Fermilab FERMILAB-Conf-01/101 June 2001

Distributed Data Access and Resource Management in the D0 SAM System

I. Terekhov, R. Pordes, V. White, L. Lueking,
L. Carpenter, H. Schellman†, J. Trumbo, S. Veseli, M. Vranicar, S. White

Fermi National Accelerator Laboratory, Batavia, IL 60510
† Northwestern University, Evanston, IL, 60208

Abstract

SAM (Sequential Access through Meta-data) is the data access and job management system for the D0 high energy physics experiment at Fermilab. The SAM system is being developed and used to handle the Petabyte-scale experiment data, accessed by hundreds of D0 collaborators scattered around the world. In this paper, we present solutions to some of the distributed data processing problems from the perspective of real experience dealing with mission-critical data. We concentrate on the distributed disk caching, resource management and job control. The system has elements of the Grid Computing and has features applicable to data-intensive computing in general.

1. Introduction

The SAM system[1] is being developed for the D0 RunII High Energy Physics (HEP) experiment at Fermilab, see [2, 3] for a recent review. HEP experiments are typically characterized by data-intensive applications, and SAM performs functions that are common to all settings. Most notable functions are data replication, disk cache management, resource management and job control, data catalogs, dataset definition and processing history. In the present paper, we concentrate on distributed caching and resource management; metadata management is more D0-specific and is described elsewhere.

With over 60 participating collaborator institutions, SAM is a truly global distributed system. As such, the system has elements of the Grid, although it is not strictly a Grid in a sense that e.g., it handles distributed collections of data and its processing rather than links heterogenous collections.

In the data-intensive environment, the computational economy is extremely data-centric. Consequently, SAM is primarily (and historically) a data handling system, and most of the following is given from such perspective.

1.1. Applications and Data Intensity

The D0 experiment collects (from the FNAL collider) and studies particle collision data. During its operation in the first few years of the new millennium, the experiment will accumulate the total of about 1 Peta-byte of event data. Each event is represented in a D0-specific format and usually takes about 250 Kilo-bytes of digitized data. Events are recorded at a rate of a few Hertz.

The D0 applications process sequentially, i.e., one at a time, these event objects stored in files. The end user deals with unordered collections of events, the D0 application infrastructure[4] translates such logical collections into physical collections of files. When, for example, an application generates a stream of events to be imported into the system, the infrastructure chops the stream into files. A complete file is made self-describing, i.e. it contains information about the events stored. The application infrastructure then hands the file and its detailed description over to SAM. Thus, the physical data movement in SAM is done at the file level whereas the application infra-structure performs object-level access.

There are several types of applications representing corresponding types of data processing, which we roughly outline here. One type produces data, by either digitizing real detector read-outs or performing Monte-Carlo simulation. Real data taking is the most critical for the experiment, of course, but it is not as strenuous on the system as the subsequent studies of the data. Another application type reconstructs the data, i.e., uses sophisticated scientific algorithms to translate raw read-outs into particles and similar objects. This processing is both CPU and I/O intensive but has to be done only a finite number of times (usually once) for every collected event. Finally, a recurrent analysis of the reconstructed data occurs indefinite number of times on a time scale usually extending years beyond the actual data taking from the accelerator. Typically, analysis applications filter events by certain criteria or extract highly condensed variables for histogramming. This recurrent activity is mostly I/O intensive.

While the SAM architecture handles all of the above application types, the most interesting and difficult problems occur with the analysis type applications. Other processing types, aside from having to be performed few times, usually can have dedicated hardware (and people) resources. Thus, for the purposes of resource management, we categorize our applications as highly data I/O intensive.

1.2. SAM Meta-data

The SAM architecture uses CORBA[5] as the means for client-server communication. The Meta-data in SAM is also accessible via the several CORBA interfaces. Its complete description may be found in some of the project reviews[2, 3] as well as on the SAM project Web pages[1]. Here we give a brief description of selected aspects. For the end user, the interfaces exist as command-line and Web-based forms. The most useful user-visible meta-data are the global event catalog (short description of each event and a mapping into file containing them) as well as the catalog of logical data collections called *datasets*. Various applications may store logical datasets, as well as combine and reuse them. In addition, users may view processing history of datasets as well as of individual physical data units (files), for book-keeping and error recovery.

