A High Performance Data Server Optimized for HEP Data

Karpjoo Jeong [a], Theodore Johnson [a], Hankil Yoon [a], Martin Lohner [b], Paul Avery [b]

[a] *CISE Department, University of Florida, Gainesville, FL 32611, USA*

[b] *Department of Physics, University of Florida, Gainesville, FL 32611, USA*

We present a data server whose major design goal is to support HEP data analysis efficiently by exploiting its application characteristics aggressively. We explain the design and current implementation status and give some preliminary experimental results.

Key words: Data Server, High Performance I/O, Tertiary storage, High Energy Physics

1 Introduction

HEP event analysis provides a great challenge for data management both because of the amount of data which must be processed many times and the fact that each analysis job can potentially access the entire data set. This requires a data access system which pays careful attention to ensuring high performance.

However, HEP event analysis has great potential for high performance due to its special I/O characteristics which allow us to predict all the I/O operations at the beginning of every analysis job and to take advantage of a significant amount of parallelism in data access.

The Nile project [1] team is developing a high performance data management system [2] for the CLEO experiment. As part of the effort, we are building a high performance data server whose design is optimized for HEP data by exploiting HEP application characteristics. That is, we attempt to cope with the HEP-raised challenge by HEP-specific characteristics.

Although we intend the data server to be an inexpensive and high performance alternative to general purpose DBMSs or storage managers, we believe that optimization techniques developed for the data server should also be useful when tuning those general purpose systems for HEP data.

2 Design

Our design approach is: (1) to support parallelism and prefetching at various levels, (2) to avoid reading unnecessary data, and (3) to provide only limited functionality for rare operations (e.g. update). For example, logging and transaction management is performed at a higher level (on much fewer data). HEP application characteristics allow us to take this aggressive and optimistic approach.

HEP event analysis characteristics: There are three major characteristics that facilitate high performance data access. First, events are independent and the access order is not important – *event independence*. Therefore, each event can be independently managed and read for analyses, provided that each event is guaranteed to be read for each analysis only once. This allows event-level parallelism and a lot of flexibility in data distribution.

The user explicitly specifies all the input/output data at the beginning of an analysis job either as a list of datasets to process, as index files (i.e. sets of pointers to events), or skims (i.e. sets of events) – *availability of the input specification*. This allows aggressive prefetching.

HEP data is mostly read, only rarely written – *extremely-rare update*. This makes expensive DBMS operations for concurrency control and reliability and consistency maintenance unnecessary.

Distributed Data Server Architecture: Our data server is distributed in a *highly-scalable* way. The *event independence* and *extremely-rare update* properties allow this design. In order to manage large volumes of HEP data, we run multiple independent data servers. We horizontally partition the entire data set into "chunks" (e.g. certain data ranges), and distribute the chunks among data servers. Since the order of events is not important, events can be processed independently (as long as integrated statistical parameters are gathered correctly); an analysis job is executed by concurrently performing the analysis on each individual chunk, then gathering statistics. In the Nile distributed computing environment [2], a job scheduler starts multiple analysis processes and assigns chunks to them. These analysis sub-jobs read each assigned chunk from the corresponding data server concurrently. Apparently, the aggregate data transfer rate of this *parallel interface* can scale up easily to a large scale distributed computing system.

In spite of this distribution, we still provide a *single database view* for the user via a centralized metadatabase which maintains information about the data chunks (e.g. data locations). Thus, the metadatabase hides distribution details from the user and provides information (e.g. data locations) to the job scheduler. In addition the metadatabase also supports WWW-based access and wide-area data sharing and collaboration. We use a commercial database package (currently Illustra) to provide the functionality required by the metadatabase. Figure 1 illustrates this architecture.

