The Preliminary Design of Papyrus: A System for High Performance, Distributed Data Mining over Clusters, Meta-Clusters and Super-Clusters

Robert Grossman

National Center for Data Mining
University of Illinois at Chicago
851 South Morgan Street M/C 249
Chicago, IL 60607
grossman@uic.edu

Magnify, Inc.
100 South Wacker Drive
Suite 1130
Chicago, IL 60606
rlg@magnify.com

Stuart Bailey, Ashok Ramu, Balinder Malhi and Andrei Turinsky

National Center for Data Mining
University of Illinois at Chicago
851 South Morgan Street M/C 249
Chicago, IL 60607

August 6, 1999

This is a draft of the paper: R. L. Grossman, S. Bailey, A. Ramu, B. Malhi and A. Turinsky, The Preliminary Design of Papyrus: A System for High Performance, Distributed Data Mining over Clusters, in Advances in Distributed and Parallel Knowledge Discovery, H. Kargupta and P. Chan, editors, AAAI Press/The MIT Press, Menlo Park, California, 2000, pages 259-275.

Abstract

Data mining is a problem for which cluster computing provides a competitive alternative to specialized high performance computers for mining large data sets. Distribued clusters provide a natural infrastructure for mining large distributed data sets. Distributed clusters can be connected by commodity networks to form what we call meta-clusters and by high performance networks to form what we call super-clusters. In this paper, we describe the design of a system called

Papyrus which is designed for mining data which is distributed over clusters, meta-clusters, and super-clusters. We also describe some experimental results of a preliminary implementation.

1 Introduction

For many problems, clusters of workstations connected with specialized switching fabrics or high performance networks provide a competitive alternative to specialized high performance computers, including MPP computers.

Clusters of workstations have proved themselves to be very effective for a variety of data mining applications [12]. The data mining process involves both compute intensive and data intensive steps. Clusters serve two fundamental roles: Data-clusters provide the storage and data management services for the data sets being mined. Compute-clusters provide the compute services required by the data cleaning, data preparation and data mining tasks.

It is natural therefore to use distributed clusters as the infrastructure for distributed data mining. Our interest in this article is to focus on the special, but important case, in which globally distributed high performance clusters of workstations are connected with networks supporting different levels of service.

Network quality of service (QoS) is measured along several dimensions: *rate quarantees* concerning the amount of data that an application requires, *delay and jitter guarantees* concerning the timeliness of delivery, and *loss guarantees* concerning the quality of delivery [16]. A network with QoS provides some type of guarantee along one or more of these dimensions; in contrast, today's commodity internet uses a best effort model.

Today a network with high performance links may offer links which are 100x (or more) faster than commodity links. For example, the NFS vBNS network supports OC-3 links (155 MB/s), which in practice provides performance about 100x faster than the commodity internet. This provides a model for emerging premium services or differentiated services [28] and [16], which allow applications, for example, to request services with bandwidth guarantees. We call clusters (or clusters of clusters) connected with commodity services *meta-clusters* and clusters (or clusters of clusters) connected with premium services *super-clusters*.

The testbed used for the experimental studies described below consisted of five workstation clusters (in Chicago, Philadelphia, College Park, Davis, and Toronto) forming a super-cluster connected to two clusters (in London and Canberra) to form a meta-cluster. More specifically, the clusters in Chicago and Toronto were connected by a 155 Mbs network; the clusters in Chicago, Davis and Philadelphia were connected by 45 Mbs network; and the remaining clusters were connected by the commodity internet.

We expect meta-clusters and super-clusters to grow more common with the emergence of the next generation internet. For example, a large company with distributed operations employing a virtual private network may have operational data in several cities connected with a high performance network employing premium services, while marketing related data may be accessible through a commodity network. As the amount of data grows, replacing servers with data and compute clusters provides a cost effective means of supplying the appropriate data management and processing capabilities required by data mining.

Concretely, data mining may be viewed as extracting a learning set from one or more (distributed) data warehouses and applying a data mining algorithm to produce a predictive model or rule set [14]. Different strategies are possible, depending upon the data, its distribution, the resources available, and the accuracy required:

MR (Move Results) Today's commodity networks can be used to move the results of local data mining computations to a central site.

MM (Move Models) Commodity networks can also be used to move predictive models from site to site.

MD (Move Data) Next generation broadband networks also create the possibility of moving large amounts of data.

The majority of the work in distributed data mining has focused on the first two strategies MR and MM [3]. In this paper, we introduce Papyrus, a distributed data mining system supporting all three strategies, as well as mixtures of the strategies. We also describe experimental studies for strategies MD and MM.

