A High Performance Implementation of the Data Space Transfer Protocol (DSTP)

S. Bailey, E. Creel, R Grossman[*], S. Gutti, and H. Sivakumar
National Center for Data Mining (M/C 249)
851 S. Morgan Street
University of Illinois at Chicago
Chicago, IL 60607

June, 1999

This is a draft of a paper which later appeared in *Large-Scale Parallel Data Mining*, M. J. Zaki and C.-T. Ho, editors, Springer-Verlag, Berlin, 2000, pages 55-64.

Abstract

With the emergence of high performance networks, clusters of workstations can now be connected by commodity networks (meta-clusters) or high speed networks (super-clusters) such as the very high speed Backbone Network Service (vBNS) or Internet2's Abilene. Distributed clusters are enabling a new class of data mining applications in which large amounts of data can be transferred using high performance networks and statistically and numerically intensive computations can be done using clusters of workstations.

In this paper, we briefly describe a protocol called the Data Space Transfer Protocol (DSTP) for distributed data mining. With high performance networks, it becomes possible to move large amounts of data for certain queries when necessary. This paper describes the design of a high performance DSTP data server called Osiris which is designed to efficiently satisfy data requests for distributed data mining queries. In particular, we describe 1) Osiris's ability to lay out data by row or by column, 2) a scheduler intended to handle requests using standard network links and requests using

[*]Point of Contact: grossman@uic.edu, http://www.ncdm.uic.edu. R. Grossman is also with Magnify, Inc.

network links enjoying some type of premium service, and 3) a mechanism designed to hide latency.

1 Introduction

In this paper we consider some of the issues that arise in distributed data mining when large amounts of data are moved between sites. One of the fundamental trade-offs in distributed data mining is between the cost of computation and the accuracy of results. We assume: 1) that there is a cost for moving data between sites, and 2) that the most accurate model is obtained by moving all the data to a single site. Leaving some or all of the data in place, building local models, and merging the resulting models, produces a model which is less accurate, but which, in general, is also less expensive to compute.

The cost of moving data to a central location with the commodity Internet has tended to produce either distributed data mining systems which build local classifiers and then combine them or data mining systems that use standard interfaces such as ODBC or JDBC. These protocols work best when moving relatively small amounts of data to a central location. Examples of the former include JAM [18] and BODHI [15]; examples of the latter include Kensington [11].

In a previous paper [22], we have pointed out that there are many intermediate cases in which building classifiers that are close to the optimal one results in moving some of the data, leaving some of the data in place, building local classifiers, and combining them. In this paper, we are concerned with the design of network protocols and middle-ware for distributed data mining systems which have the ability to move some data and to leave other data in place. For example, Papyrus [7] is a distributed data mining system of this type.

Three fundamental challenges faced by distributed data mining systems are:

> *Problem A.* How can the analysis of distributed data be simplified?
>
> *Problem B.* How can the amount of data per site be scaled?
>
> *Problem C.* How can the number of sites be scaled?

To address Problem A, we introduced a protocol called the Data Space Transfer Protocol (DSTP) [1]. In this paper, we are concerned with how we can design DSTP data servers for distributed data mining which scale up as the amount of data per site increases (Problem B). We describe a high performance DSTP server we are designing called Osiris, which is a component of a distributed and high performance data mining system we are building called Papyrus [7].

One method of satisfying the computing and i/o requirements for high performance data mining is to use clusters of workstations [7] [16] [19] — *compute clusters* to satisfy the CPU requirements and *data clusters* to satisfy the i/o requirements. With the recent advances in high performance networks, geographically distributed clusters of workstations can be connected not only with commodity networks but also with high performance networks such as the NSF vBNS Network supported by MCI and the Internet2 Abilene Network supported by Qwest. For example, for the distributed data mining tests reported below, we used a data cluster in Chicago connected to a compute cluster in Washington, D.C. over a DS-3 link running at 45 Mb/s. Our first DSTP implementation provided approximately 3 Mb/s of throughput, while our current implementation provides approximately 30 Mb/s of throughput, a 10x improvement. See Table 1.

