

Non-shared disk cluster – a fault tolerant, commodity approach to high-bandwidth data analysis.*

E. Otoo^a, D. Olson^a, E. Hjort^a, J. Lauret^b, M. Messer^b, A. Shoshani^a, A. Sim^a,

^aLawrence Berkeley National Laboratory, Berkeley, CA, USA

^bBrookhaven National Laboratory, Upton, NY, USA

CHEP submission ID: 4-026

Abstract

The STAR experiment, in collaboration with the NERSC Scientific Data Management Group is prototyping / developing an approach to accomplish a high bandwidth data analysis capability using commodity components in a fault tolerant fashion. The prototype hardware consists of two small clusters of linux nodes (about 10 dual-CPU nodes), a few 100 GB local disk on each node. One cluster is at the RCF at Brookhaven and the other at PDSF at LBNL/NERSC. The local disk on each node is not exported on the network so that all processing of data occurs on processors with locally attached disk. A file catalog is used to track and manage the placement of data on the local disks and is also used to coordinate the processing with nodes having the requested data.

This paper describes the current status of this project as well as describe the development plans for a full scale implementation, consisting of 10s TB of disk capacity and more than 100 processors. Initial ideas for a full implementation include modifications to the HENP Grand Challenge software, STACS (<http://gizmo.lbl.gov/stacs>) that combines queries of the central tag database to define analysis tasks and a new component, a parallel job dispatcher, to split analysis tasks into multiple jobs submitted to nodes containing the requested data, or to nodes with space to stage some of the requested data. The system is fault tolerant to the extent that individual data nodes may fail without disturbing the processing on other nodes, and the failed node can be restored. Jobs interrupted on the failed node are restarted on other nodes with the necessary data.

Introduction

There are numerous models of how computing facilities provide access to data for the I/O intensive class of data analysis computing problems. In the network-data-access models the data analysis processes access data from disk files on a server over a LAN (or SAN)¹ using any of a number of types of network-file access software (NFS, RFIO, Objectivity/DB, ...). In the non-shared disk model described below a different approach is taken to achieve a relatively low cost and high bandwidth data analysis capability. A hardware configuration for this model is illustrated in figure 1.

The processing nodes of this cluster contain significant amounts of local disk and data is distributed across these disks. However, these local disks are not shared with other nodes so that all processing of data must take place on the node where the data is located. As in the shared disk model, the data may be staged to these local disks from tertiary storage servers. This model also has advantages and disadvantages relative to the shared disk model. In the non-shared model it is necessary to be able to coordinate the data placement/location with processing of the data and there are a large number of

* This work is supported by the US Department of Energy under contract nos. DE-AC03-76SF00098 and DE-AC02-98CH10886.

¹ local area network or storage area network

filesystems to be used which complicates the data management problem. One advantage of low cost of disk comes from populating the nodes with inexpensive IDE drives. However, this has the disadvantage that these disks (and nodes) are expected to be somewhat less reliable than compute-only nodes. Another principal advantage of the non-shared disk model is that the number of data analysis processes accessing a particular disk drive at any time is small because of the limited number of processors in these nodes (2 CPU/node). This is expected to contribute to a significant improvement in effective bandwidth capability for these drives since the random head motion from many independent processes accessing the drives is reduced or eliminated.

