations available in SRB. In addition, the Modified Andrew Benchmark itself does not incorporate all the facilities available in SRB. The fact that a small fraction of the metadata is heavily utilized by few repetitive SQL statements can be used for making the majority of the metadata accesses faster, and potentially significantly reduce the response time of the system.

5 Related Work

To the best of our knowledge, the present paper is the first comprehensive performance evaluation of the Storage Resource Broker. The architecture and the user interface of SRB are described in more detail by Baru et al. [4] and Rajasekar et al. [19]. The studies by Bell et al. [5] and Nallipogu et al. [18] focused on pipeline optimizations of SRB that improved the overlapping between network communication and disk accesses for I/O intensive file transfers. The Datacutter middleware that is currently part of SRB has been introduced by Beynon et al. to support handling of multidimensional rate queries on scientific datasets [8].

Stevens et al introduce the myGrid middleware framework for the structural and semantic description of data and services that make possible to build and execute distributed workflows [24]. Singh et al. describe the Metadata Catalog Service (MCS) to manage attributes for stored data [23]. Although similar to SRB from several aspects, MCS is designed to only handle logical metadata, thus factoring out information for physical locations of datasets, and functionality for access control. Allcock et al. introduce the GridFTP extension of the FTP protocol for high-speed transfers of files, and the Metadata Catalog/Replica Catalogs for managing descriptive and location information of datasets [2]. In comparison to SRB, GridFTP optimizes transfers of individual files rather than collections, while the Replica Catalog provides to users and applications direct control of data replication tasks. Also, SRB is a data discovery and sharing facility rather than a distributed application authoring environment which is target of the WebDAV network protocol [13].

Early peer-to-peer storage systems advocated the need to protect the anonymity of content publishers, subscribers, and maintainers, and provide decentralized organization of the software and hardware resources [12]. More recently, peer-to-peer systems became syonymous with any distributed system, where all the nodes have the same responsibilities, and communicate with each other symmetrically [21]. In that context, sophisticated routing substrates were introduced to provide the framework for automatically handling load balancing, reliability, data replication, and caching services [16, 25]. On the other hand, centralized search engines are currently available that handle queries for public web content over the entire Internet [10]. In contrast, data grids combine search functionality with storage and secure access of data over wide-area networks.

The Avaki Data Grid is proprietary technology that offers federated data services across multiple sites, but we only have limited knowledge about its internal architecture [3]. Its data grid directory provides direct access to the underlying data objects at their source locations. Authentication of users and groups can be done through directory services already existing at each individual location. File data and metadata are cached locally for improved access response time. Object search operations are supported based on attributes maintained by the underlying file system, or custom user attributes attached to individual objects.

Andrew (AFS) is a scalable distributed file system that allows data sharing over a wide-area network [15]. Files are organized into volumes that can be replicated across multiple machines forming volume storage groups. Copies of a file are cached locally on individual client machines; the local copies are invalidated before a file can be modified on a different machine. A common directory tree provides access to files in a location-transparent way. Despite its sophisticated functionality, Andrew leaves unsolved the problem of attribute-based file search, which is one of the main issues SRB tries to address.

Nest is a storage appliance that supports multiple data transfer protocols and concurrency control models [6]. Pangaia is a wide-area file system built on a symmetrically decentralized collection of commodity computers [22]. It replicates files and directories aggressively when they are accessed, which improves the system performance, reduces the network utilization, and leads to high availability.

Farsite is a distributed file system that provides file availability and reliability through replication, file content secrecy through cryptography, and integrity through fault- tolerant protocols [1]. It has been designed to perform efficiently through data caching, lazy update propagation, and authentication delegation. Instead, Ivy is a multi-user read/ write peer-to-peer file system consisting of a set of logs. The logs are stored on a distributed hash table [17]. Data retrievals and modifications are conducted by scanning and appending log records respectively, and applications access the system through a conventional file system interface.

Brandt et al. investigate the metadata management problem in the context of distributed storage systems [9]. They propose a two-level metadata server hierarchy and use hashing for balancing the access load across the different nodes. They maintain a hierarchical directory tree for preserving the traditional file access semantics. They describe a method to improve the access efficiency by delaying the propagation of metadata updates among the different servers. In an earlier study, Tyler and Fisher examine the design of a high-performance storage system based on commodity Encina OLTP technology [26]. Metadata distributed across multiple nodes can be maintained consistently by making use of nested transactional operations.

