A Large-Scale Digital Library System to Integrate Heterogeneous Data of Distributed Databases

Mariella Di Giacomo, Mark Martinez, and Jeff Scott
Los Alamos National Laboratory, Los Alamos, NM 87545, USA,
{mariella,mlbm,jscott}@lanl.gov

Abstract. The Web has become the primary means for information dissemination of all kinds; our interest is in dissemination of scientific information from on-line digital libraries. We have designed a Web application, called *SearchPlus*, based on a distributed, scalable, fault-tolerant, and secure architecture, to allow access to tens of millions of scientific bibliographic records and their citations, integrating information from multiple heterogeneous data sources, and making this information available for querying and analysis. A full-scale test-bed environment has been developed to assess hardware and software configuration and performance. This paper gives the motivations for building such a system, describes the architecture of our distributed database system, and highlights performance analyses and subsequent improvements.

1 Motivation

The Los Alamos National Laboratory (LANL) Research Library (RL) [2] focuses on digital information services. It provides commercially available scientific data, through Web applications, to LANL scientists as well as several external institutions. One such application is *SearchPlus*. The primary objective of *SearchPlus* is the construction of a comprehensive distributed database of scientific journal articles—now over 55 millions—and citation information in a common format—now over 500 million entries, and providing access to this information to RL customers. Scientists now rely on these resources to meet deadlines, write articles, and vie for funds in highly competitive fields. Such a critical research resource must exhibit as little service disruption as possible. To fulfill this requirement, a robust, fast, flexible, scalable, and secure system has been developed. Commercial products and those deriving from other research projects have been explored, but no complete solutions have been found. The underlying mass-storage systems and search engine are commercial products; the rest of the application is homegrown.

The the architecture design of *SearchPlus* was driven by the following functional requirements. 1) Transformation of bibliographic data for scientific publications in different formats into a common XML format, storage for indexing and retrieval. 2) Processing of data in a secure environment behind a firewall and making it available to users through a web application outside the firewall. 3) Providing an information retrieval system in the form of a web application with a flexible interface, allowing search and retrieval of bibliographic data, linking to cited and citing articles, linking to full-text articles, and providing weekly alerts. 4) Delivering a responsive, interactive service. 5) Providing a reliable, fault-tolerant and highly-availability system[1], tolerating no loss of data. 6) Building a scalable and adaptable system capable of handling weekly updates, new data sources, formats, and content.

This paper makes two main contributions. A first is the description of the architecture of *SearchPlus*, which comprises a large collection of software and hardware components. We argue that the lessons learned may be of wide interest in the research community. The second contribution is the description of the methodology that has been used to optimize the performance, enhance the usability, and improve the robustness of *SearchPlus*. The rest of this paper is organized as follows: Section 2 outlines the software and hardware architecture of *SearchPlus*; section 3 provides insight into a large number of optimizations that have been performed on *SearchPlus*, together with their impact on the overall performance; section 4 provides concluding remarks.

2 Architecture

This section describes the architecture of the system, focusing on the main components and the reasons for choosing them. The overall architecture of the proposed environment is shown in Figure 1, the elements of the hardware configuration are listed in Table 1.

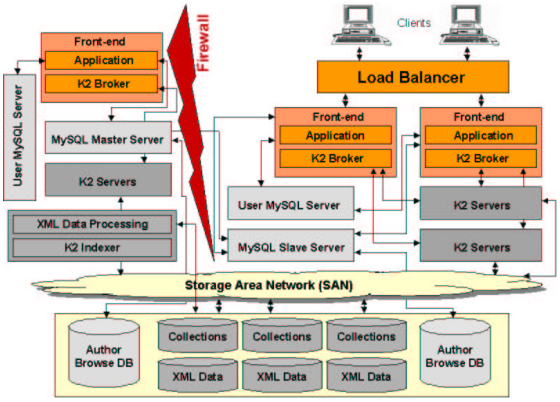

Fig. 1. The architecture of *SearchPlus*.

Processing Nodes	Processors	Main Memory	Disk Storage
12	46	234 GB	7 TB

Table 1. Elements of the hardware architecture of *SearchPlus*.