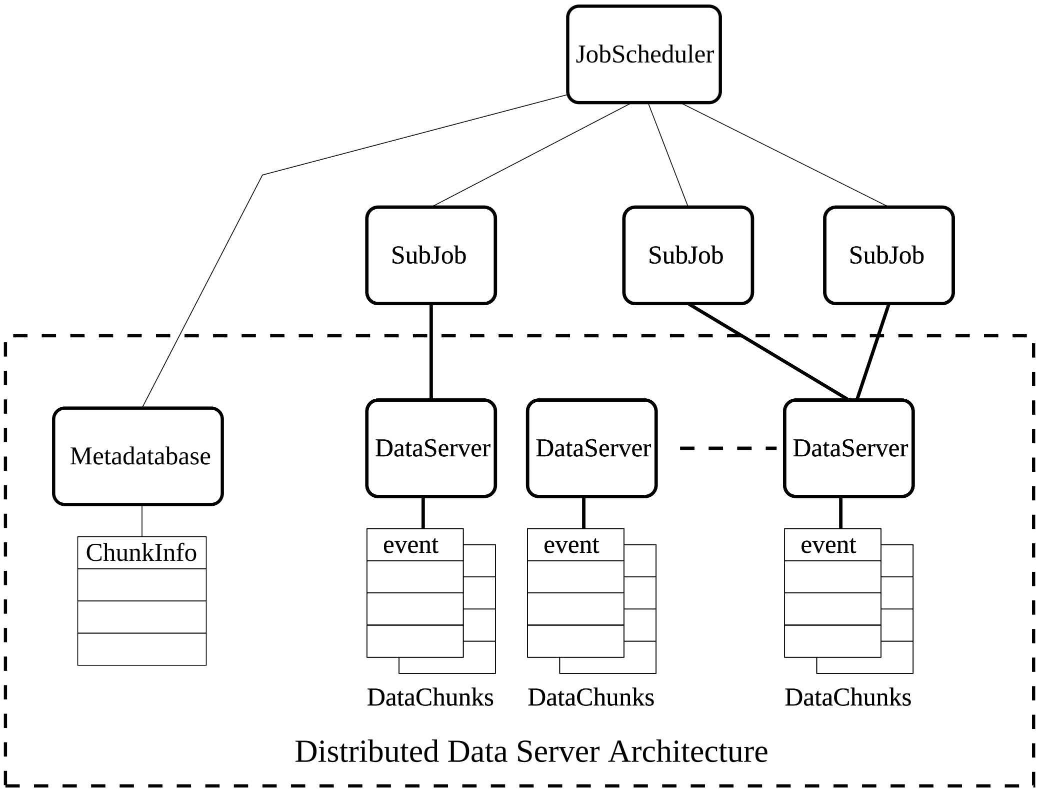

Fig. 1. Distributed Data Server Architecture

Local Data Server Architecture: Each local data server partitions data into multiple local disks and supports *parallel I/O* (to increase through-put) and *prefetching of data* (to hide disk access latency). In order to prefetch data from disks, the data server executes as multiple "front-end" and "back-end" servers. Front-end servers interact with analysis jobs while back-end servers prefetch data from disks to shared buffers. Figure 2 illustrates this architecture.

Vertical Event Partitioning: The data server vertically partitions event data and allows analysis jobs to read only necessary "components" (attributes) in order to reduce the disk I/O. However, this requires combining components at runtime. In order to execute these operations efficiently, the data server maintains all the components in sorted order by an identifier (currently