Xu et al. envision semantics-aware file servers that provide, among other features, advanced search capabilities, customized name space views, and task automation through dependency relations [29]. One of the challenges is managing semantic relations with lightweight database systems, assuming that the data protection properties of full database systems have high overhead and are unnecessary for the above application.

6 Conclusions

We briefly described the architecture of the Storage Resource Broker, and comprehensively examined several of its features through experimental performance evaluation. We found that commands for bulk file transfers (such as `Sbload` and `Sbunload`) incur high overhead for handling small numbers of files, but scale very well with increasing numbers of files. Their performance is limited by the processing power of the database, and the processing power and data transfer capacity of the SRB storage server. Commands for transferring individual files (`Sput` and `Sget`) can only handle a small number of files per second, and they are limited by the processing power and the data transfer capacity of the SRB storage server. File searches based on filenames and user-defined attributes (`Sls` and `Sufmeta`) are mainly limited by the processing power of the SRB storage servers, even though they also incur significant database processing load. In addition, the turnaround time of semantic search operations grows exponentially with the number of attributes involved in the query due to the database processing involved.

We found commands not optimized for bulk operations very sensitive to round-trip latencies typical in cross-country network connections. We also found the database processing activity to be affected by the organization of the file namespace. SRB file transfers with GSI authentication take more time to complete than transfers of lower security based on passwords, but are similar to file accesses based on tickets. All of them are significantly faster than the widely used SSH/SCP facility. Fine granularity data accesses of SRB through file system interfaces can also be supported. However, they are much slower than those of widely used distributed file systems, and are also sensitive to the file namespace organization. We found table searches to be the most common operation on the database, and few queries to be accountable for most of the database activity. Overall, we believe that the system performance is likely to improve by additional indexing, and reorganization of the relational table structure towards better efficiency, along with data and metadata caching locally on each client site.

Acknowledgements. We are thankful to Gregory McCarthy for his encouragement, Mark James and Jonathan Sacks for coordinating the multi-site collaboration involved, and Arcot Rajasekar and the rest of the SRB development team for providing the source code of SRB and helping us understand several of its features. This research has been supported by the Brain Morphometry Biomedical Informatics Research Network (BIRN, http://www.nbirn.net), funded under contract number 3M01RR00030 by the National Center for Research Resources (NCRR) at the National Institutes of Health (NIH).

References

1. Adya, A., Bolosky, W. J., Castro, M., Cermak, G., Chaiken, R., Douceur, J. R., Howell, J., Lorch, J. R., Theimer, M., and Wattenhofer, R. P. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In *USENIX Symposium on Operating Systems Design and Implementation* (Boston, MA, Dec. 2002), pp. 1–14.
2. Allcock, B., Bester, J., Bresnahan, J., Chervenak, A. L., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnal, D., and Tuecke, S. Data Management and Transfer in High Performance Computational Grid Environments. *Parallel Computing Journal* **28**, 5 (May 2002), 749–771.
3. Keep It Simple: Overcome Information Integration Challenges with Avaki Data Grid Software. Tech. rep., Avaki Corporation, July 2003.
4. Baru, C., Moore, R., Rajasekar, A., and Wan, M. The SDSC Storage Resource Broker. In *IBM CASCON* (Toronto, ON, Nov. 1998).
5. Bell, K., Chien, A., and Lauria, M. A High-Performance Cluster Storage Server. In *IEEE International Symposium High Performance Distributed Computing* (Edinburgh, Scotland, July 2002), pp. 311–320.
6. Bent, J., Venkataramani, V., Leroy, N., Roy, A., Stanley, J., Arpaci-Dusseau, A. C., Arpaci-Dusseau, R. H., and Livny, M. Flexibility, Manageability, and Performance in a Grid Storage Appliance. In *IEEE International Symposium on High Performance Distributed Computing* (Edinburgh, Scotland, July 2002), pp. 3–12.

