

Techniques for Efficient Data Management in Data Grids

Pradeep Padala
Computer & Information Science & Engineering
University of Florida
Gainesville, Florida 32611-6120
Email: ppadala@cise.ufl.edu

Abstract—

I. INTRODUCTION

Grid[1] is becoming the favourite choice for executing data intensive and scientific applications. These applications often analyze and derive large amounts of data (some times of the order of peta bytes). Existing grid infrastructures provide basic capabilities to deal with the data. To efficiently utilize the dynamic resources, sophisticated scheduling mechanisms are needed. Sphinx[2] is a scheduling middleware that provides various features for scheduling jobs on a grid with dynamic resources.

One of the important aspects of Sphinx is the efficient management of data for overall optimal workflow. The dynamic nature of resources and jobs pose a significant challenge in achieving this.

In this paper, we first review the requirements for efficient data management. Then, we review existing techniques and their significance. Next, we describe a data management architecture for Sphinx. We conclude by showing experiments using the data management component in Sphinx.

II. PREVIOUS WORK

Due to distributed nature of data on the grid, various mechanisms have been developed for managing the data. In this section, we provide a brief analysis of previous work. A complete survey of data middleware is available in the appendix.

Fill Analysis

Many projects using grid have come up with various solutions for data management. The emphasis of these data middleware has been providing a framework for data management. Little work has been done in understanding the various issues involved in scheduling jobs. Few experiments are done in understanding the

overheads involved in data management. Existing experimental data is specific to the application and is often produced without considering other jobs in the grid.

Our approach is to build a framework that can be used by various projects and can be used to evaluate the performance of data management. We provide some preliminary experiments showing overheads in data management.

III. REQUIREMENTS

The complete requirements for a scheduling middleware are discussed in Sphinx architecture paper[2]. In this section, we provide more detailed analysis of data management requirements.

- *Optimal Transfer of Data:* The primary requirement for the middleware is to provide means for transferring the data optimally. In scientific applications, data is stored in tertiary storage at a central place. Scientists who would like to work with the data, make copies of parts of the data. These copies are called disk caches. To analyze large amounts of data, the disk caches are distributed locally. The job of the middleware is to efficiently manage these mechanisms and provide an optimal workflow.
- *Uniform Access to Underlying Data:* Due to heterogeneous nature of the grid, data may be stored in different formats in VOs. The data may even be stored in different formats locally for efficiency. The middleware has to provide uniform access to these heterogeneous formats.
- *Replica Management:* Due to heterogeneous nature Replicas are maintained for fault tolerance and for finding data quickly. This requirement effects the 'optimal transfer' requirement above.
- *Meta data Management:* Meta data is the data about the data. There are various kinds of meta data including

- File Level Meta data: size of the file, creation, modification and access time, creator etc.
- Storage meta data: storage type, size of the storage, duration available etc.
- Access control meta data: Access control lists by user, group per file
- Descriptive and Provenance meta data

A data management middleware should provide basic meta data management facilities and support for at least file level meta data. It should also support mechanisms through which additional meta data can be managed.

IV. ARCHITECTURE

A. *Sphinx Architecture*

brief sphinx architecture description

B. *Data Management Component(DMC)*

The DMC consists of four components

- *Dag Reducer*: The DAG reducer reads new, incoming DAGs from the tracking/prediction database and eliminates previously completed jobs. Such jobs can be identified with the use of a replica catalog. The DAG reducer simply checks for the existence of the output files ¹. of each job, and if they all exist, the job and all child jobs can be deleted ². This module simply operates on any DAG which has state unreduced and changes the DAG state to unpredicted. These pruned DAGs are then written back into the tracking database for further processing.
- *Data Planner*: The data planner provides the intelligence for optimal data management. It reads the DAGs and makes decisions regarding required replication jobs. It adds replication jobs as required to the DAG. The replication jobs are added based on heuristics for optimal data management.
- *Automatic Replication*: This component provides smart data replication with out considering the current/future job distribution. It provides efficient hot spot management. A hot spot is identified by looking at the file access patterns. Every time a file

¹It is assumed that the virtual data and replica catalogues provide globally unique identifiers (or possibly unique name spaces) for data products

²This assumes that the existence of data from an earlier completed job implies that that earlier job has also been optimally completed for the current DAG. There may be cases where it is more cost effective to neither co-locate the job with the pre-existing data nor to transport the data over the network to the compute site, but rather to re-create the data on demand at another location

is touched by a job, DMC keeps file access pattern information (**more info**). Once a hot spot has been identified, an appropriate replication site is chosen depending on existing policies.

- *Uniform Data Middleware Interface*: This component provides a uniform interface to existing data middleware. Currently, it provides interfaces to data transfer and replica services.

V. TECHNIQUES FOR OPTIMAL DATA MANAGEMENT

VI. EXPERIMENTS

APPENDIX

*Survey of Data Middleware A grid middleware provides facilities to use the grid for applications and users. Middleware like Globus[3], Legion[4] and UNICORE[5] provide software infrastructure to tackle various challenges of computational and data grids. A *data middleware*, which usually is part of general purpose grid middleware, provides facilities for data management.

A data middleware provides mechanisms and software infrastructure for data management. A storage middleware or low-level data middleware allow storage resource management in heterogenous resource environment.

We have surveyed various projects that developed data middleware and came up with a set of common features that are available in data middleware. In the following sections, we provide a detailed list of these services.