Internally, SAM also uses meta-data for the physical data and resource management. Of primary importance is the replica catalog where each physical file is associated with zero or more *locations*. These include Mass Storage System (MSS) locations (e.g. in an Automated Tape Library, ATL), SAM-managed disks, and externally managed locations. (Administrative interfaces exist for managers of the latter to update the replica catalog.) In addition, SAM maintains persistently configuration and dynamic state of certain servers, for the purposes of crash recovery.

The meta-data catalog is implemented at present in a centralized Oracle database. The D0 experiment contemplates replication and decentralization of the metadata; however at present it is satisfied with the performance and reliability of the centrally managed database.

We emphasize that unlike the meta-data catalog, the physical data handling (which is the primary factor in user application latencies), is already fully distributed as we discuss in the remainder of the paper.

2. Data Replication

The SAM system has advanced facilities for distributed caching and data replication. These are realized through a network of processing *stations* which provide both local caching services and global data exchange across the WAN at the multi-Terabyte level. In the following, we will sometimes use "data replication" inter-changeably with "caching". The term "caching" is largely historical: canonically, the data-intensive HEP applications relied heavily on the Mass Storage Systems (MSS) as the primary places for data storage, with disks serving as caches for the most actively accessed data. It is remarkable that, with the proliferation of cost-efficient clusters and de-centralization of computing resources, exponentially growing amounts of distributed disk become available so that the distributed disk cache becomes a storage in its own right at D0; in fact, the distributed disk is the primary storage for the analysis applications. The term "caching" also represents the physical proximity of the data to the application, which in D0 case always accesses local disk, whether it was the data or the application that was brought to the proximity.

2.1. Local Data Replication for Cluster Computing

A *station* in SAM is a locally managed collection of machines with their resources. In the data-intensive world the most important local resource is the disk, that is why the *station master* was originally, and is primarily, responsible for management of data on disk. Thus, the station master keeps track of a collection of disk mounted possibly on physically multiple machines, and the SAM files on the disks. When users run their jobs, their requested datasets are made known to the station master, Sec. 3.1, so that it can replenish the caches, possibly erasing unused files.

In doing so, the SM analyzes the access history of each cached file (also stored persistently as part of the meta-data) and uses one of the several cache replacement algorithms available to different groups. Cache usage is categorized by groups of physics researchers, and different groups may use independent caching algorithms. In the course of the cache replacement, the SM updates the aforementioned replica catalog accordingly. See our earlier description[6] for more details.

Eventually (whenever the underlying Batch System decides), user processes are started on the cluster for this data consumption *project*. From SAM's viewpoint, processes of the same job are *consumer processes* sharing the same *consumer ID*. A special server called *project master* (PM) coordinates input data consumption[6]. In particular, it ensures that each file from the project dataset is delivered to exactly one process. (For a special SAM access mode called "Freight Train" the PM also supports multiple consumers whereby each consumer must "see" each file from the dataset.) It is the PM that on its consumer behalf communicates to the SM files from the dataset being released, so as to trigger the cache replacement algorithm in the SM when necessary. As we will describe later (Sec. 3.1), the SM interacts with the user job submission system (the Batch System) so as to attempt the proximity of user jobs to the

