

# Towards Network-Aware Data Mining

**Srinivasan Parthasarathy**  
Computer and Information Science  
Ohio State University,  
Columbus, OH 43235  
srini@cis.ohio-state.edu

Distributed data mining algorithms executing on a shared network of workstations often suffer from unpredictable performance problems due to limited network resources that are being shared. We show that data mining algorithms, which have an approximate nature, can adapt to network-resource constraints. We argue that existing network monitoring and quality of service mechanisms are insufficient to support the needs of such applications. We then discuss the mechanisms (systems support) needed to support such network-aware applications. Finally, we describe the current status (work-in-progress) of the system under development and discuss some general issues involved in developing resource-aware data mining algorithms.

**Keywords:** Flexible QoS, Resource-Monitoring, Distributed Data Mining, Shared Cluster Computing

## 1 Introduction

As our ability to collect, store, and distribute huge amounts of data increases with advancing technology, discovering the knowledge hidden in these ever-growing databases has become a pressing problem. This problem referred to as data-mining, an effort to derive interesting conclusions from large bodies of data, is an interactive process. In fact, interactivity is often the key to facilitating effective data understanding and knowledge discovery. In such an environment response time is crucial because lengthy time delay between responses of two consecutive user requests can disturb the flow of human perception and formation of insight. However, extracting knowledge from these massive databases is a compute and data intensive process which makes the task of guaranteeing quick response times difficult. In order to solve this problem researchers have taken a two pronged approach. To minimize the I/O traffic involved in these applications researchers have evaluated the viability of using data reduction techniques such as discretization, wavelet transforms, and sampling, while sacrificing little in terms of result quality. Simultaneously to compute results faster,

researchers are turning to effective parallelization of existing data mining algorithms [19, 23, 3].

Modern-day enterprises usually contain a cluster of shared memory workstations connected by some (intra-enterprise) network. Such a cluster of shared-memory symmetric multi-processors (SMPs) can be a cost effective powerful computational resource. However, the performance achieved by parallel programs on a non-dedicated network of workstations is unpredictable and often leaves a lot to be desired. This is because the performance is affected by dynamic contention for processors, network links, and I/O resources. As an example, for programs using the transmission control protocol/internet protocol (TCP/IP for short), a small amount of contention can play havoc with performance, due to TCP/IP's inbuilt contention-avoidance mechanisms that can reduce communication rates drastically resulting in many idling processors.

One approach to deal with such resource contention is to police the allocation of such resources and having applications adapt to resource constraints. Policing resources requires appropriate mechanisms [12, 21], to arbitrate, allocate, and enforce resource reservations while providing feedback to applications regarding available resources, so that they may adapt. In this paper we focus on how this approach can be adopted for data mining applications when *network resources* are at a premium.

The availability of network monitoring services [5], and quality-of-service (QoS) aware network protocols [6, 4] to police network resources, suggests a natural solution. The problem with directly using existing QOS mechanisms is that they are based on some assumptions that do not hold in the data mining context. These mechanisms have largely been developed for constant bit-rate, low-bandwidth media flows in unreliable network protocols [13, 7]. Interactive data mining applications often exhibit *bursty* traffic patterns, operate on large *remote* datasets, and are typically implemented on top of *reliable* networking protocols. Therefore existing mechanisms are unlikely to suf-

fice. Also, QOS mechanisms for accessing large remote data stores has not been considered in the literature. For data mining applications which operate on tera-bytes of data, this is extremely important.

In this paper we make the following contributions. We demonstrate that data mining applications that are cognizant of network constraints can take advantage of knowing about these constraints and can adapt accordingly. We then consider what mechanisms are lacking and therefore necessary for application adaptability and improvement in overall application performance (on a non-dedicated network of workstations). This latter aspect is *work-in-progress*.

In the next section (Section 2) we sketch our distributed architecture and describe existing QOS work in the IP context, upon which we are building our network QOS-aware support. Then in Section 3 we present how two representative distributed data mining applications can adapt to fluctuations in network performance, motivating the need for effective systems support. Also in this section we pinpoint some of the unpredictable traffic patterns that argues for providing a more flexible Network QOS-aware interface than is currently supported for media applications. In Section 4 we describe the details of the system under development. In this section we describe an overview of the overall resource-aware system, then describe the intended QOS support for local networks and remote data access, including a very preliminary interface. In Section 5 we highlight some critical research issues in translating application-level QOS to system-level QOS. Section 6 documents relevant related work pertaining systems support for data mining applications. Finally we conclude in Section 7.

## 2 Background

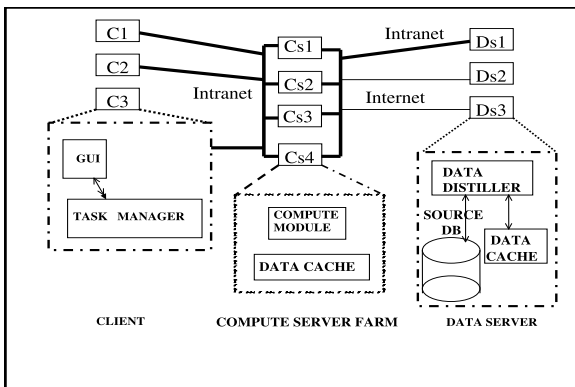


Figure 1. System Architecture

We envision an environment in which a cluster is shared amongst multiple users in a voluntary and cooperative

fashion rather than one which is centrally managed and operates on the single program executing at anytime constraint. The latter scenario typically suffers from a gross under-utilization of resources.