In this paper, we are interested in distributed data mining problems with the following characteristics: data is available on data clusters, the fundamental task is to correlate data between two or more different data clusters, different strategies are available depending upon whether the clusters form a super-cluster or meta-cluster, and the appropriate compute services are provided by compute clusters.

Section 2 provides background material and describes related work. The key concepts underlying the system are reviewed in Section 3. The design of the system is described in Section 4. The implementation is described in Section 5. Experimental studies are described in Section 6. Section 7 contains the conclusion and future work.

2 Background and Related Work

This section is based in part on [15].

Several systems have been developed for distributed data mining. Perhaps the most mature are: the JAM system developed by Stolfo et. al. [26], the Kensington system developed by Guo et. al. [17], and BODHI developed by Kargupta et. al. [18]. 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, Kensington and BODHI all employ local learning.

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 methods 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 [26]. *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 [17]. 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 [25], stacking [30], partition learning [11], and other statistical methods, such as mixture of experts [31]. 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 and Kensington, 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.

All four systems make use of Java. JAM employs Java applets to move machine learning algorithms to distributed data. Kensington uses Java JDBC to mine distributed data. Papyrus uses Java aglets [20].

The distributed data mining system developed by Subramonian and Parthasarathy [27] 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.

Distributed clusters of workstations are an example of what are becoming known as computational grids. A *computational grid* according to [7] is a "hardware and software infrastructure that provides dependable, consistent,

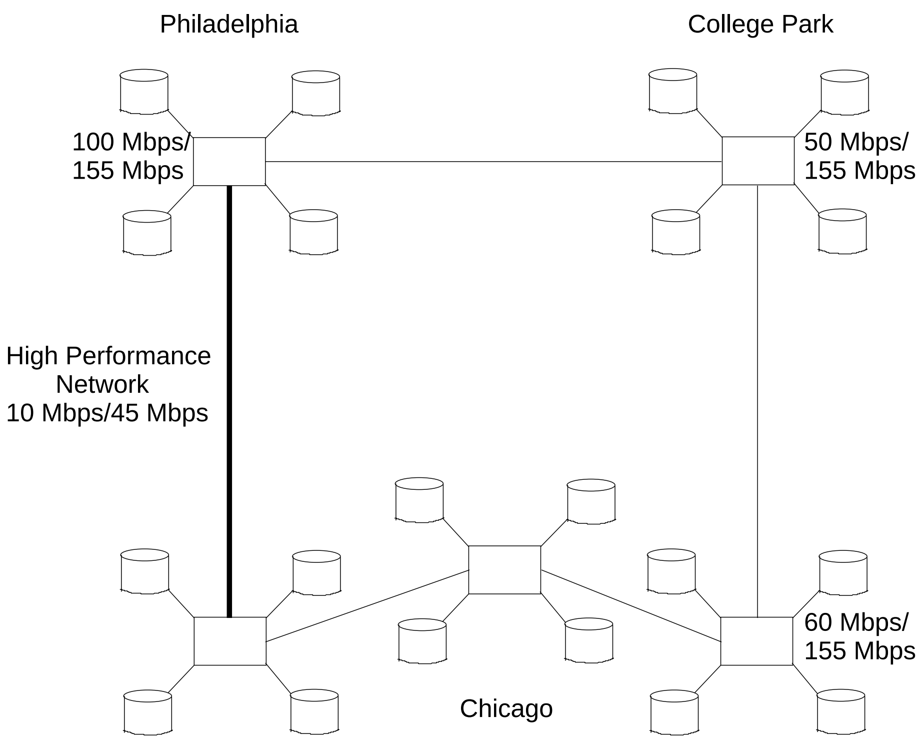

Figure 1: Clusters in Chicago and Philadelphia are connected with a high performance network to form a super-cluster. The super-cluster is connected with a cluster in College Park to form a meta-cluster. Clusters are of two types: data clusters providing data or compute cluster providing cycles. Papyrus applications have transparent access to any of the data or compute clusters in the meta-cluster.

pervasive, and inexpensive access to high-end computational capabilities." In [7], grid applications are divided into five classes: 1) distributed supercomputing applications requiring lots of cpu and memory, 2) high-throughput computing applications which use otherwise idle resources to tackle large problems, 3) on demand computing applications which integrate remote resources with local computation, 4) data intensive computing applications requiring the analysis of large data sets, and 5) collaborative computing applications which enhance human to human interactions.

Data mining cuts across each of these classes: data mining applications can be both cpu and i/o intensive (Classes 1 and 4); data mining algorithms such as genetic algorithms which require searching a space of hypotheses can exploit high-throughput techniques (Class 2); data mining has been applied to data from remote instrumentation (Class 3); finally, the interactive exploration and visualizaton of large data sets is becoming more common (Class 5).