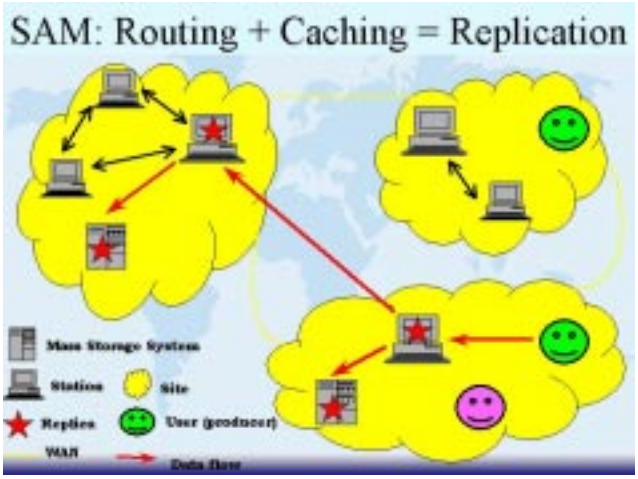

Figure 1. File routing while storing data with SAM.

data. Nevertheless, the SM will in general replicate the data on demand within the cluster as to ensure a proper data consumption. Such intra-cluster, or *intra-station* in SAM terminology, transfers are executed by the SM based on the knowledge of *consumption sites* for each job provided to it by the project master.

The SM translates *consumption sites* information into the one about the *delivery sites* using a Cache Accessibility Matrix which specifies what disks are accessible from what nodes. This matrix is part of the station configuration stored persistently as part of the SAM metadata catalog. In the absence of shared file systems, the matrix is a unity, i.e., consumer processes running at a node may only access data from that node. In order to efficiently allow the highly intensive I/O for the applications running on clusters of low-end workstations, SAM never relies on shared file systems and always provides the data locally.

2.2. Global Data Replication

In addition to local caching and replication, SAM replicates data globally, i.e., at multiple sites (institutions participating in the D0 collaboration). SAM provides this service in a dynamic and fully automated fashion by routing of data through the network of communicating stations, each managing its disk cache. The exact location on site and lifetime of each replica is therefore controlled by the local station(s) as described in Sec. 2.1. In this section, we consider the station a point-like node on the global grid. The SAM grid is formed by the stations as well as MSS's connected with Wide-Area Network (WAN) links, see Fig. 1. The MSS's are leaf nodes whereas stations are capable of *data routing*.

Consider a real example that includes SAM installation at Nikhef in the Netherlands, which is a D0 collaborator institution. Files are produced at one station at Nikhef which is specifically tailored for Monte-Carlo generation of simulated data. The data is then routed to Fermilab site, to either a station or to the Fermilab's primary Mass Storage System (Enstore[7]). In the end, there are replicas of the data both locally in the Netherlands and at FNAL (in addition to the MSS's when applicable). Thus, the two SAM features of data routing and disk caching in combination yield global data replication in a dynamic and natural fashion, see [8].

When a SAM station on such a Grid retrieves a file, it will need to choose the best location for each replica. In the future, SAM will use information about relative costs of obtaining a replica. At present, we have given the station administrators the facility to configure their stations as to prefer, semi-statically, some locations over other (e.g., disk over tape, onsite location over offsite, etc.).

3. Computational Economy: Resource Management

Given the volume of data and access intensity, hardware resource management (RM) is critical to SAM. While many resource management issues are inter-related, it has been possible to architect a hierarchical RM scheme. The goals of all the RM components are fundamentally common, however, and stem from the experiment (conflicting, and therefore prioritized) requirements:

1. Implement the experiment policies for resource usage by data access modes (e.g., types of processing) and by research groups of the experiment members;

2. Optimize the resource usage so as to maximize the overall throughput defined in terms of "real" (physics applications) jobs.

In what follows we describe the two levels of RM in SAM as well as the common strategies used to achieve the goals.

3.1. The Hierarchy of Resource Managers

At the level of the processing stations, the SAM system integrates the data delivery and caching infrastructure with the user job control (allocation and scheduling). Such integration is one of SAM's main novelties; it has never been fully addressed before despite its fundamental importance to the data-intensive HEP computing. In its initial implementation, SAM performs user job control through interfaces to the abstract batch system. The batch system uses its specific, opaque logic to schedule and run data processing jobs using SAM-supplied additional constraints and requirements. By co-allocating processing (computing)

resources together with the data delivery resources SAM increases overall throughput. Furthermore, while a pure batch system typically schedules user jobs based on highly compute-centered considerations (e.g., CPU usage), SAM provides a more comprehensive and flexible framework described later.