Based upon our previous experience analyzing the performance of another distributed data mining system we built [10], we decided to focus on three questions:

What do we store? More precisely, how should we physically layout the data on the server? By row or by column? Can we precompute intermediate quantities to speed up queries?

What do we move? More precisely, to what extent should data or meta-data be moved from node to node? There are several possibilities: we can move data, we can move predictive models, or we can move the results of computations. If we decide to move data, we can move data by table, by row, or by column.

How do we move it? What application protocol should be used for moving data in data space? How can data be moved in parallel between nodes? How can QoS be exploited to improve data transport? What is the effect of latency on data mining queries? What transport protocol should we use? Given multiple requests to a data server, how should the requests be scheduled?

The 10x performance gain we mentioned above resulted from careful understanding of these issues. Section 2 describes related work and background material. Section 3 describes data space and the data space transfer protocol. Section 4 describes the DSTP server and three experimental studies. Section 5 is the conclusion and summary.

2 Background and Related Work

In this section, we provide some background material and discuss some of the related work in this area. With the exception of [19] and [16], the work we know of in this area is limited to data mining over commodity networks. This section is adapted from [8].

Several systems have been developed for distributed data mining. Perhaps the most mature are: the JAM system developed by Stolfo et. al. [18], the Kensington system developed by Guo et. al. [11], and BODHI developed by Kargupta et. al. [15]. These systems differ in several ways:

Data strategy. Distributed data mining can choose to move data, to move intermediate results, to move predictive models, or to move the final results of a data mining algorithm. Distributed data mining systems which employ *local learning* build models at each site and move the models to a central location. Systems which employ *centralized learning* move the data to a central location for model building. Systems can also employ *hybrid learning*, that is, strategies which combine local and centralized learning. JAM and BODHI both employ local learning while Kensington implements a centralized approach using standard protocols such as JDBC to move data over the commodity Internet.

Task strategy. Distributed data mining systems can choose to coordinate a data mining algorithm over several sites or to apply data mining algorithms independently at each site. With *independent learning*, data mining algorithms are applied to each site independently. With *coordinated learning,* one (or more) sites coordinate the tasks within a data mining algorithm across several sites.

Model Strategy. Several different methods have been employed for combining predictive models built at different sites. The simplest, most common method is to use *voting* and combine the outputs of the various models with a majority vote [4]. *Meta-learning* combines several models by building a separate meta-model whose inputs are the outputs of the various models and whose output is the desired outcome [18]. *Knowledge probing* considers learning from a black box viewpoint and creates an overall model by examining the input and the outputs to the various models, as well as the desired output [11]. Multiple models, or what are often called ensembles or committees of models, have been used for quite a while in (centralized) data mining. A variety of methods have been studied for combining models in an ensemble, including Bayesian model averaging and model selection [17], partition learning [6], and other statistical methods, such as mixture of experts [23]. JAM employs meta-learning, while Kensington employs knowledge probing.

Papyrus is designed to support different data, task and model strategies. For example, in contrast to JAM, Papyrus can not only move models from node to node, but can also move data from node to node, when that strategy is desired. In contrast to BODHI, Papyrus is built over a data warehousing layer which can move data over both commodity and high performance networks. Also, Papyrus is a specialized system which is designed for clusters, meta-clusters, and super-clusters, while JAM, Kensington and BODHI are designed for mining data distributed over the Internet.

Moore [16] stresses the importance of developing an appropriate storage and archival infrastructure for high performance data mining and discusses work in this area. The distributed data mining system developed by Subramonian and Parthasarathy [19] is designed to work with clusters of SMP workstations and like Papyrus is designed to exploit clusters of workstations. Both this system and Papyrus are designed around data clusters and compute clusters. Papyrus also explicitly supports clusters of clusters and clusters connected with different types of networks.

3 Data Space and the Data Space Transfer Protocol

We begin by describing some of the key concepts following [1].