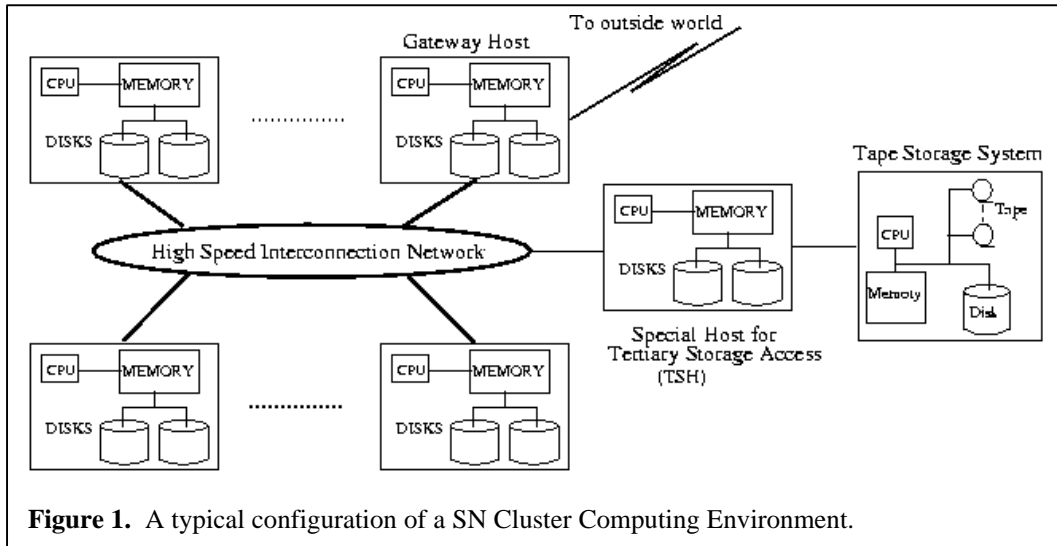


Figure 1. A typical configuration of a SN Cluster Computing Environment.

Current status

In order to achieve the coordination of processing with data location a simple non-scalable mechanism has been implemented at present while a more full featured, scalable and dynamic is planned for the future. First we describe the simple non-scalable mechanism and follow this with a description of the scalable and dynamic solution.

The data model used by the STAR experiment includes placing components of the same event into different files. However, at present, there is a one-to-one correspondence between the events represented in different component files, which is that the same set of events are represented in the different component files. This means that there is a trivial algorithm for identifying which files need to be accessed for a given set of events. With this model, it is easily arranged that all component files for a certain set of events can be placed on the same processing node local filesystem. While this simplifies the data management problem at present it is not a requirement of the system described below.

The simple processing model currently used on this cluster is to submit a single LSF batch job for each event file to be processed in a DataSet and a file-based catalog has the correspondence between files and nodes. A set of scripts takes a single command procedure for analyzing a file and generates as many batch submissions as there are files in a DataSet that get submitted to the appropriate node where the files reside. A minimal level of fault tolerance is employed currently in that the files on a single node are not correlated with any physics signals in a given DataSet so that if a node fails while an analysis is going on then the result on the physics is simply lower statistics but not a distortion of

the signal being studied. See <http://www-rnc.lbl.gov/PDSF/HB.html> for a description of the current user guide.

Planned implementation

Users generally specify jobs to be run through a cluster management software (CMS) package. Examples of such packages include Condor-G, SUN's Grid Engine software, Load Sharing Facility (LSF), and Portable Batch System (PBS). Our focus in this project is to develop coordinated storage and compute resource management that works cooperatively with such CMS packages. Our research includes analytical studies and prototype implementation of a resource management framework that coordinates both the computational resources and storage resources for processing data intensive computations on a shared-nothing cluster of workstations. The problem of identifying events based on user's selection criteria and subsequently mapping these to the correct files are addressed in a different part of this proposal. The major goals in this project are:

1. To develop a framework for a cluster computing resource management for SN-cluster of workstations.
2. Study analytically, different task scheduling policies based on data locality that optimize an objective function of the average job completion times.
3. Implement a prototype of the storage resource management that works cooperatively with cluster management software (CMS) to achieve optimal scheduling policies of tasks.
4. Develop and implement a mechanism for the fail-safe and recoverable execution of tasks as specialized distributed transactions on an SN-cluster of workstations.

Planned Architecture

We focus on the concurrent execution of a set of jobs. Each job J_i consists of a number of tasks, $J_i = \{T_{i,0}, T_{i,1}, \dots, T_{i,j}\}$. Each task $T_{i,k}$ has an affinity to a set of files $\{F_{i,k,0}, F_{i,k,1}, \dots, F_{i,k,t}\}$. A task, in executing on a host, requires that the files that it processes be resident on the same host. We assume that the disk capacity of a host can hold all the files required by a task. In the simplest case a task would require only one file. Figure 2 illustrates the typical configuration of hosts and the specific modules we propose to use to support the co-location of storage and compute resources.