From SAM's perspective, the end user job consists of two parts: the input dataset (project) to be processed and the application, such as a script, name to invoke. The following sequence of events occurs when a user submits a job to SAM.

1. The SM client forwards the request to the SM which retrieves the explicit file set comprising the project from the Meta-data catalog;
2. The SM initiates input file delivery for the project as the disk space becomes available;
3. The server further determines whether there is enough data (at least one file in fact) to start the user job; the job is marked as runnable or suspended accordingly;
4. The SM client wraps the user application with another SAM client and submits the job into the batch system, possibly in a state suspended "until further notice" from SAM. The exact suspension mechanism depends on the Batch System and may be a resource specification or external event occurrence;
5. If there were not enough data initially, then when eventually the data start arriving, the SM notifies the Batch System via the adapter, which takes the BS-specific action and the job is released, i.e., becomes runnable;
6. When the job is eventually dispatched, the SAM wrapper performs some SAM-related book-keeping in a user-transparent way.

Note that the most of the job scheduling is done by the Batch System; all that SAM really needs to do is impose one additional requirement (data availability) onto each job.

More resource management is done at the site level. Each SAM site - consisting of multiple compute, storage and caching resources - is controlled through a "Site Optimizer". The Site Optimizer accepts information about the state, current and past performance of the components of the system. We anticipate that the most scarce resource for delivery of D0 data files will be the movement of the robot arm to mount the tapes, and the number of tape drives allocated to the group. In the current initial implementation, therefore, for each data analysis job submitted the optimizer evaluates the resources required in delivery of the data files. It looks at the number of tape mounts in the robot, the number of tape drives used, the current performance of the route from the storage system and the expected rate of file processing by the job. It authorizes execution of the jobs based on the resource needs and their current availability. Scheduling of the job execution is based on an attempt to minimize the number of tape mounts and the length of time the tape drives will be "occupied" transferring data. In the next version information gathered from the actual network performance and file delivery will be incorporated into the scheduling algorithms.

In the present SAM implementation, all the sites share the site optimizer so that it is in fact global. In the future, we plan to split such Optimizer into at least two levels: site-wide (controlling local MSS's and LANs) and truly global (controlling inter-station data transfers over the WAN).

3.2. Economic Concepts for Job Scheduling

For the most efficient global data and job management, a unified approach to the allocation and scheduling of all the resources is being developed that pertains both to data delivery jobs and to the user jobs. It is based on the economics concepts of "benefit", "cost", "value", "fair share". Scheduling of jobs is based on a fair share allocation (FSA) algorithm. We say that execution of a job (either data delivery job or real user job) *benefits* the associated abstract user. Abstract user is either a physics research group (D0 chose not to manage resources on a person level) or a special data processing activity done on behalf of the entire experiment, such as data taking from the detector. The benefit is a vector of calculable quantities. Benefit types (components of the vector) include different resource usages as well as number of data units processed, amount of "useful physics work" done, etc. We additionally compute the cumulative benefit vector received by an abstract user. From the vector, we compute a scalar benefit for each abstract user (see below). For every submitted job, therefore, we can determine where in the queue of jobs it should be placed, by comparing the total scalar benefit received by the user with the allotted (fair share) fraction. This comparison is done at the time of job submission and optionally during the job run (depending on the functionality offered by the particular underlying queuing system). Benefits are accounted every time when significant events occur such as file transfer, file processing, whole job completion, etc.

The exact way the scalar benefit is computed is outside of the scheduling algorithm. A very practical way to compute the scalar benefit for an abstract user is a linear weighted sum of the benefit vector components (to be exact, the vector components are dimensionless fractions of each benefit received by the user). Thus, each benefit type is assigned a dynamically configurable weight. The numerical values of these weights represent the current views of D0 (and/or a local processing station administrator) on the relative importance of the different resource types as well as amount

of work done. For example, at the time of writing this paper, D0's most scarce resource is, as was already mentioned above, the tape volume mount in the ATL (robot), thus, the weight for the benefit type "volume mounts" will likely be higher than others.