There are several projects underway developing a computing infrastructure for grid applications, including Globus [6], Legion [10, 8], and NOW [1]. In contrast to these systems which are broadly concerned with providing general grid services to distributed applications, Papyrus is focused on just those services required for distributed data mining over grids built from clusters and clusters of clusters. The philosophy of Papyrus is to use general grid and cluster services whenever possible. As grid services such as Globus and Legion mature, we expect that future versions of Papyrus will be based, in part, upon them.

Similar to grid services such as Globus [6], Papyrus provides a shared (wide area) persistent data space, transparent remote execution (of data mining processes), wide area parallel processing (of data mining processes), and scheduling (of data mining processes). Of course, Globus provides this for general grid applications, while Papyrus provides specialized vertically integrated versions of these services in a framework suitable for distributed data mining. Both Papyrus and Globus take an "hour glass" layered approach to system design in which a few protocols and services (the neck of the hour glass) are used to connect adjacent layers in the system.

Legion [8] advocates an object and component view toward grid services. Objects have well defined interfaces, communicate with data and control paths and respond to events broadcast by other objects. Papyrus exploits data paths between clusters. Clusters communicate by using agents, and data access is via specialized servers designed for high performance and distributed data mining rather than through general object services or databases.

The Berkeley NOW Project [1] provides infrastructure for high performance distributed computing using workstation clusters, including scheduling middleware, a high performance messaging system, and a scalable parallel file system. Papyrus uses a high performance, light weight object manager [12] instead of a parallel file system and is designing a scheduler specifically targeted towards data intensive computations over wide area networks with different levels of services instead of using a more general purpose scheduler for distributed applications.

3 Key Concepts

In this section, we describe some of the key concepts underlying Papyrus following the exposition in [29].

Data Ensembles. In this paper, we restrict our attention to data mining problems in which we are given 1) a mechanism for partitioning a data set and 2) a mechanism for combining partitioned data into a single data set. In this case, we speak of a *data partition* or *data ensemble.* For example, when building a fraud model for credit card accounts, the data may be partitioned by customer, and separate models built from different sets of customers. As another example, a delinquency model for credit card accounts may be built from two data sets – a set of credit card transactions and a set summarizing account level data, such as when payments were received and for how much. The first example illustrates what is sometimes called horizontal partitioning – the domain is partitioned, while the second example illustrates what is sometimes called vertical partitioning – the range is partitioned [19].

Data Mining. Data mining can be viewed quite broadly as the semi-automated extraction of knowledge from data [5]. We take a much narrower point of view, regarding data mining as the process of extracting a learning set from a data warehouse and applying a data mining algorithm to produce a predictive model or rule set [14]. Continuing, the model can be applied to data to obtain a numerical result, which we view as a vector.

Model Ensembles. In this paper, we also restrict attention to collections or *ensembles* of predictive models which may be combined to produce a single model. A basic example is provided by predictive models built from partitioned data and then combined by voting [4]. As another example consider a cluster model, in which a data set is summarized by specifying the centroids of k clusters. The model may be partitioned by creating two collections of centroids containing k_1 and k_2 centroids, where $k_1 + k_2 = k$ and two cluster models may be combined by simply concatenating their centroids.

The Fundamental Trade-Off. Large data files may be several orders of magnitude larger than a file describing a model, while a model file may be several orders of magnitude larger than a result vector. In a distributed data mining system, different strategies are possible depending upon whether data files, model files, or result vectors are moved from node to node.

In general the most accurate result is obtained by moving all the data to a single node. This is usually also the most expensive. In this paper, we measure the expense using a cost function which includes both computation and communication costs. At the other extreme, we can process all the data locally obtaining local results, and combine the local results to obtain the final result. In general, this approach is less expensive, but also less accurate. To summarize, in distributed data mining, there is a fundamental trade-off between the accuracy desired for the predictive model and the cost one is willing to bear for the computation.

Configurations. We assume that there are n different sites connected by a network. A *configuration* is given by specifying the following data:

1. A graph with nodes i, $i = 1, \ldots, n$ describing the network. One of the nodes, say the K^{th}, is the network *root* where the overall result will be computed.
2. A vector $\{D_i\}_{i=1}^n$ describing the initial distribution of the data.
3. The cost c_{ij} (measured in dollars per Gigabyte) to move data from the i^{th} to the j^{th} node, via the least expensive path.
4. The cost c_i (measured in dollars per Gigabyte) to process data at the i^{th} node into a predictive model.
5. A constant α describing the amount of compression when the data on a node is processed to compute a predictive model.