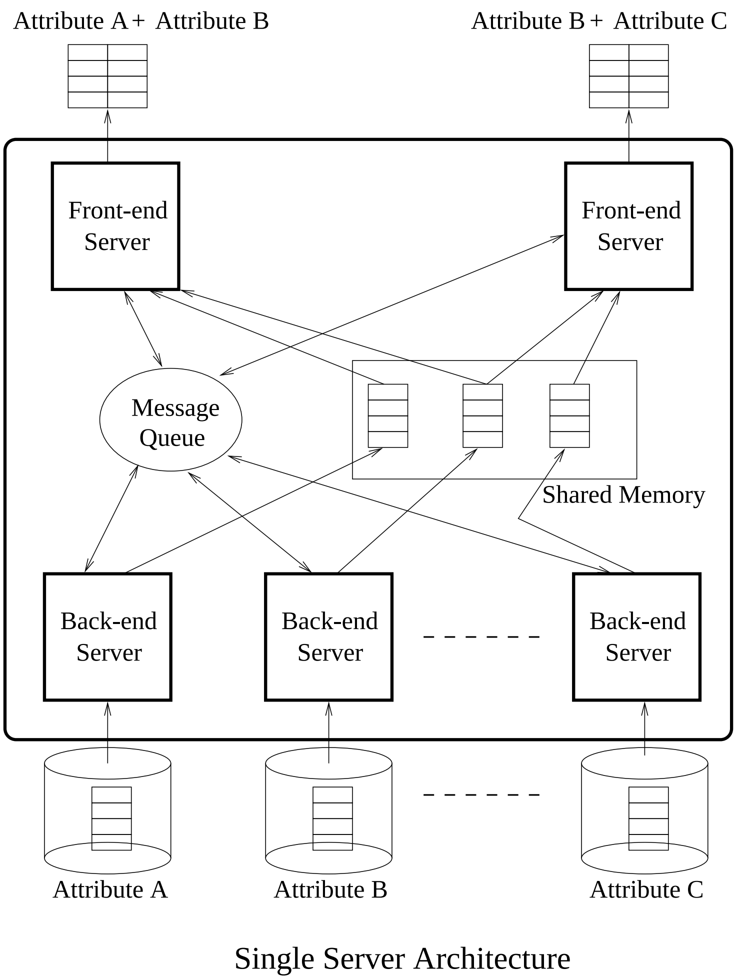

Fig. 2. Local Data Server Architecture

run and event number) and uses a “merge-join” algorithm. We illustrate this scheme in Figure 2.

Tertiary Storage Support: Tertiary storage is essential for HEP data due to its large volume. We have been extending the data server to support tertiary storage involving a tape robot. Unlike disks, tapes are not random-access storage media and have a very different performance behavior. Therefore, we need access strategies suited to tapes which are different from those for disks.

Currently, we are considering two data distribution schemes and developing I/O scheduling techniques which hide high mount and seek latencies of the tape robot and tape drive. In the first scheme we store hot attributes on disk and cold attributes on tape and combine them on-the-fly. Tapes are just treated as slower disks and the basic I/O scheduling approach is to prefetch cold attributes as aggressively as possible but to prefetch hot attributes to the extent that hot and cold attributes keep pace together. In the second scheme we store only hot attributes on disk but complete events on tape. In this scheme, analysis jobs that require cold attributes access only tapes. The I/O scheduling focuses only on prefetching data from tapes.

The advantage of the former scheme is to use the same data format for disks and tapes; the advantage of the latter is no need to synchronize hot and cold attributes (i.e., disks and tapes).

Management Support: Besides features for high performance data access, our data server provides mechanisms for facilitating management. The data server allows the system administrator to re-partition events vertically and horizontally in order to balance load on workstations or disks. Moreover the system administrator can move data (e.g. a collection of events) to another server as a single operation. The metadatabase is automatically updated to reflect these changes, maintaining consistency with the data servers.

3 Current Status and Experimental Results

A prototype of the data server has been implemented using C++ and standard UNIX system functions. We tested the basic functionality with CLEO II event data as part of the CLEO III Data Access System prototype [3]. Examples of different data components in the test were track hypotheses (pion, kaon, electron, muon, etc.) and electromagnetic showers. A simple data analysis needed only the kaon and pion track hypotheses, and therefore only those components were read in and used in the analysis.

To test the raw I/O performance of the data server prototype, we used 1.2 GB of dummy data. We examined the performance behavior with respect to: number of concurrently read components; the size of a read block; and the number of attached disks. Multiple disks not only reduce the number of seek operations per disk but can also increase parallelism in disk I/O. We find that once the size of a read block is bigger than 16 kB, the performance does not vary with respect to the number of components read, and in spite of overheads due to various server operations, the overall performance of the data server is very close to that of the UNIX file I/O (i.e. reading data using the UNIX "read" system call without further processing). Figure 3 summarizes the performance results graphically.

