SOME REMARKS ON PARALLEL DATA MINING USING A PERSISTENT OBJECT MANAGER

Neil Araujo, Robert Grossman, David Hanley, and Wen Xu
University of Illinois at Chicago, USA
Seung Ahn, Kirill Denisenko, Mark Fischler, Mark Galli
Fermi National Accelerator Laboratory, USA
David Malon and Edward May
Argonne National Laboratory IL, USA

Abstract

Our underlying assumption is that high performance data management will be as important as high performance computing by the beginning of the next millennium. Given this, data mining will take on increasing importance. In this paper, we discuss our experience with parallel data mining on an IBM SP-2, focusing on four issues which we feel are are emerging as critical for data mining applications in general.

1 Introduction

We feel that as the next millennium approaches high performance data management will assume a role as important as high performance computing. In this note we describe some of our work developing a system which exploits high performance data management to aid in the statistical analysis of large amounts of high energy physics event data.

In the first part of the paper, we describe four issues which arose in our work with parallel data mining on an IBM SP-2, focusing on issues which we feel are are emerging as critical for data mining applications in general. In the second part, we briefly describe the design and architecture of the system.

This paper is a summary of the longer report[1] which has been submitted for publication elsewhere.

Lightweight Data Management. Since data mining applications read data often but write data rarely, the full functionality of a database is not needed. As an alternative, we view the data as a collection of persistent objects and use lightweight software tools to create, access, manage, and query it. This has the important added advantage that such a lightweight data management system can be more easily parallelized and tuned than can a full-function database.

Balancing ***Parallel Input-Output and Parallel Execution.*** Our study used the Computing for Analysis Project (CAP) IBM SP-2 at Fermilab, which consisted of twenty four nodes interconnected by a high speed communications switch. Eight of the twenty four nodes of the IBM SP-2 were configured as input-output nodes, with approximately ten Gigabytes of disk attached to each one. The other sixteen nodes were designated as Query Nodes. Query Servers ran simultaneously on each of the Query Nodes, and each Query Node could request data from any of the input-output nodes, allowing each of them to obtain, in principle, an eightfold speed up in the delivery of data. While

some data mining applications can effectively exploit complete data parallelism, with data analyzed locally on each node and no data communication necessary, for others such as ours, it is important to balance data-centered parallelism with parallel input-output.

Data Placement. We developed an object oriented data model which logically viewed the data as a collection of events whose attributes were the banks of the underlying data. There are at least three ways of physically clustering the data on disk: by event, by attribute, or by some hybrid combination of event and attribute. An initial understanding was gained of the trade-offs for these different clustering strategies.

Resource Management. We were caught unawares by the amount of effort that we had to devote to resource management. For example, even with only 24 nodes, a more flexible scheme was needed to dynamically reassign the function of a node, for example, from a Query Node to an input-output node. As another example, our multiple Gigabyte data sets were divided into tens of thousands of physical collections, called folios. The folios were stored on disks attached to the input-output nodes, but their location would change from time to time. In retrospect, a more efficient and flexible scheme for tracking the physical location of the folios was needed. As a third example, the index sets for the data sets were too large to fit on the disks of the Query Nodes and they themselves had to be managed. However, the object storage model we used provides a natural mechanism to manage these index sets in the same manner as data sets.

2 Design

2. I Data Model

By an ***object store we*** mean a collection of objects that persist or continue to exist beyond the lifetime of the process which creates them. Objects have attributes and methods associated with them. An object store is created by a process called population, which is a method of reading raw data and creating persistent objects which exist on disk. During population, each object is assigned a Persistent Object ID or PID. The Persistent Object ID is guaranteed to be unique within each store.

Objects and attributes provide a *logical* description of entities. Perhaps the most fundamental database tenet is to separate the logical description of data from the details of its physical storage. Our system architecture is based upon two abstractions related to the *physical* organization of the data, which we now describe.

As in the previous work[2,4], we assume that the stores are organized as a physical collection of objects called *folios*, which in turn are a physical collection of smaller-grained collections of objects which are called *segments*. A segment may contain one or more objects. Folios and segments are abstractions, but relate to the physical organization of the data, just as objects and attributes are abstractions relating to the logical organization of the data.

The essential point here is that the physical management of the data is hierarchical: there is simply too much data present to manage it simply on the basis of a single construct such as segments. Roughly speaking, folios represent physical collections which are managed from disk to disk or from

disk to tertiary storage, while segments represent subcollections which are managed from disk to disk or from disk to virtual memory.

Another essential feature of our data model is to distinguish between two types of attributes. It is a characteristic of data mining applications that they can have so much data that not all of it fits on secondary storage or that even for the data on secondary storage, some of it is stored on devices with a higher latency. For this reason it is useful to distinguish between data which is accessed more frequently.

An attribute is called *cardinal* in case it is the basis of sufficiently many queries that physical folios and segments associated with it should be on secondary storage (disks). An attribute is *auxiliary* if it can safely be stored on tertiary storage devices or devices with higher latency. An attribute need be of neither type.

2.2 *Network Architecture*

Network Model. Our network model consists of a collection of nodes connected by a high performance network. The nodes were of two types: 1/0 Nodes and Query Nodes. In our model, the 1/0 Nodes are connected to large disks and/or tertiary storage and run processes which provide requested segments to the Query Nodes. The Query Nodes run processes which provide objects to applications. We assume that the Query Nodes and 1/0 Nodes communicate through a message passing protocol.

I/O Nodes. The 1/0 nodes serve as data servers. Each node is locally attached to a number of SCSI disks. We assume that the object stores are physically distributed over these nodes. The 1/0 nodes function exclusively as data servers. No queries run on this node.