4 Strategies

Fix a configuration as defined above. A node can employ one of three different strategies:

MD Move Data. Ship raw data across the network to another node for processing.

MM Move Models. Process the data locally to produce a predictive model and ship the predictive model to another node for further processing.

MR Move Results. Process the data locally until a result is obtained and ship the result to another node for further processing.

In general, as we progress from MD, to MM, to MR strategies, there is a loss of accuracy, but a decrease in the cost of the computation. A complete strategy moves data, models, and results from node to node until the root produces a final result.

For some problems, sufficient accuracy is required so that all the data must be moved to a central root. For other problems, sufficient speed is desired so that all the computation is done locally. Recall that it takes approximately 24 hours to move a terabyte of data over a OC-3 network. Given terabytes of data distributed over a meta-cluster of clusters and super-clusters, the correct strategy can mean the difference between waiting minutes instead of hours.

To simplify the discussion, we only consider mixed strategies involving MD and MM; the general case can be handled similarly. See [29]. A *strategy* X as a matrix of numbers

$$X = [x_{ij}]_{i,j=1}^n$$

where x_{ij} is the amount of data D_i that is moved from the i^{th} node to the j^{th} node for processing by the strategy. After the move, the amount of data at the j^{th} node becomes $\tilde{D}_j$. The *cost function* for a strategy X is easily computed as

$$C(X) = \sum_{ij} \left(c_{ij}x_{ij} + c_j x_{ij} + c_{jK}\alpha x_{ij}\right).$$

The sum is over nodes i which are the sources of data and over nodes j which are the targets of data.

A critical observation is that in many cases the cost function is convex [29]. In such cases, the least possible error ϵ_0 occurs when all data is moved to a single node and the largest possible error $\epsilon_{\max}$ occurs when the data is evenly distributed among all nodes. Note that the vector $\tilde{D}(X) = \left(\tilde{D}_1, \ldots, \tilde{D}_n\right)$ defines the relevant data distribution, where each $\tilde{D}_j = \sum_i x_{ij}$.

We use the following form of the *error function* for a strategy X:

$$\epsilon(X) = \epsilon_0 + \rho_D \left(1 - \frac{\|\tilde{D}(X)\|}{|D|}\right), \qquad \rho_D = \frac{\epsilon_{\max} - \epsilon_0}{1 - \frac{1}{\sqrt{n}}}$$

where $|D| = \sum_j \tilde{D}_j$ is the overall amount of data in the network, $\|\tilde{D}\|$ is the usual Euclidean norm of a vector $\tilde{D}(X)$, and ρ_D is a scaling coefficient.

Suppose we are given an error tolerance threshold ϵ_{tol}. We can now find an optimal strategy by solving the optimization problem:

$$\mathrm{Min}_{\mathrm{X}}\ C(X)$$

$$\epsilon(X) \leq \epsilon_{tol}.$$

It is easy to find examples [29] in which there are intermediate strategies that are more cost effective than the naive ones of either leaving all the data in place or moving all the data to a single fixed node. Because of this, it makes sense to design distributed systems with the flexibility of moving data, moving predictive models, or moving the results of local computations. This is one of the key ideas behind the Papyrus system. Papyrus, as far as we know, is the first distributed data mining system to be designed with this level of flexibility.

5 Architectural Design

Recall that we are interested in distributed data mining systems in which the data is accessed through data clusters, analyzed using compute clusters, and where the data and compute clusters are combined using commodity networks to form meta-clusters and using high performance networks to form super-clusters.

For simplicity, we view a single node as a cluster of size one. In Papyrus, clusters interact in two ways:

1. Metadata is moved between clusters using agents. Each cluster has a single node with is designated as the Cluster Access Point (CAP). Agents can move queries, predictive models, result vectors, and other metadata between the CAPs of different clusters.

2. Data is moved between clusters using a distributed data warehouse. In practice, even with only moderate amounts of data, this is only practical if the two clusters are connected with a high performance network.

Given the viewpoint described above, in which data mining is viewed as the process of applying data mining algorithms to learning sets extracted from data warehouses to produce predictive models, it is natural to design Papyrus as a layered system, consisting of the following four main layers [13]:

Data warehouse layer. The lowest layer consists of a data warehouse which is designed for local and distributed clusters of workstations. Training sets are extracted from the data warehouse. In practice, small amounts of data can be moved between clusters in general, and larger amounts of data can be moved when the clusters are connected with high performance networks.

Data mining layer. The role of the data mining layer is to apply one or more data mining algorithms to training sets extracted from the data warehouse to produce predictive models, rule sets, or results. More precisely, we assume that the input to the data mining layer is a data set — the learning set — and the output is a model or ensemble of models. By a model, we mean both predictive rules and rule sets.