The physical architecture outside the firewall consists of a load balancer, two front-end systems running Verity K2 brokers and web applications, two systems running Verity K2 servers, a system running a MySQL server for an authentication/authorization database, and a system with a MySQL slave server for the author browse database. The architecture inside the firewall includes all the components residing outside the firewall with the exception of the load balancer and MySQL slave server. The MySQL master server and the XML data processing and K2 indexer system also reside inside the firewall. At a high level, there is a single user interface that provides a unified environment

for both data retrieval and citation linkage. Users can access all the functionalities independent of the physical architecture of the system. At a low level, the load balancer accepts client connections and balances them between two front-end systems. Each front-end server runs the web application and a K2 broker. The application examines a MySQL database to verify the user authentication and authorization rights to the integrated XML data. If that is successful the application sends a query to the K2 broker running on the same machine. The K2 broker forwards the query to the appropriate K2 servers which search the collections. The K2 servers return results to the K2 broker which sorts and returns them to the application. At this point, the application may also query the author browse and citation database to build bibliography and citation counts. Java and Apache/Tomcat have been chosen as the platforms to provide web accessibility. Access to the MySQL databases is handled using servlets and connection pooling. The main components of the hardware and software architecture on which we will focus our attention are the following: storage architecture, XML data layout, Verity K2 Enterprise, and MySQL database.

Storage Architecture Digital library data centers have demanding size, speed, reliability and flexibility requirements. In addition we also need to provide a secure environment for our data and systems. Our institution has a firewall with services inside and outside. All the data, systems, and services behind the firewall benefit from better security. However, we need to run our application and access data from outside the firewall for our external customers. Our choices for storage, file system, and distribution of processing have been predicated on protecting the data while keeping it easily accessible. Redundant Arrays of Inexpensive Disks (RAID) and Storage Area Network (SAN) technologies have been deployed to prevent data lost and provide storage capacity. The most complicated part of our design has been sharing data among servers located inside and outside the firewall. The combination of a SAN and a shared-access file system gives us the capability to achieve our objective. Sun's Quick File System (QFS), a Large Storage Configurations (LSC) file system, is designed to solve file system performance bottlenecks by maximizing the performance of the file system in conjunction with the underlying disk technology. QFS is implemented using a standard Solaris virtual file system interface and can be shared among Solaris environments. Several servers can read the data distributed in a file system, while another server can write and modify the same data. Using a shared-access file system, we can build, update, and modify data inside the firewall, while the web application outside the firewall has read-only access.

XML Data Layout Scientific articles typically have associated metadata which provides such information as author, title, abstract, keywords, source, volume, issue, number of page, etc. An article may also have a bibliography of citations (cited references). There are two sets of records, stored in XML format: those containing the metadata for scientific articles and those containing citations for the same articles. A record containing the metadata for an article is stored in a single file on a file system reserved for metadata records. For those articles with a bibliography, citation data is stored in a single file on a file system reserved for citation records. For citations that have a corresponding metadata record in our XML repository, a link is established from the cited reference to the corresponding metadata record.

Verity K2 Enterprise Native XML search engines have been evaluated and found to be underdeveloped, so we have investigated full-text search engines, settling on Verity K2 Enterprise (K2). Two of the compelling features of K2 are its distributed design and scalability. The design makes it easy to distribute indexing, search and retrieval, and administration. The K2 architecture consists of client, broker, server, admin server, and indexer components. The K2 client, in our case, refers to the Web application, developed in-house, which is integrated with K2 using Java. A K2 server is the core of the search and retrieval component. The K2 server contains the search engine for a specific set of indexed documents (collections) and the viewing service which renders documents returned by a search. The K2 broker manages communications between K2 clients and one or more K2 servers. When multiple collections are searched, each K2 Server performs a search against its collections, returning the results to the broker responsible for merging, sorting, and presenting results to the client. A broker can communicate with all its K2 servers simultaneously, whether or not they are on the same machine. Brokering enables scaling the system as the amount of information being searched for grows: brokers can be added according to demand. Similarly, as the number of documents to search grows, more K2 servers can be added, and collections can be mirrored to balance an increased load.

MySQL/Relational Database Why has a relational database been used to store data related to XML bibliographic records when they are already stored on file systems and searchable with the Verity search engine? The XML data repository consists of millions of small files, and backup and recovery of such a file system can be problematic. Mirroring the data stored on disk in a relational database offers the additional benefit of faster backup time and useful data redundancy. In addition to searching and retrieval, we need to provide browsing capability on authors and cited and citing articles, and dynamic citation counts. A relational database provides flexibility to build browsing functionality. MySQL is an open source relational database. Four reasons for choosing MySQL are (1) Speed: MySQL has proven to be fast at handling links among 1,435,000,000 rows of data in several virtual tables; (2) Data storage capabilities: we currently manage over 400GB of data; to limit individual table size, we take advantage of merged tables; (3) Fault tolerance: As mentioned, we use MySQL in production and as disk-based backup for our data; (4) Security: MySQL replication is used to protect and update our data; the master MySQL server runs behind a firewall, while a slave server accesses the data from outside the firewall, in read-only mode.