The general job scheduling scheme based on benefit accounting is another principal novelty of the SAM system presented in this paper. Our scheme is dramatically more flexible than existing job scheduling systems. For example, the LSF commercial batch system bases FSA solely on the CPU usage (which in fact is not even the most important factor in data-intensive computing); such a simple scheme is trivially reproduced in SAM by setting the "CPU" benefit type weight to one and all other weights to zero. Benefit weight reconfiguration can be done at run time.

Our present "benefits" are in fact a mixture of costs and real benefits. In the future, we plan to develop a more consistent economic framework of costs and benefits, inspired by works on computational economy [9, 10, 11]. Such a framework could be useful for the general problem of scheduling of user jobs in the data-intensive environment such as SAM. One simple illustration of the problem is given by the dispatch of physics analysis jobs on a cluster where the set of nodes whose disks hold data does not, at any given time, correlate with the set of the nodes whose processors are idle. The application is data-intensive and the job control system must balance the "expensive" data delivery onto the nodes which don't have the job data with forcing the job to wait for the processors where the data is already present. The decisions must be made depending on relative "costs" of data delivery and on user benefits which decrease as the job latency increases. The data transfer costs must incorporate experiment policies, local prioritization of system resources, and conditions at the mass storage systems. All such costs are dynamic and influenced by the job placement decisions (since job submission initiates data delivery soon), yet largely measurable and predictable.

On the global scale of Grid Computing, one similarly seeks to schedule a job at an optimal site participating in the Grid. For the SAM system, it is interesting to be able to broker user jobs, or projects, as to balance them properly among the stations. Work is being done[12, 13] that address such issues but concentrate on problems arising from the crossing of administrative boundaries (e.g., global authentication, heterogenous environments, etc.). Obviously, in addition to such considerations, we need to build a metacomputing system for D0 where the data availability drives the costs of jobs and therefore their scheduling.

3.3. Deadlock Prevention

As in any resource managing system, we need to address the issue of deadlock, by means of either prevention, avoidance or resolution[14]. The data handling facet per se of the SAM system is deadlock-free by design. Remember that the access in **S**AM is **S**equential so that an application can wait on a file delivery only when it has released the previous file (i.e. all other files) from the dataset, Sec. 1.1. Thus, the SAM application does not use any data handling resources (the disk space) when it is blocking for a data handling resource (by means of blocking on a file delivery). Thus, the pure sequential access *prevents* deadlock by removing the *partial allocation* condition.

When compute resources (CPU, memory, etc.) are added to the data handling resources, the situation changes. A job that blocks on input file delivery (with output storing being similar) does, in fact, hold compute resources, i.e., there occurs partial allocation across the different resource types. Although a single-threaded application does not consume CPU under such conditions, all applications hold, most notably, virtual memory for the process (10-100 MB for a typical D0 application) and possibly network connections. Ultimately, the Batch System may not dispatch new jobs whose data may be available on local disk, because of the shortage of virtual memory and other compute resources.

We have designed a timeout-with-restart scheme where such an application receives a special message from the project master introduced in Sec. 2.1 and by default, the SAM client in the D0 application framework causes the application to exit. If the user so desires, the job is then resubmitted. If one regards the re-submitted job as the old job being resumed, we say that we have therefore *preempted* the compute resources. If one views the re-submitted job as a new job, then we say that by aborting the application we have resolved the deadlock. The difference in views is largely cultural but also depends on the underlying batch system (most BS's, however, support the re-submission concept).

Note that, whether a job is submitted initially or after its SAM application has timed out waiting for data, the job will never be dispatched unless it can process at least one file from the dataset before it may block again, see the discussion of SM-BS integration in Sec. 3.1. In addition to completing our deadlock prevention scheme, this important consideration also prevents a livelock, i.e., the condition where jobs are continuously dispatched and then pre-empted without making progress!