A. *Universal Name Space*

Since data is stored in different name spaces in the grid, it is essential to have a universal name space encompassing underlying naming mechanisms. In a data grid, data might be stored in a file, collection of files or in a database table. They may be replicated in various sites for efficiency. A universal global name space provides a heirarchical logical name space to access these physical names. For example, an LFN (Logical File Name) used by Giggle[6] translates into PFN (Physical File Name)s, which point to replicas of the file. The logical organization usually closely follows the physical hierarchy or grid topology but not necessarily.

B. *Latency Management*

Due to distributed nature of data, it is necessary to access the data quickly without much latency. Data middleware should provide mechanisms to access the data quickly. Replication is one of the common methods to reduce latency. Replication also increases availability of the data.

1) *Replication*: Replication is the process of maintaining identical copies of data in various sites. The main purpose of replication is to maintain copies of the data in nearby sites thus increasing read performance. But, it introduces other issues like consistency management and life cycle management. Pre-fetching and disk caching can be thought of as a form of replication. In pre-fetching, a grid scheduler knowing the execution site of a job, may pre-fetch the data required by the job to that site. Disk caching refers to replication at a local level. A mass storage system in a site may maintain local “cached” copies of data for efficiency purposes.

Replication introduces various synchronization or consistency issues for the data. Replicas maintained at different sites may have to be maintain various levels of consistency. If there is no consistency among replicas of data, it is equivalent to copying of files to various places. On the other hand, data middleware may maintain a strong read/write consistency for the replicas. In this scenario, all the replicas are kept by propagation of writes. This required locking mechanisms over the grid and may induce high performance penalty. Usually, data middleware provide relaxed replication schemes like read-only replicas. More information on consistency models can be found in [7].

Data middleware should be able to support different replication mechanisms depending on the policies of different virtual organizations. These mechanisms can be used by administrators and task schedulers to perform specific actions like creation and purging of replicas depending on space constraints.

Replication of metadata along with the data creates interesting and complex problems. Consistency of metadata is an important concern in a data grid.

Adaptive replication depending on access patterns of data is an interesting concept that is still to be explored in data grids. An intelligent data middleware can keep track of access patterns and may create or purge replicas automatically. For example, if one class of applications access the data on a site (called hotspot) lots of times, an intelligent middleware may decide to replicate the data in various sites.

2) *I/O Command Aggregation*:

3) *Aggregation of Data*:

C. Data Transfer

A data middleware should provide mechanisms for transferring the data. Currently, various data middleware provide data access using low-level file transfer mechanisms like GridFTP[8]. Since there might be different

underlying mechanisms to transfer data, the middleware should provide a uniform mechanism to transfer data. SRB (Storage Resource Broker) [9] provides a uniform storage interface for underlying storage provide not only uniform transfer mechanisms but also uniform data access.

D. Data Access APIs

Data middleware should provide APIs for data access, transfer and discovery. At the minimum, it should provide UNIX file system semantics for accessing the data. Mechanisms for data discovery in the grid are still explored and various mechanisms like MDS[10] (Monitoring and Discovery Service) are proposed for data discovery.

E. Metadata Management

Metadata is the data about data. Metadata in a grid can be broadly divided into technical metadata and contextual metadata[?] **data requirements for the grid, scoping study report citation**. Technical metadata includes information about location, size and other data resource characteristics. Contextual metadata refers to derivation and provenance information about the data.

A data middleware is required to maintain metadata. **rephrase !!**

F. Security

Though grid security is not directly controlled by data middleware, certain aspects play an important role in data security. Since disk caches and replicas are maintained at various sites having differing security policies, security of the data may be compromised. The owner of the data may also want to specify a certain level of security which may not be able to be enforced at particular sites.

Sensitivity levels for the data and the meta data may be different and may require different security mechanisms.

The owner of the data or administrator of a site should be able to specify access privileges for the data. The access control mechanisms should be flexible enough to specify control at object, file, user and site level.

REFERENCES

- [1] I. Foster and C. Kesselman, Eds., *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.
- [2] J. U. In, A. Arbree, P. Avery, R. Cavanaugh, S. Kategeri, and S. Ranka, “Sphinx: A scheduling middleware for data intensive applications on a grid,” May 2003.

- [3] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, no. 2, pp. 115–128, 1997.
- [4] A. S. Grimshaw, W. A. Wulf, and the Legion team, "The legion vision of a worldwide virtual computer," *Communications of the ACM*, vol. 40, no. 1, pp. 39–45, Jan. 1997.
- [5] V. Huber, "UNICORE: A Grid computing environment for distributed and parallel computing," *Lecture Notes in Computer Science*, vol. 2127, pp. 258–266, 2001.
- [6] A. L. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney, "Giggle: A framework for constructing scalable replica location services," in *SC'2002 Conference CD*. Baltimore, MD: IEEE/ACM SIGARCH, Nov. 2002, pap239.
- [7] D. Dillmann, W. Hoschek, J. Jaen-Martinez, B. Segal, H. Stockinger, K. Stockinger, and A. Samar, "Models for replica synchronisation and consistency in a data grid," in *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC)*, San Francisco, California, August 2001, pp. 67–75.
- [8] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, "Data management and transfer in high-performance computational grid environments," *Parallel Computing*, vol. 28, no. 5, pp. 749–771, May 2002.
- [9] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The sdsc storage resource broker," in *Proceedings of IBM Centers for Advanced Studies Conference*. IBM, 1998.
- [10] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, "A directory service for configuring high-performance distributed computations," in *Proc. 6th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 1997, pp. 365–375.