Data Space. We assume that data is distributed over nodes in a global network, which we call a *data space.*

Rows and Columns. Although the data may be more complicated, we assume that the data is organized into tables, and that each table is organized into rows (records) and columns (observations). Records may contain missing values.

Catalog Files. Each DSTP server has a special file called the *catalog file* containing meta-data about the data sets on the server.

Horizontal Store: Store Size = 4.4 GB				
NC	ADR in Mb/s	TDT in Giga bytes	TTT in seconds	EPR in Events/second
1	3.06	4.4	11777.5	64
2	6.07	4.4	5926.59	253
4	10.05	4.4	3590.40	655
8	16.92	4.4	2132.05	2811
16	23.32	4.4	1550.91	7731
32	34.93	4.4	1032.24	23245

Vertical Store: Store Size = 4.0 GB				
NC	ADR in Mb/s	TDT in Mega bytes	TTT in seconds	EPR in Events/second
1	1.39	269.5	1549.05	400
2	2.75	269.5	797.00	1554
4	3.81	269.5	566.42	4377
8	6.75	269.5	320.45	15482
16	9.74	269.5	223.05	44590
32	13.96	269.5	152.52	126918

Table 1: Performance analysis of horizontal vs. vertical stores.
NC - Number of clients requesting data
ADR - Aggregate Data Rate
TDT - Total Data Transferred
TTT - Total Time Taken for completion of application
EPR - Events Processing Rate

DSTP. We assume that there is a server on each node which can move data to other nodes using a protocol called the *data space transfer protocol (DSTP).* Depending upon the request, DSTP servers may return one or more columns, one or more rows, or entire tables. DSTP servers can also return meta-data about tables and the data they contain.

Universal Correlation Keys. A row may have one or more keys. Certain keys called *Universal Correlation Keys* (UCK) are used for relating data on two different DSTP servers. For example, key-value pairs (k_i, x_i) on DSTP Server 1 can be combined with key-value pairs (k_j, y_j) on DSTP Server 2 to produce a table (x_k, y_k) in a DSTP client. The DSTP client can then find a function $y = f(x)$ relating x and y.

Since DSTP client applications need only collect meta-data from the catalog files and need only move the relevant columns, these type of applications tend to scale better as the number of sites increases (Problem C) than distributed data mining applications which must move entire files. Recall that we are interested in the case in which some data is moved. Of course, if sufficient accuracy can be obtained by local analysis followed by combining models, this is usually less expensive than strategies which require that data be moved.

Notice that from this perspective, distributed databases are concerned with the efficient *updates* of distributed *rows*, while distributed data mining applications are concerned with the efficient *reading* and analysis of distributed *columns.*

In the next section, we describe our efforts to produce DSTP servers which can efficiently manage large data sets.

4 The Osiris DSTP Server

Osiris is a high performance DSTP Server which is designed to provide efficient read access to data. In our design, efficient read access is delivered by implementing high performance storage support, high performance network transfer support, and differentiated service support.

In this section we discuss our implementations of these support mechanisms and some preliminary experimental results which attempt to quantify the relative performance gains for each technique. All three mechanisms are implemented in process space and do not require any special tuning of the underlying hardware or operating systems. We felt it was important to provide performance improvements that were independent of the underlying system in order to increase portability.

4.1 Rows and Columns

Tabular data may be laid out on disk by row or by column. Since data from disk is transfered by block, certain queries will be more efficient when the data is laid out by row (*horizontally*) and other queries will be more efficient when the data is laid out by column (*vertically*).

DSTP client applications accessing data may request either rows of data or columns of data. If a column of data is requested and the underlying storage layout is *horizontal*, then each block will contain quite a bit of unwanted data. The same is true if a row of data is requested and underlying layout is *vertical*.

Since horizontal layouts speed up certain distributed data mining queries and vertical layouts speed up others, Osiris stores data in both formats. Although this doubles the amount of space required, the I/O traffic is reduced significantly. Since Osiris is a distributed system, the I/O traffic ultimately passes through a network communication link. Since network bandwidth is sufficiently more expensive than disk capacity, we feel the 2X increase in required storage is more than compensated for.