Predictive modeling layer. This layer manages predictive models and ensembles of predictive models. We have contributed to the development of an XML [22] mark up language called the Predictive Model Markup Language (PMML) for predictive models and rule sets. PMML supports predictive models, ensembles of predictive models, and all the meta-data required to describe them and use them effectively [14]. A description of the current draft of PMML can be found in [23]. The predictive modeling layer handles models in PMML. Concretely, the data mining layer extracts learning sets from the data warehouse and produces models or rules in PMML, which are managed by the predictive modeling layer.

Agent layer. The agent layer moves queries, predictive models, metadata, and the results of local computations between the Cluster Access Points. We have developed an XML markup language which we call the Data Discovery Markup Language (DDML) which is used to describe the queries and metadata associated with a distributed data mining computation. Agents move DDML metadata and PMML models between Cluster Access Points. Separately, through control expressed in DDML files, data can be moved directly between two distributed data clusters.

As described above, a query to Papyrus can be processed using three different strategies (MR, MM, and MD).

Selecting a Query

1. A Papyrus application dispatches agents to gather DDML files from the various registered CAPs.
2. The DDML files returned by the agents are analyzed by the application and the user chooses or specifies a query.

Move Results (MR)

3. The DDML query is carried by agents to the appropriate CAPs, where the query is executed to produce results.
4. The agents return the results in DDML using a commodity network to the Papyrus application, where they are processed to produce a final result.

Move Models (MM)

3. The query in DDML is carried by agents to the appropriate CAPs, where the query is executed to produce predictive models.
4. The agents return the predictive models in PMML using a commodity network to the Papyrus application, where they are processed to produce a final result.

Move Data (MD)

3. The query and the request to move data in DDML is carried by agents to the various CAPs. To process the query, data is moved by the data warehousing layer using a high performance network to a root cluster, where the query is processed.
4. Agents return the results in DDML from the root cluster to the Papyrus application, where they are processed to produce a final result.

6 Preliminary Implementation

We have developed two versions of Papyrus following the general design and ideas sketched above. Papyrus Version 0.8 was developed and tested during 1997 and demonstrated at the Supercomputing 97 Conference in San Jose, California in November, 1997. Papyrus Version 0.9 was developed and tested during 1998 and demonstrated at the Supercomputing 98 Conference in Orlando in November, 1998. We are currently developing Version 1.0 of Papyrus.

The data management layer of Version 0.8 of Papyrus used a high performance, light weight persistent object manager we developed called PTool [12]. Papyrus Version 0.9 used a version of PTool we developed that provided some support for wide area clusters. In particular, we were able to stripe data over wide area data clusters. The data management layer of Papyrus Version 1.0 will use a different design, with explicit support for data mining primitives, specialized protocols for moving data over high performance networks, and explicit support for multiple network protocols.

Since our main interest was in the system design of a high performance, distributed data mining system, and not in work related to high performance data mining algorithms per se, we chose to use standard algorithms for the data mining layer. We chose the popular C4.5 implementation of decision trees as the main tool in the data mining layer [24].

We modified C4.5 to work with data clusters and to emit PMML. Papyrus Version 0.8 used SGML as the basis for PMML. Papyrus Version 0.9 used XML [22], as will Papyrus Version 1.0. The general design for Papyrus calls for a layer to manage predictive models in PMML called Anubis. In Versions 0.8 and 0.9 of Papyrus, we did not use a separate layer for this, but rather implemented the necessary code in the Papyrus applications themselves. For example, some of the applications combined several PMML models into a single meta-model using a simple voting strategy.

We built an agent layer called Bast to move queries, predictive models, results and meta-data between the various data clusters. Bast Version 0.8 used Agent-TCL [9] to move SGML data describing queries and the other associated data and metadata managed by Bast. Bast Version 0.9 uses aglets [20] to move XML data and metadata in a language we designed for the this purpose called the Data Discovery Markup Language (DMML).

For simplicity, we used the following strategy for moving data and metadata:

1. For data distributed across a super-cluster, we used PTool to move the data and mine it as if it were locally resident.

2. For data distributed across a meta-cluster, we built predictive models locally on each data cluster or super-cluster and merged the results with a voting scheme.

7 Experimental Results

In this section, we describe a series of five experiments which used Papyrus Version 0.9. We performed two of the experiments in November, 1998 at the Supercomputing 98 Conference in Orlando. The other three experiments were conducted afterwards to explore some of the issues that arose in the first two experiments. We also describe how these experiments influenced the design of Papyrus 1.0, which is currently under development.