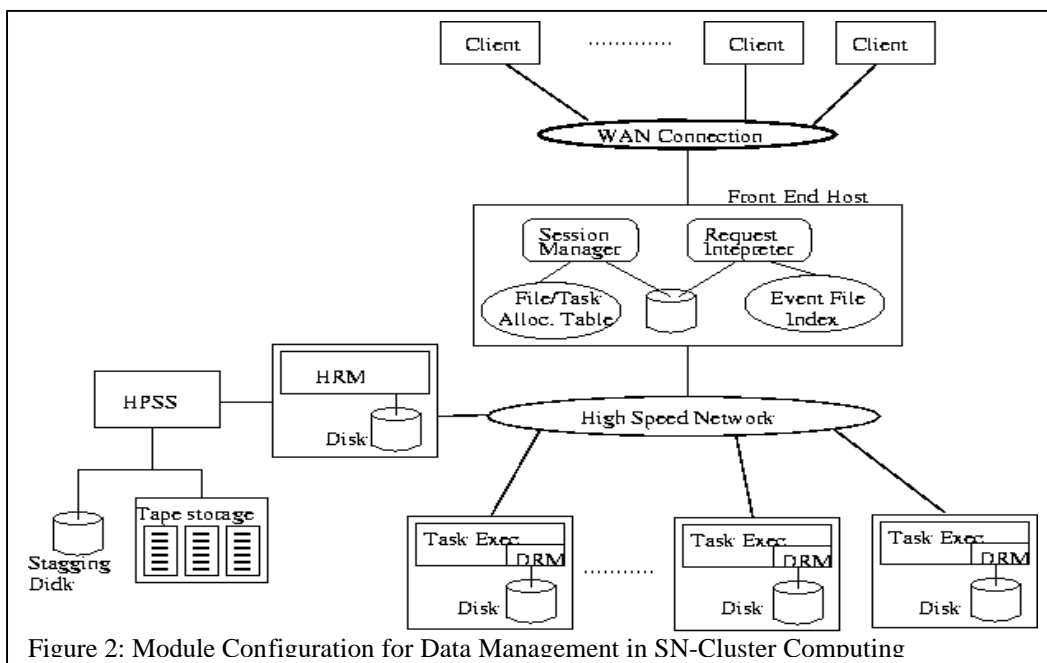


Figure 2: Module Configuration for Data Management in SN-Cluster Computing

The main idea of the architecture to support the co-location of storage and compute resources is to develop independent storage resource managers at each node. There are two types of prototype software modules we plan to implement: a Cluster-Computing Disk Resource Manager (ccDRM) for managing a disk resource at each host, and a Cluster-Computing Hierarchical Resource Manager (ccHRM) for managing a tape-disk resource at the single host that interacts with the mass storage system (we'll use HPSS since it is supported at our institution). These modules will be derived from the two modules that were developed in an earlier project, the Disk Resource Manager (DRM) and the Hierarchical Resource Manager (HRM) that coordinate file requests and transfers in a wide area network. In addition, we will implement a Session Manager (SM) that interfaces with the cluster management software (CMS). The SM will use the strategies we will develop for optimal course-grain parallel task execution in an SN-Cluster. These strategies will combine load-balancing algorithms, file placement algorithms and caching policies of the ccDRMs and ccHRM. In the next phase, we will develop fault-tolerance and recovery strategies for our prototype modules.

The SM will reside in a Front-End-Host. This host will also have a "request interpreter" which uses the bit-sliced index. The ccDRM will reside on each of the hosts of an SN-cluster. The SM communicates with the cluster management software (in this project we will use LSF), to schedule tasks optimally. The SM maintains a table, called the Global File Allocation Table (GFAT) that contains the combined information on the task and file allocations. The SM goal is to assign tasks to hosts to achieve load balancing and co-location of tasks with the files to be processed.