The following results are from a proof of concept DSTP data server located at the University of Illinois at Chicago being accessed by multiple clients located at an Internet2 member facility in Washington, D.C. called Highway One. The two sites are connected by the NSF/MCI vBNS Network. Even though vBNS is an OC-3 network offering maximum bandwidth of 155 Mb/s, the end nodes at Highway One were connected via a DS-3 link, which limited the maximum bandwidth of the testbed to 45 Mb/s.

For this test, we used an application benchmark we developed called the *Event Benchmark*, which is broadly derived from high energy physics. The data consists of a large collection of *events*, with each event containing several hundred attributes. An energy like function is computed from the attributes of an event and the energies of the event are histogrammed.

To better understand quantitative effects of the Horizontal/Vertical Layout strategy, we first laid out the data *horizontally* and ran the application, and then we laid out the data *vertically* and ran the application.

In the first case, all the event data was stored as rows (i.e., each event was a row). In the second case, the event data was stored attribute by attribute as columns. The *Event Benchmark* specifies that event level summary data is

NC	ART-P in seconds	ART-PS in seconds	ART-C in seconds	ART-CS in seconds
1	606.3	570.4	422.1	447.4
2	577.5	557.5	445	463.5
3	574.4	558.5	568.5	581.3
4	566.6	566.5	715	740
5	565.9	562.16	880.9	892.7

Table 2: Performance of Diff-Serv scheduler
NC - Number of clients requesting data per service type
ART-P - Average run-time for premium clients (no scheduling)
ART-PS - Average run-time for premium clients (with Diff-Serv scheduling)
ART-C - Average run-time for commodity clients (no scheduling)
ART-CS - Average run-time for commodity clients (with Diff-Serv scheduling)

to be stored separately and analyzed at run-time to find out which attributes are to be requested and processed. In other words, this particular application requests columns of data based on some criteria. Therefore, we expected that a *vertical* layout should provide better performance.

Table 1 shows the performance results. The Event Processing Rate (EPR) is an an application benchmark of efficiency. Aggregate Data Rate (ADR) and Total Data Transferred (TDT) are system performance measures. The desired result is to maximize application efficiency with the least load on the system. Clearly, the *vertical* layout provided better performance, as expected.

This experiment demonstrates the effect that layout has on application performance. Because we cannot predict whether applications will request rows or columns, storing the data both *horizontally* and *vertically* will guarantee performance gain.

4.2 Differentiated Service Support with Diff-Serv Scheduler

Osiris is being developed to simultaneously serve clients on both commodity and high performance networks. Because of the *premium* nature of high performance networks, it is desirable that clients on these networks have some precedence over clients on commodity networks. Treating *premium clients* and *commodity clients* differently constitutes a type of Quality of Service(QoS) called *differentiated services* [20].

Differentiated service support in Osiris is another mechanism that at-

TM	TT in seconds	AATR in Mbps
traditional single socket	96	8.3
PSocket size 2	57	14.0
PSocket size 3	34	23.5
PSocket size 4	30	26.7
PSocket size 5	26	30.8
PSocket size 6	26	30.8
PSocket size 7	26	30.8

Table 3: Performance of Transport Layer Multiplexing with PSocket. (Note: The practical limit of the 45 Mb/s DS-3 appears to be about 35 Mb/s.)
TM - Transport Mechanism
TT - Transfer Time for 100 MBytes
AATR - Application Apparent Transfer Rate

tempts to contribute to the requirement of efficient read access. In this context, efficiency refers to system wide resource utilization as opposed to per process performance.

Because the currently popular Internet protocol suite (IP) does not support any kind of QoS mechanism, we chose to implement differentiated service support as a characteristic of the scheduling mechanism for client requests to Osiris. We refer to this scheduler as the Diff-Serv Scheduler.