The goal in the first two experiments was to gain information about the overall suitability of the Papyrus system architecture and design, especially the

Layer	**Name**	**Version 0.8**	**Version 0.9**
Agent	Bast	Agent-TCL moving PMML	Aglets moving DDML & PMML
Predictive Model	Anubis	unimplemented	unimplemented
Data Mining	Thoth	C4.5 producing PMML	C4.5 producing PMML
Data Warehouse	Osiris	PTool suppporting local clusters	PTool supporting local & wide area clusters

Table 1: Summary of implementation of Version 0.8 and 0.9 of Papyrus. C4.5 was modified to produce SGML for v0.8 and PMML for Version v0.9.

design of the data warehouse and agent layers. We emphasize that our goal was not to test distributed data mining algorithms per se but rather to improve our understanding of some of the critical factors effecting a distributed data mining system operating over networks with different levels of services.

In the first experiment at Supercomputing 98, we analyzed high energy physics data with a 30 node super-cluster distributed between Chicago, Philadelphia, College Park, Davis, Orlando, and Toronto. In the second experiment, we analyzed health care outcome data with a 6 node meta-cluster in Chicago, Philadelphia, College Park, Orlando, London and Canberra.

In the first experiment, the super-cluster had access to approximately 480 Gigabytes of high energy physics data, organized into approximately 79 million "events," representing putative particle collisions. We adopted standard physics analysis code to run on the super-cluster and created an application benchmark called Event1. The goal with Event1 is to maximize the number of events analyzed per second. As an example, the Event1 benchmark on a subset of approximately 6 million events, spanning approximately 38.4 Gigabytes distributed over 14 nodes, took between 18,000 and 54,000 seconds depending upon the node.

The US data clusters were connected by the NSFs very high speed Backbone Network Server (vBNS), which consists of a 622 Mb/sec, fully switched, Asynchronous Transfer Mode (ATM) internal backbone and 45 Mb/sec - 155 Mb/sec edge ATM links to each site. The Toronto cluster was connected to the vBNS via a 45 Mb/sec link. The limit for the data analysis on the super-cluster was essentially the speed with which data could be managed and moved across the network, which was the responsibility of Papyrus' data warehouse layer. Roughly speaking, each network interface of the Papyrus system could move data at approximately 4 Mb/sec per process, providing an upper bound per node of approximately 20-30 Mb/sec, and an upper bound per site of approximately 60-90 Mb/sec.

In Version 0.9 of Papyrus, we used custom code which connected the various nodes in the supercluster with an appropriate number of sockets. Without taking this approach, the performance of the Event1 benchmark was rather disappointing, despite the presence of the high performance links. In a follow up experiment described in Table 2, we examined this issue more carefully, varying the number of sockets connecting two nodes in a supercluster from 1 to 7. On the basis of these two experiments, we made the decision to develop a library for moving data between two nodes in a supercluster which multiplexes several sockets to maximize the amount of data transfered per second. This library will be used in Papyrus Version 1.0.

In the second follow up experiment described in Table 3, the same event data was arranged two ways: by event and by attribute. For the Event1 benchmark, although more data was moved with the horizontal layout by event, more events could be analyzed with the vertical layout by attribute. The difference was over 6x, and could be larger depending upon the query. Of course, other queries would see the same speed up for exactly the opposite layout. For this reason, we are investigating supporting both vertical and horizontal layouts in Papyrus Version 1.

For the second series of experiments at Supercomputing 98, the meta-cluster analyzed approximately 600,000 health care outcome records comprising less than a Gigabyte of data, but distributed over three continents. In this case, the analysis was essentially limited by the congestion of the commodity internet, which was especially pronounced for the international links. The agent based communication utilized by the Papyrus cluster layer worked effectively for the meta-cluster queries we tested.

In these experiments, we combined C4.5 trees [24] built at each local site into a single classifier using the Papyrus cluster layer Bast. The Papyrus application combined the trees using a majority vote, which is a standard technique when working with ensembles of classifiers [4]. Our interest was understanding whether Papyrus would work over globally distributed meta-clusters, in which there were wide variations in latency and bandwidths between the various sites. We concluded that this approach could work effectively. Since some queries were not able to complete due to network congestion, we are considering putting explicit support in Papyrus Version 1.0 for more gracefully handling nodes which cannot respond in a timely fashion.