We have also implemented a metadatabase system as a "stand-alone" server. The system maintains meta-data in a commercial DBMS (for concurrency control and reliability) and provides CLEO collaborators at various institutes with wide-area access via a Web interface. The system allows the user to browse meta-data, follow hypertext links, or retrieve information using template-based queries.

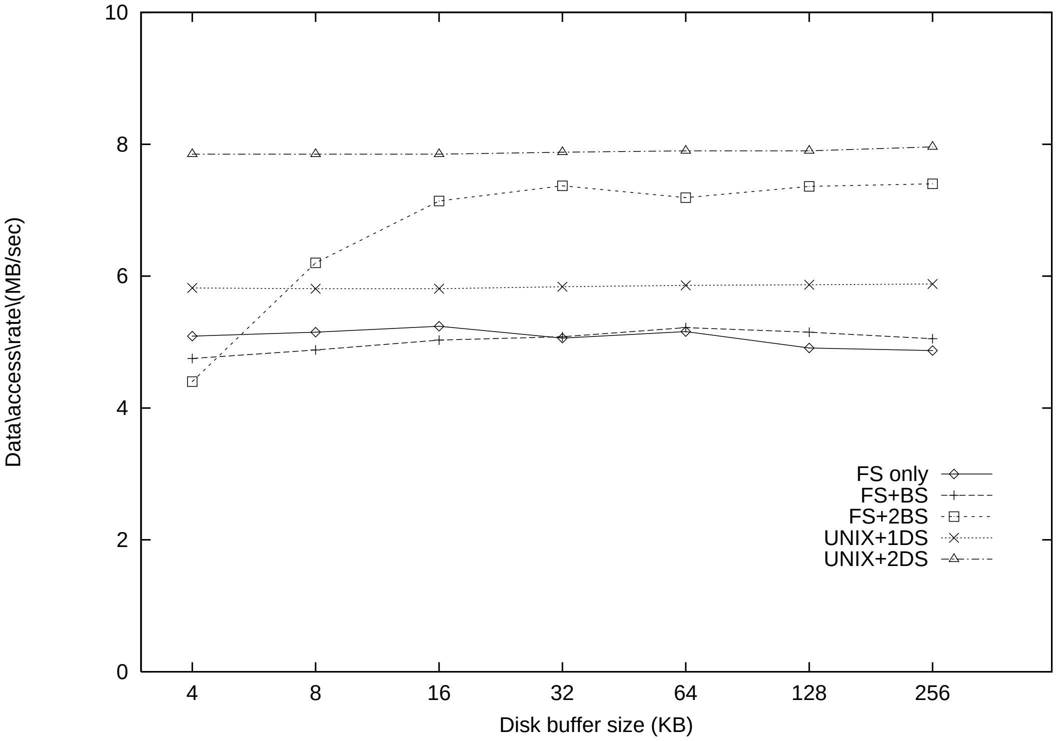

Fig. 3. Performance of the Data Server Prototype in terms of data access rate (MB/sec) vs size of the disk buffer (kB) for various configurations: a front-end server without back-end servers (FS only), with one back-end server (FS+BS), and with two back-end servers (FS+2BS). As an upper bound we show the performance of UNIX file I/O with either one (UNIX+1DS) or two disks attached (UNIX+2DS).

Future plans include a multi-threaded data server design and further study of tertiary storage options. Threads will allow us to optimize disk I/O for multiple clients; for instance data need to be read only once for clients which requested the same data set.

This work is funded by the Department of Energy, grant DEFG0586ER40272, and the National Science Foundation, grant ASC-9318151.

References

[1] Please see `http://www.nile.utexas.edu`.

[2] Mike Ogg *et.al.*, *Nile's Distributed Computing Site Architecture*, CHEP97 Conference Proceedings.

[3] Chris Jones *et.al.*, *Design and Implementation of the CLEO III Data Access System*, CHEP97 Conference Proceedings.