When a client attaches to Osiris, it informs the server whether it is a *premium client* or a *commodity client*. Data block requests are then scheduled for service as they arrive with *premium client* requests getting preferential treatment. Please see [12] for full design and implementation details of the Diff-Serv Scheduler.

For our experimental study, a single server was run on a machine connected to the network through Switched Fast Ethernet (100 Mbps). An equal number of clients connected to both Switched Fast Ethernet (*premium clients*) and Switched Ethernet (*commodity clients*) were launched and connected to the server.

The *premium clients* each made 10,000 random block requests, and the *commodity clients* each made 5,000 random block requests. The default block size for Osiris is 16KB. Every client waited for an exponentially distributed random delay between block requests. This delay was introduced to cause a Poisson distribution of request arrivals to the server and was an attempt to simulate real application behavior.

Experiments were conducted which compare system performance with Diff-Serv scheduling turned on against system performance with Diff-Serv scheduling turned off. Measurements were made with a total of two to ten clients. The results are presented in Table 2. Please note that when Diff-Server scheduling is turned off, the system defaults to FIFO scheduling.

The desired results were achieved. In all cases, *premium client* response time improved while *commodity client* response time diminished when our implementation of Diff-Serv scheduling was turned-on.

4.3 High Performance Network Transfer Support with PSocket

It has been well documented that latency characteristics of TCP over wide area networks, or more precisely networks with large "bandwidth · delay" products, have a significant negative impact on per process communication performance [13]. Various protocol and implementation level solutions have been suggested [14] [5]. One technique is for the sender to send multiple messages to the receiver in parallel [21].

In order to provide high performance network transfer support, we allow a single process to break up a message and then send the pieces in parallel over multiple communication links (e.g., TCP sockets) to the receiver who then rebuilds the entire message. We refer to this technique as Transport Layer Multiplexing and have implemented a simple-to-use interface for application integration called PSocket (as in Parallel Socket). For full details please refer to [2].

Osiris will use PSockets to increase the data transfer rate on a per client process basis. The results in Table 3 are from a single, non-threaded sender process using PSocket, located at the University of Illinois at Chicago, sending data to a single non-threaded receiver process using PSocket located at Highway One. The two sites are connected by the NSF/MCI vBNS Network. Highway One connects to vBNS via a DS-3 link, which limited the maximum theoretical bandwidth of the testbed to 45 Mb/s.

The experiment measured the wall clock transfer time of a 100 MByte buffer. Results show that with a PSocket of size 5, a large portion of the practical transfer rate of the DS-3 was consumed by the transfer. As a reference, the transfer rate using traditional, single socket programming was given.

5 Conclusion

In general, the less data which distributed data mining systems move, the less expensive the computation. However, due to the level of accuracy required or to the nature of the data, it is sometimes necessary to move large amounts of data between sites. With the emergence of high performance networks this becomes practical in many circumstances in which it would have previously been impractical.

In this paper, we have described some of the design considerations for a high performance data server called Osiris which is part of the Papyrus distributed data mining infrastructure and presented some experimental results describing its use on an application benchmark requiring the computation of histograms.

In particular, we describe a design which supports high performance storage, high performance network transfer, and differentiated network services, such as commodity links and high performance links. This design provides at least a 10x improvement over a more naive design. We expect this to grow to 100x for certain applications and network configurations.

References

[1] S. Bailey, E. Creel, and R. L. Grossman, DataSpace: Protocols and Languages for Distributed Data Mining, National Center for Data Mining/Laboratory for Advanced Computing Technical Report, http://www.ncdm.uic.edu, 1999.

[2] S. Bailey, R. L. Grossman, Transport Layer Multiplexing with PSocket, National Center for Data Mining Technical Report, 1999.

[3] P. Chan and H. Kargupta, editors, Proceedings of the Workshop on Distributed Data Mining, The Fourth International Conference on Knowledge Discovery and Data Mining New York City, 1999, to appear.

[4] T. G. Dietterich, Machine Learning Research: Four Current Directions, AI Magazine Volume 18, pages 97-136, 1997.