In the third follow up experiment, we empirically examined the trade-offs when moving models (MM) built on distributed data vs. moving the data (MD) and building a centralized model for the health care outcome data. This experiment used a super-cluster and a particular application benchmark called HCOD1. See Tables 4 and 5. For this particular benchmark, essentially the same accuracy could be obtained by building local trees and merging them (MM). Moving models was approximately 2.5x faster than moving data. Notice that with a supercluster moving data (MD) takes essentially the same length of time whether the supercluster is local or geographically distributed. Of course, with other queries and other data sets, the loss suffered when employing the MM strategy might not be acceptable. These observations lead to the cost based approach to selecting strategies described in Section 4. We plan to implement this for Papyrus Version 1.0

Number of Sockets	TT in seconds	AATR in Mb/sec
1 socket	96	8.3
2 sockets	57	14.0
3 sockets	34	23.5
4 sockets	30	26.7
5 sockets	26	30.8
6 sockets	26	30.8
7 sockets	26	30.8

Table 2: This table shows the result of using more than one socket with multiplexing to move data between nodes in a supercluster. The practical limit of the 45 Mb/s DS-3 link appears to be about 35 Mb/s. TT - Transfer Time for 100 MBytes, AATR - Application Apparent Transfer Rate

Horizontal Store: Store Size = 4 GB, Total Data Moved = 4GB				
Comp. Servers	Processes	Events/second	Mb/seconds	Seconds
1	1	92	4.67	7388
1	2	153	7.7	4466
1	4	242	12.2	2825
1	8	315	15.84	2168
4	4	320	16.12	2140
4	16	448	22.4	1526
8	32	503	25.6	1359

Vertical Store: Store Size = 6.85 GB, Total Data Moved = 0.47GB				
Comp. Servers	Processes	Events/second	Mb/seconds	Seconds
1	1	392	1.42	2833
1	2	664	2.42	1673
1	4	911	3.17	1218
1	8	951	3.31	1166
4	4	1555	5.64	714
4	8	2498	9.12	444
4	16	3044	10.53	365
4	32	3272	11.32	339

Table 3: Data may be placed on the disk by record (horizontally) or by attribute (vertically). The optimal choice depends upon the query. This table analyzes the performance of the Osiris data server for both of these choices for a particular benchmark query of high energy physics data called Event1. The events processed per second, the amount of data moved per second, and the total time in seconds to complete the query are given.

Data Transfer vs. Model Transfer (5 locations)							
Strategy				Data Transfer		Model Transfer	
# Records	LAN DT	WAN DT	C45	LAN Total	WAN Total	LAN Total	WAN Total
5000	0.6	1.661	9.25	9.85	10.911	17.6	21.1
10000	0.96	2.885	18.62	19.58	21.505	20.91	27.28
50000	3	11.96	156.81	159.81	168.77	86.78	79.56
100000	4.3	21.75	424.14	428.44	445.89	114.03	161.24

Table 4: Query time using different strategies for local (LAN) and wide area (WAN) clusters connected with high performance links. Columns 2-8 indicates time in seconds for the health care application benchmark HCOD1. Total refers to the time required for both the execution time of C4.5 and the time required to move either the data (DT) or the model (MT) as indicated. DT - Data Transfer, MT - Model Transfer

# Records	Number of Models						
	1	3	5	7	9	15	21
1000	14.4	14.0	13.6	13.6	15.2	14.8	14.0
5000	14.9	14.96	15.36	15.36	15.12	15.6	15.2
10000	12.7	13.44	12.92	14.32	13.92	14.24	14.16
50000	13.8	11.984	13	12.912	13.072	13.072	13.126
100000	12.8	12.812	12.912	12.844	12.852	12.972	13.128
200000		12.37	12.16	12.284	12.226	12.228	12.198

Table 5: The prediction error in percent as the number of records and the number of models varies for the HCOD1 application benchmark.

8 Conclusions and Future Work

As workstation clusters and high performance networks grow more common, clusters, meta-clusters (clusters linked by commodity networks), and super-clusters (clusters linked by high performance networks) should prove a popular infrastructure for mining large and distributed data sets. Papyrus uses a simple layered architecture and is flexible enough to employ strategies which can either move data, move predictive models or move the numerical results of computations over distributed clusters of workstations linked by high performance and commodity connections.

As described in Section 7, the Papyrus architecture and implementation has demonstrated the practicality of mining distributed Gigabyte size data sets over high performance DS-3 and OC-3 networks. With queries taking hundreds to thousands of seconds, it is important to develop systems which are intelligent enough and flexible enough to move only the smallest amount of data consistent with achieving results of acceptable accuracy. This is one of the goals of Papyrus Version 1.0

Based upon the experiments performed to date using Versions 0.8 and 0.9 of Papyrus, we are currently developing Version 1.0:

1. Effectively using high performance links in Version 0.9 of Papyrus required that Papyrus applications explicitly make use of multiple sockets to deal with the well known latency problems of TCP ACKs. A library (PSocket) is provided in Papyrus 1.0 so that Papyrus applications can transparently use high performance links.