Query Nodes. The Query Nodes are the nodes on which the actual queries run. These nodes have a limited amount of local disk space.

Message Passing ***Interface.*** The architecture requires that nodes on a network to pass messages and data to each other. For the initial design of the system, we used the P4 parallel programming system to implement this message passing[5]. P4 is a library of macros and subroutines developed at Argonne National Laboratory for, among other things, message-passing in the distributed-memory model. P4 is designed to be used to program networks of workstations as well as advanced parallel supercomputers. P4 supports a wide variety of network protocols specifically on the IBM SP-1, on which our system was designed, like TCP/Ethernet, TCP/switch, EUI, and EUI-H.

2.3 *Object Store Architecture*

Query Client. A Query Client is a process which needs to access some or all of the objects in a store. A typical query makes requests for objects from the store and process these objects for further analysis. If the segment containing a referenced persistent object is not currently available in virtual memory, a fault is generated to the Query Server, which requests the desired segment from the (appropriate) Segment Server.

Query Servers. A Query Server is a program that runs continuously on each network node on which a Query Client can run. A Query Server services

requests from Query Clients running on the same node. A Query Server can service multiple Query Clients concurrently. The Query Server forwards requests made by the Query Clients to the appropriate Segment Servers on the network.

Segment Servers. A segment server is a program that runs continuously on each network node on which an object store resides. The segment server services requests from Query Servers on other network nodes for segments that reside on disks attached to the node it runs on. The segment server uses a simple pre-fetching algorithm to cache segments.

3 Implementation

We choose an IBM SP-1 as the platform for our experiments, which at the time we are writing this paper has been upgraded to an IBM SP-2. The SP system we used had two types of nodes: 1/0 nodes which each had approximately ten gigabytes of attached disk and Query Nodes which had a minimal amount of attached disk. The data used in these tests was obtained the D-zero experiment at the Fermilab Collider. An object data model was developed which mapped the original Zebra formatted data into C++ classes. The cardinal data was stored on 1/0 nodes, the auxiliary data on attached hierarchical storage. The Query Nodes were used to execute the queries. With this approach, both CPU requirements and 1/0 requirements are scalable.

PTool. To provide the light weight object management for this project, we choose PTool[3],[2]. PTool provides persistence for instances of C++ classes through an application program interface (API) and class libraries. Persistent objects are grouped together into persistent container classes provided by the class libraries.

The physical design of version 0.6 of PTool is based upon three concepts: segments, folios and stores. A ***segment*** is a continuous range of virtual memory that is managed by PTool. A segment may contain one or more objects. A ***folio*** is a physical collection of segments. Both segments and folios are managed by PTool as files. A ***store*** is a physical collection of folios.

Query Language. Since an Object Query Language (OQL) Interpreter was not available for the IBM SP-2, an application called MQL was written. MQL parses a limited subset of OQL and produces a C++ program with the appropriate PTool API. For familiarity to the physicists, the syntax was chosen to be close to a de facto standard in the high energy physics community called PAW[6]

Resource Management. A two hundred Gigabyte event store requires managing approximately ten thousand megabyte size folios. With so many folios, the physical location (which node there are located on) is apt to change. This part of the resource management, as well as related issues, such as moving nodes from Query to 1/0 nodes, turned out to be a significant part of the implementation effort.

4 Status and Future Work

The work described above provided an initial implementation and testbed for this approach. The approach appears sufficiently promising that the CAP

Project at FNAL is currently developing a second version of the system, including a light weight persistent object manager called POPM, which is specialized for the IBM SP-2. The UIC group is currently developing versions of PToo1 which are specialized for high performance, local, and wide area clusters of workstations, as well as started work on general techniques for resource management needs for high performance data management in clustered computing environments.

Acknowledgments and for Additional Information

The UIC portion of the research was supported in part by NASA grant NAG2-513, DOE grant DE-FG02-92ER25133, and NSF grants IRI 9224605 and CDA-9303433, and the National Scalable Cluster Project. The FNAL portion waa supported by the DOE and the Computing for Analysis Project (CAP).

For additional information, contact R. Grossman, Laboratory for Advanced Computing, University of Illinois at Chicago, 851 S. Morgan Street, Chicago, IL 60607, USA, grossman@uic.edu.

1. N. Araujo, K. Denisenko, M. Fischler, M. Galli, R. Grossman, D. Hanley, D. Malon, E. May, and W. Xu, "Parallel Data Mining of High Energy Physics Data Using a Persistent Object Manager: A Case Study" ***Laboratory for Advanced Computing Technical Report Number 95-R14***, University of Illinois at Chicago, 1995, submitted for publication.
2. R. L. Grossman and X. Qin, "Ptool: a scalable persistent object manager," ***Proceedings of SIGMOD*** 94, ACM, 1994, page 510.
3. R. L. Grossman, N. Araujo, X. Qin, and W. Xu, "Managing physical folios of objects between nodes," ***Persistent Object Systems (Proceedings of the Sixth International Workshop on Persistent Object Systems),*** M. P. Atkinson, V. Benzaken and D. Maier, editors, Springer-Verlag and British Computer Society, 1995.
4. R. L. Grossman D. Hanley, and X. Qin "Caching and migration for physical collections of objects: Interfacing persistent object stores and hierarchical storage systems," in ***Proceedings of the 14th IEEE Computer Society Mass Storage Systems*** Symposium, S. Coleman, editor, IEEE, **1995,** to appear.
5. R. Butler and E. Lusk, "User's Guide to the p4 Parallel Programming System," Technical Report ANL-92/17, Argonne National Laboratory, October 1992 (revised 1994).
6. R. Brun, O. Couet, C. Vandoni, and P. Zanarini, "PAW Users Guide," Program Library Q121, CERN, 1991.