3 Performance Analysis and Improvements

In order to determine why application performance was not as good as expected—the initial response time for a simple search was tens of seconds—we undertook a number of performance studies. To simplify the process we concerned ourselves with examination of individual problems. We analyzed each component of the overall architecture: layout of disk arrays and file systems, memory use, the benefits of running in a 32-bit versus a 64-bit environment, network infrastructure, tools used by the application (MySQL DB, Verity K2 Engine, Java Virtual Machine, Java compiler, XSL, JSP, Apache/Tomcat) and of course, the application code itself. We estimated the impact

of every component on performance and scalability and focused our tuning efforts on those which would provide the most benefit.

3.1 Hardware Architecture

Hardware performance tuning efforts were focused on evaluating the number of database/collection servers, the number of CPUs per server, the amount of memory needed, and the number of files per file system. Initial assessments of performance typically involve processor speed or memory consumption, not transfer rates to and from disk storage. Since disks are several orders of magnitude slower than RAM, avoiding access to disk and making necessary access as fast as possible can have a huge impact on application performance. To solve this problem, we have used RAID technology. If configured properly (several experiments were performed using different disk striping strategies), the disk arrays are fast, cheap (considering the amount of data stored) and safer. We have stored over 7 TB of data, distributed in two categories: the first set consists of millions of small files laid out on file systems, the second of MySQL database tables. These require different choices for disk configuration. We looked at disk organization and layout, making sure that all parameters were appropriately tuned for the type of data stored. To sum up, important decisions impacting the I/O performance were choosing a correct page size, the unit of disk, the metadata, and data distribution of each file system. Looking at the network infrastructure, all the servers and storage devices that must communicate have been connected on the same network path and controlled by the same network switch to minimize latency and network delays.

3.2 Verity Tuning Optimizations

We knew before the project started that there would be a large number of users accessing the application, but we did not have a clear picture of the search distribution. Some time was spent monitoring how the Verity collections were queried and how many users access the system in a specific time frame. We also performed a number of optimizations, mostly on the allocation and caching policies of Verity K2, with improvements shown in Table 2. As can be seen, many of them had a significant performance impact.

Broker Caching	Broker Thread Allocation	Sever Caching	Server Thread Allocation
20%	15%	22%	18%

Table 2. Impact of Verity K2 optimizations on the basic performance.

3.3 MySQL Optimization

Over 400 GB of data are stored in the MySQL database. MySQL server configuration, table structure, table allocation, query handling, concurrency, and replication were examined for optimization.

Server Optimization The first component analyzed was the MySQL server, its compilation, and linkage. By using a suitable compiler and appropriate compiler options, a 10-30% speed increase was realized. MySQL was compiled to take advantage of the 64-bit Solaris architecture and address up to 32 GB of memory, very helpful for query caching. We observed how MySQL server uses system memory, how the memory is

shared, and how it is used by MySQL threads when performing queries. Several experiments were done tuning the size of the memory buffer that MySQL uses for storing the indexed data. Several tests were made using different block size for storing and retrieving the indexed data.

Table Structure Optimization The first table optimization involved structuring the data and indexes to take as little space as possible on the disk and in memory. This results in significant improvements because disk reads are faster and less memory will be used. Indexing also takes fewer resources if done on smaller columns. The second optimization was to lay out the tables accessed at the same time on different file systems. The third and ongoing process of optimization involves running the MySQL table check mechanism to remove fragmentation and re-sort the indexes after updates. The fourth performance improvement that has been undertaken, which has not yet been completed, is restructuring the data inside the tables so that we can take advantage of some of the newer features of MySQL.

Replication MySQL replication has been used in our architecture to protect and update our data. When using the MyISAM index mechanism, MySQL has extremely fast table locking (multiple readers/single writer). The biggest problem with this table type occurs when one has a mix of a steady stream of updates and slow selects on the same table. One way to address this problem is to use MySQL replication, where a master server updates the data in a secure environment and a slave server gets the data from the master and provides the data to the users. Using this mechanism, we have been able to reduce the MySQL query time while updates are made on the same data. Replication also provides a secure environment for the data to be updated. Table 3 summarizes the impact of the major MySQL optimizations.

Compilation	Tables	Block Size	Table Structure	Buffer Cache	Query Cache
15%	from 24 to 2 hours	5%	20%	25%	30%

Table 3. Impact of MySQL optimizations on the basic performance.

4 Conclusion

The main motivation of *SearchPlus* has been to develop a powerful, responsive, robust, and intelligent distributed database environment for knowledge discovery information. We have described the architecture of our relatively complex, multi-TB system and outlined our performance optimization methodology. All the optimizations performed have reduced the response time of the system to less than three seconds, a substantial performance improvement with respect to the original system. *SearchPlus* is therefore very responsive and fault-tolerant and is currently used by a sizable number of customers.

References

1. A. Fox and D. Patterson. Self-Repairing Computers. *Scientific American*, 288(6):54–61, 2003.
2. R. Luce. Evolution and Scientific Literature: Towards a Decentralized Adaptive Web. *Nature*, May 2001.