[5] D. J. Farber, J. D. Touch, An Experiment in Latency Reduction, IEEE Infocom, Toronto, June 1994, pp. 175-181.

[6] R. L. Grossman, H. Bodek, D. Northcutt, and H. V. Poor, Data Mining and Tree-based Optimization, Proceedings of the Second International

Conference on Knowledge Discovery and Data Mining, E. Simoudis, J. Han and U. Fayyad, editors, AAAI Press, Menlo Park, California, 1996, pp 323-326.

[7] R. L. Grossman, S. Bailey, S. Kasif, D. Mon, A. Ramu and B. Malhi, The Preliminary Design of Papyrus: A System for High Performance, Distributed Data Mining over Clusters, Meta-Clusters and Super-Clusters, Proceedings of the Workshop on Distributed Data Mining, The Fourth International Conference on Knowledge Discovery and Data Mining New York City, August 27-31, 1998, to appear.

[8] R. L. Grossman and Y. Guo, An Overview of High Performance and Distributed Data Mining, submitted for publication.

[9] R. L. Grossman, S. Bailey, A. Ramu and B. Malhi, P. Hallstrom, I. Pulleyn and X. Qin, The Management and Mining of Multiple Predictive Models Using the Predictive Modeling Markup Language (PMML), Information and Software Technology, 1999.

[10] The Terabyte Challenge: An Open, Distributed Testbed for Managing and Mining Massive Data Sets, Proceedings of the 1998 Conference on Supercomputing, IEEE.

[11] Y. Guo, S. M. Rueger, J. Sutiwaraphun, and J. Forbes-Millott, Meta-Learnig for Parallel Data Mining, in *Proceedings of the Seventh Parallel Computing Workshop,* pages 1-2, 1997.

[12] S. Gutti, A Differentiated Services Scheduler for Papyrus DSTP Servers, Master's Thesis, University of Illinois at Chicago, 1999.

[13] V. Jacobson, Congestion Avoidance and Control, SIGCOMM '88, Stanford, CA., August 1988.

[14] V. Jacobson, and R. Braden, TCP Extensions for Long-Delay Paths, RFC-1072, LBL and USC/Information Sciences Institute, October 1988.

[15] H. Kargupta, I. Hamzaoglu and B. Stafford, Scalable, Distributed Data Mining Using an Agent Based Architecture, in D. Heckerman, H. Mannila, D. Pregibon, and R. Uthurusamy, editors, *Proceedings the Third International Conference on the Knowledge Discovery and Data Mining,* AAAI Press, Menlo Park, California, pages 211-214, 1997.

[16] R. W. Moore, C. Baru, R. Marciano, A. Rajasekar, and M. Wan, Data-Intensive Computing, Ian Foster and C. Kesselman, editors, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, San Francisco, 1999, pages 105-129.

[17] A. E. Raftery, D. Madigan, and J. A. Hoeting, 1996. Bayesian Model Averaging for Linear Regression Models. Journal of the American Statistical Association 92:179- 191.

[18] S. Stolfo, A. L. Prodromidis, and P. K. Chan, JAM: Java Agents for Meta-Learning over Distributed Databases, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining,* AAAI Press, Menlo Park, California, 1997.

[19] R. Subramonian and S. Parthasarathy, An Architecture for Distributed Data Mining, to appear.

[20] B. Teitelbaum and J. Sikora (1998), Differentiated Services for Internet2, Draft Proposal, http://www.internet2.edu/qos/may98Workshop/html/diffserv.html

[21] J. D. Touch, Parallel Communication, *the proceedings of Infocomm 1993*, San Francisco CA. March 28-April 1, 1993.

[22] R. L. Grossman and A. Turinsky, Optimal Strategies for Distributed Data Mining using Data and Model Partitions, submitted for publication.

[23] Xu, L.; and M.I. Jordan, M. I. 1993. EM Learning on A Generalised Finite Mixture Model for Combining Multiple Classifiers. In Proceedings of World Congress on Neural Networks. Hillsdale, NJ: Erlbaum