7. Berners-Lee, Hendler, J., and Lassila, O. The Semantic Web. *Scientific American* **284**, 5 (May 2001), 34–43.
8. Beynon, M., Ferreira, R., Kurc, T. M., Sussman, A., and Saltz, J. H. DataCutter: Middleware for Filtering Very Large Scientific Datasets on Archival Storage Systems. In *IEEE Symposium on Mass Storage Systems* (College Park, MD, Mar. 2000), pp. 119–134.
9. Brandt, S. A., Miller, E. L., Long, D. D., and Xue, L. Efficient Metadata Management in Large Distributed Storage Systems. In *IEEE/NASA Goddard Conference on Mass Storage Systems and Technologies* (San Diego, CA, Apr. 2003), pp. 290–298.
10. Brin, S., and Page, L. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems* **30**, 1–7 (Apr. 1998), 107–117.
11. Carson, M., and Santay, D. NIST Net - A Linux-based Network Emulation Tool. *ACM Computer Communication Review*. (to appear).
12. Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability* (Berkeley, CA, July 2000), pp. 311–320.
13. Fielding, R. T., Jr., E. J. W., Anderson, K. M., Bolcer, G. A., Oreizy, P., and Taylor, R. N. Web-Based Development of Complex Information Products. *Communications of the ACM* **41**, 8 (Aug. 1998), 84–92.
14. Foster, I., Kesselman, C., Tsudik, G., and Tuecke, S. A Security Architecture for Computational Grids. In *ACM Conference on Computer and Communication Security* (San Francisco, CA, Nov. 1998), pp. 83–92.
15. Howard, J. H., Kazar, M. L., Menees, S. G., Nichols, D. A., Satyanarayanan, M., Sidebotham, R. N., and West, M. J. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems* **6**, 1 (Feb. 1988), 51–81.
16. Kubiatowicz, J., Bindel, D., Chen, Y., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. OceanStore: An Architecture for Global-scale Persistent Storage. In *ACM Symposium on Architectural Support for Programming Languages and Operating Systems* (Cambridge, MA, Nov. 2000), pp. 190–201.
17. Muthitacharoen, A., Morris, R., Gil, T. M., and Chen, B. Ivy: A Read/Write Peer-to-Peer File System. In *USENIX Symposium on Operating Systems Design and Implementation* (Boston, MA, Dec. 2002), pp. 31–44.
18. Nallipogu, E., Ozguner, F., and Lauria, M. Improving the Throughput of Remote Storage Access through Pipelining. In *International Workshop on Grid Computing* (Baltimore, MD, Nov. 2002), pp. 305–316.
19. Rajasekar, A., Wan, M., and Moore, R. MySRB & SRB: Components of a Data Grid. In *IEEE International Symp High Performance Distributed Computing* (Edinburgh, Scotland, July 2002), pp. 301–310.
20. Rajasekar, A., Wan, M., Moore, R., Jagatheesan, A., and Kremenek, G. Real-life Experiences with Data Grids: Case Studies using the SRB. In *International Conference on High Performance Computing - HPCAsia* (Bangalore, India, Dec. 2002).
21. Rowstron, A., and Druschel, P. Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility. In *ACM Symposium on Operating Systems Principles* (Banff, Alberta, Oct. 2001), pp. 188–201.
22. Saito, Y., Karamanolis, C., Karlsson, M., and Mahalingham, M. Taming Aggressive Replication in the Pangaia Wide-Area File System. In *USENIX Symposium on Operating Systems Design and Implementation* (Boston, MA, Dec. 2002), pp. 15–30.
23. Singh, G., Bharathi, S., Chervenak, A., Deelman, E., Kesselman, C., Manohar, M., Patil, S., and Pearlman, L. A Metadata Catalog Service for Data Intensive Applications. In *ACM Supercomputing Conference* (Phoenix, AZ, Nov. 2003).
24. Stevens, R., Robinson, A., , and Goble, C. myGrid: Personalised Bioinformatics on the Information Grid. *Bioinformatics* **19**, 1 (2003), 302–304.
25. Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *ACM SIGCOMM Conference* (San Diego, CA, August 2001), pp. 149–160.
26. Tyler, T., and Fisher, D. Using distributed OLTP technology in a high performance storage system. In *IEEE Symposium on Mass Storage Systems* (Monterey, CA, Sept. 1995), pp. 45–45.
27. Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L., and Tuecke, S. Security for Grid Services. In *International Symposium High Performance Distributed Computing* (Seattle, WA, June 2003), pp. 48–57.
28. Wilkes, J. Data Services - from data to containers, Mar. 2003. Keynote at the USENIX Conference for File and Storage Technologies, San Francisco, CA.
29. Xu, Z., Karlsson, M., Tang, C., and Karamanolis, C. Towards a Semantic-Aware File Store. In *Workshop on Hot Topics in Operating Systems* (Lihue, HI, May 2003), pp. 145–150.