4. The Project Status

The SAM system has been operational for some two years. It has been successfully used for storing of several Terabytes of Monte-Carlo simulated data, which was subsequently reconstructed several times with many datasets further analyzed repeatedly for the purposes of scientific algorithm development by the physicists. In March 2001, the

experiment has begun real data taking, and real data processing both on the FNAL site and outside is commencing this summer. Collaborators use SAM at their home institutions, with local batch and storage systems, and implement local site policies for resource allocation and use. Soon we will possess data on the user access patterns; however, already in the ramp-up stage of data taking and processing we begin to see the benefits of disk caching and data replications.

To support the global and universal data access, SAM is being interfaced to various Mass Storage Systems and to various batch systems - Condor (see [15]), LSF, PBS, etc. - and different optimizing algorithms are being implemented and tested. As part of the proposed Particle Physics Data Grid project, D0 is planning to collaborate with Computer Scientists and incorporate the Grid technologies for job control, global resource allocation and optimization.

Acknowledgements

The authors are grateful to the anonymous referees whose insightful comments have helped us improve the content and clarity of the paper. This work is sponsored by DOE contract No. DE-AC02-76CH03000.

References

[1] The SAM team, L.Lueking (co-leader), V.White (co-leader), L. Loebel-Carpenter, C. Moore, H.Schellman, I.Terekhov, J.Trumbo, S.Veseli, M.Vranicar. The home page `http://d0db.fnal.gov/sam`.

[2] L. Carpenter *et. al. SAM Overview and Operation at the D0 Experiment*, submitted to The International Conference on Computing in High Energy and Nuclear Physics (CHEP 2001), September, 2001.

[3] V.White for the SAM project, *The Data Access Layer for D0 Run II: Design and Features of SAM*, talk given at The International Conference on Computing in High Energy and Nuclear Physics (CHEP 2000), February, 2000, Padova, Italy,

[4] The D0 application framework is described by J. Kowalkowski, in *The D0 Framework*, talk given at CHEP2000, see [3].

[5] The OMG home page `http://www.omg.com`. For C++ components, "ORBacus for C++ and Java" `http://www.ooc.com`. For Python components, "Fnorb, a Python ORB" `http://www.fnorb.org`.

[6] I. Terekhov and V. White, *Distributed Data Access in the Sequential Access Model at the D0 Experiment at Fermilab*, poster presentation in proceedings of The Ninth IEEE International Symposium on High Performance Distributed Computing, Aug, 2000, Pittsburgh, PA

[7] The Enstore home page `http://www-isd.fnal.gov/enstore`.

[8] I. Terekhov *et. al. SAM for D0 - a fully distributed data access system*, talk presented at VII International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2000), Oct. 2000, Batavia, IL, in proceedings.

[9] B. Awerbuch, S. Kutten, D. Peleg, *Competitive Distributed Job Scheduling*, proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of computing, 1992, pp. 571-580.

[10] Y. Amir, B. Awerbuch, R. S. Borgstrom, *A Cost-Benefit Framework for Online Management of a Metacomputing System*, proceedings of the first International Conference on Information and Computation economies, 1998, pp. 140-147.

[11] A. Anastasiadi, S. Kapidakis, C. Nikolaou, J. Sairamesh, *A Computational Economy for Dynamic Load Balancing and Data Replication*, pp. 166-180 of the Proceedings in [10].

[12] The Globus Project home page: `http://www.globus.org`.

[13] J. Frey, T. Tannenbaum, I. Foster, M. Livny and S. Tuecke, *Condor-G: A Computation Management Agent for Multi-Institutional Grids*, in proceedings of The Tenth IEEE International Symposium on High Performance Distributed Computing, Aug. 2001, San Fransisco, CA.

[14] A. Silberschatz, P.B. Galvin, *Operating System Concepts*, Fifth Edition.

[15] The Condor project home page `http://www.cs.wisc.edu/condor/`.