2. Version 0.9 of Papyrus supports strategies where data is moved, where models are moved, and where numerical results are moved, but does not support mixed strategies where this choice varies from node to node. Papyrus Version 1.0 will remedy this by supporting optimal strategies as described in Section 4.

3. Version 0.9 of Papyrus supports clusters of workstations linked by high performance networks and by commodity networks, but cannot effectively support mixed networks, where nodes may have several connections, each with a different quality of service. Papyrus Version 1.0 will remedy this by supporting servers which can effectively schedule requests incorporating different qualities of service.

References

[1] T. Anderson, D. Culler, and D. Patterson, A Case for NOW (Networks of Workstations), IEEE Micro, Volume 15, pages 54-64, 1995.

[2] P. Chan and S. Stolfo, A Comparative Evaluation of Voting and Meta-Learning on Partitioned Data, In Proceedings of the Twelfth International

Conference on Machine Learning, Morgan-Kaufmann, San Francisco, California, pages 90-98, 1995.

[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, AAAI Press, 1999.

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

[5] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, ¿From Data Mining to Knowledge Discovery: An Overview, in Advances in Knowledge Discovery and Data Mining, edited U. M Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, AAAI Press/MIT Press, pp. 1-34, 1996.

[6] I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, International Journal Supercomputing Applications, Volume 11, pages 115-128, 1997.

[7] I. Foster and C. Kesselman, editors, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, San Francisco, 1999.

[8] D. Gannon and A. Grimshaw, Object-Based Approaches, in The Grid: Blueprint for a New Computing Infrastructure, I. Foster and C. Kesselman, editors, Morgan Kaufmann, San Francisco, pages 205-236, 1999.

[9] R. Gray, D. Rus, and D. Kotz, Transportable Information Agents, Technical Report PCS-TR96-278, Department of Computer-Science, Dartmouth College, 1996.

[10] A. Grimshaw and W. Wulf, The Legion Vision of a Worldwide Virtual Computer, Communications of the ACM, Volume 40, pages 39-45, 1997.

[11] R. L. Grossman, H. Bodek, D. Northcutt, and H. V. Poor, Data Mining and Tree-based Optimization, in the Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, AAAI Press, Menlo Park, California, page 323-326, 1996.

[12] R. L. Grossman, S. Bailey, and D. Hanley, Data Mining Using Light Weight Object Management in Clustered Computing Environments, In Proceedings of the Seventh International Workshop on Persistent Object Stores, Morgan Kauffmann, San Mateo, California, pages 237–249, 1997.

[13] R. L. Grossman, Supporting the Data Mining Process with Next Generation Data Mining Systems, Enterprise System Journal, 1998.

[14] R. L. Grossman, S. Bailey, A. Ramu, 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 System Technology, Volume 41, pages 589-595, 1999.

[15] R. L. Grossman et. al., Papyrus: A System for Data Mining on Clusters, Meta-Clusters, and Super-Clusters, in preparation.

[16] R. Guerin and H. Schulzrinne, Network Quality of Service, The Grid: Blueprint for a New Computing Infrastructure, I. Foster and C. Kesselman, editors, Morgan Kaufmann, San Francisco, pages 479-503, 1999.

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

[18] 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.

[19] H.Karagupta, E. Johnson, E. R. Sanseverino, B-H Park, L. D. Silvestre, and D. Hershberger, Scalable Data Mining from Vertically Partitioned Feature Space Using Collective Mining and Gene Expression Based Genetic Algorithms, KDD-98 Workshop on Distributed Data Mining, to appear.

[20] D. B. Lange, M. Oshima, Programming and Deploying Java(tm) Mobile Agents With Aglets(tm), Addison-Wesley, Reading, Massachusetts, 1998.

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

[22] World Wide Web Consortium, http://www.w3c.org.

[23] The Predictive Model Mark Up Language Version 0.9, The Data Mining Group, 1998, http://www.dmg.org.

[24] J. R. Quinlan, C4.5 Programs for Machine Learning, Morgan Kauffmann, San Mateo, California, 1993.

[25] A. E. Raftery, D. Madigan, and J. A. Hoeting, 1996. Bayesian Model Averaging for Linear Regression Models, Journal of the American Statistical Association, Volume92, pages 179–191, 1996.

[26] 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.

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

[28] B. Teitelbaum and T. Hanss, QoS Requirements for Internet2, submitted for publication.

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

[30] D. Wolpert, Stacked Generalization, Neural Networks Volume 5, pages 241-259, 1992.

[31] L. Xu, and M.I. Jordan, EM Learning on A Generalised Finite Mixture Model for Combining Multiple Classifiers, in Proceedings of World Congress on Neural Networks, Hillsdale, New Jersey, Erlbaum, 1993.