Each ccDRM manages its local disk independent of the others. It maintains a Local File Allocation Table (LFAT), that contains which files are cached to its disk. The updates to the LFATs (i.e. caching of file to or removing of files from the disk) are propagated to the GFAT. The action of the ccDRM is to ensure that if a file for a task to be scheduled already exists on its disk, that file is pinned. Otherwise, it has to allocate space for that file and request the file from the ccHRM or another ccDRM. The ccHRM is a special host of the cluster that interfaces with HPSS. Its functionality is similar to a ccDRM except that it manages the requests from HPSS. To implement ccDRM and ccHRM, we will take advantage of the implementations of a DRM and HRM previously developed. We will expedite the initial implementations of ccDRM and ccHRM using CORBA. CORBA has been successfully applied in parallel cluster computing where the hosts are heterogeneous with respect to the hardware architecture and data management controls. We plan to subsequently implement high performance ccDRM and ccHRM using MPICH.

Fault Tolerance and Recoverable Processing

Work on high performance computing has concentrated mainly on the development of equivalent parallel algorithms of their sequential counterparts for numerical applications. The applicability of cluster computing to data intensive applications has spurred some interest in the study of fault-tolerant issues in parallel computing. Implementers of MPI packages have only recently added features for application control of host failures [1]. We address fault-tolerance in our work from a data management point of view in order to sustain long transactions. The classical model of processing very long transactions is to partition it into sequences of short committable transactions using a nested transaction model [2]. Our approach is to wrap the task to be executed in a lightweight flat transaction model while the entire job is executed as a nested transaction. This approach addresses the problem of fault tolerance and recoverability.

To illustrate the direction of our work in fault tolerant computing, consider read-only files. A task $T_{i,j}$ constitutes a unit of a transaction. For simplicity, consider also that each transaction processes only one file $F_{i,j}$. A job is composed of multiple transactions and therefore the job commits only after

all its tasks, i.e., sub-transactions have committed. A job remains pending (but not aborted) if any of its sub-transactions aborts as a result of a crash of a host. A job aborts if all its tasks abort.

A task commits if it successfully processes all of its files and it aborts if it does not complete processing. The session manager that schedules tasks to be run on the hosts, maintains a journal for logging its activities. This includes information on when a task was delegated to a host, the host identifier, and a commit or an abort response from the host. We will assume that when a host fails it sends no messages and the session manager eventually detects that it has failed. There is an implicit *two-phase commit* from the point of view of the job as a nested transaction. Each task is run as an independent simple flat transaction.

Each ccDRM maintains a local log on an independent “*durable*” disk. The content of the log will include the entries described below. Each entry is preceded by a task or a transaction identifier:

Task_Begin: An indication of the start of the task.

Task_Abort: An indication of the abort of the task.

Task_Commit: A commit record of the task.

File_Caching_Begins: An indication of the start of caching a file.

File_Caching_Ends: An indication of the completion of file caching.

File_Purged: An indication of a purged file.

Checkpoint_Begin: A checkpoint begin record indicator.

Checkpoint_End: A checkpoint end record indicator.

Using the information in the log of a host, the system can recover quickly from soft-crashes of the host. A hard-crash that results in the loss of the host's local disk, is more complex. It involves analyzing the content of the session manager's journal. The front-end-host that runs a session-manager may be replicated there by allowing clients to access the cluster through any one of them. However, the GFAT and the journals would be replicated and synchronized on each of these front end hosts. We propose to explore other sophisticated methods of fault-tolerance and recovery methods and utilize them in the prototype implementations. In particular, we will further investigate the use of non-centralized non-blocking atomic commit protocols to control jobs that are run as long transactions and also involve update-in-place file operations.

Implementation Phases

Phase 1: Develop strategies for dynamic file allocation and load balancing the task execution within the hosts to obtain optimal file processing schedules.

Phase 2: Implement a prototype system that includes a Session Manager (SM) and the storage resource managers ccDRM and ccHRM that work cooperatively with cluster management software (CMS) to achieve optimal scheduling policies of tasks.

Phase 3: Develop and implement a protocols for the fail-safe and recoverable execution of tasks as specialized distributed transactions on an SN-cluster of workstations.

References

- [1] G. Fagg and a Dongarra. FT-MPI: Fault tolerant MPI, supporting dynamic applications in a dynamic world. In Euro PVM/MPI User's Group Meeting 2000, pages 346 -- 353, Springer-Verlag, Berlin, Germany, 2000.
- [2] J. Gray and A. Reuter. Transaction processing: Concepts and Techniques. Morgan Kaufmann, San Mateo, California, 1993.