In this section we sketch (see Figure 1) the architecture of the distributed data mining system we have in mind. The design of our system took into account the interactivity and large datasets involved in mining applications. The design also takes into account the fact that in many of these applications it is often possible to provide succinct descriptions of the input required (e.g. “partition dataset X into 4 clusters” or “discretize the continuous attributes in dataset Y”). This enabled us to decouple task description from the actual data required by the task and is reflected in our decoupled architecture. This decoupling is important as it potentially enables different tasks to obtain data from multiple sources simultaneously.

Our architecture consists of the following logical components:

- **Clients (C#)** consisting of the GUI, a task manager that directs the mining process, and a local cache (not shown) of data/results of prior computations. It is responsible for the interacting with the data mining engine in terms of invoking, guiding and monitoring computations as well as visualization of the results.
- **Compute Servers (Cs#)**, each consisting of a task manager (not shown), a compute module, which is the core data mining engine, and a local data cache. The compute servers and the clients are typically interconnected with each other on a high bandwidth low latency interconnect (Intranet).
- **Data Server (Ds#)** consisting of a data distiller and the source database. The data distiller reads data from the database and performs appropriate data compaction transformations if required (more on this in Section 4.3). We envision that these data servers may be accessible via the local Intranet or via the commodity Internet (low bandwidth, high latency). Although the data servers in the figure are represented by a single box, we envisage that the data servers are based on a cluster architecture, and we take advantage of this assumption, when we describe QOS support for remote data access (Section 4.3).

The physical layout of these logical components depends on available resources. In a fully distributed mode the components, i.e. the client, each compute server and the data server are physically separated. When the data sets are relatively small and the client is relatively powerful, all components could be resident on the client. A hybrid model may combine the compute and data server operations keeping the clients distinct.

As mentioned earlier it is our intention to provide a set of mechanisms that can enable data mining applications to dynamically adapt to network (Intranet and Internet) resource constraints on such an architecture. Since most distributed mining applications are implemented on top of TCP/IP we next discuss existing mechanisms within the context of TCP/IP.

## 2.1 QoS in IP

We briefly review the current state-of-art in network QOS mechanisms within the context of IP. In IP, packets typically make their way from source to destination via a series of routers. Each individual router must decide which, of usually several packets, to forward as a function of the perceived QOS. The two most common approaches to QOS that have been proposed is the Integrated Services approach [6] and the Differentiated Services [4] approach.

The Integrated Services Approach, requires reservations to be placed by end points on a router by router basis. Each router is responsible for and treating for identifying and treating each application-flow separately according to individual QOS requirements. The Differentiation Services approach, limits the burden on routers by having only edge routers classify which packets should receive better services and then place the class label in the header. For instance, if a better quality service is to be provided to video-on-demand packets, the edge router will mark it accordingly. Then interior network nodes simply need to aggregate packets based on classes of service and ship them out according to their level of importance. This approach improves over the Integrated Services approach by simplifying the task of routers.

In addition to classifying and marking packets, edge routers may perform policing and shaping, to ensure that senders do not send too much high quality traffic too fast, and to handle bursty traffic, respectively. Policing is accomplished normally by a token bucket mechanism. The size or number of tokens in the bucket controls how quickly an application can send data. Policing is very useful for *enforcing* quality of service. Shaping is accomplished by smoothing out bursty traffic and is done to reduce packet loss.

Mapping these parameters (bandwidth, token depth) for media applications that generate a fixed amount of data (say 8Kb) per frame, at a fixed rate (say 15 frames a second) is pretty straightforward (bandwidth 120 Kbps, token depth 8Kb). The communication patterns associated with data mining applications, described in the next section, are significantly more complex, while the requirements often may be less stringent.

## 3 Application Adaptability

Data mining applications and the human user driving them, often have varying needs in terms of quality and performance. However, applications (and users) can often adapt to resource constraints by trading off result quality for performance (improved response time), when resources are constrained. In other words the result quality can be sacrificed in a controlled manner when resources are at a premium and response time is crucial. Below we give specific examples of how two applications, discretization and dataset clustering implemented on our architecture can adapt to constraints in network resources.

### 3.1 Discretization

In the process of growing a decision tree [17], the problem is to determine which leaf node to split and for each leaf node, which attribute to use for the decision at that node. Consider the computation at a node. While typically, a single attribute is used as the decision variable, one can well consider extensions to more than one base attribute (e.g.,  $X > 5 \wedge Y < 6$ ) as long as the decisions remain simple [15]. Limiting oneself to selecting base attributes pair-wise, the problem is to determine  $f(X_i, X_j)$  for all  $X_i$  and for all pairs  $X_i, X_j$ , where  $\{X_i\}$  are the attributes and  $f(X_i, X_j)$  measures the goodness of  $X_i, X_j$  as a decision attribute. This problem is referred to as 2-Dimensional Discretization and is computed by the following algorithm:

```

FOR each pair of attributes
  Step 1: Compute the probability density function (pdf)
          induced by the 2 base attributes.
  Step 2: Compute optimal discretization (based on
          some goodness function like entropy)
  Step 3: Score (entropy value) and save result
END FOR
Step 4: Sort results and display top scoring attribute-pairs
        and corresponding discretizations.

```

Next we consider how to map this algorithm on to our distributed architecture. Step 1 will typically be computed by the corresponding data server, since this step distills the pairwise attribute columns to a much smaller summary (pdf), that can then be economically transmitted to remote compute servers. Steps 2 and 3 are naturally computed by the local compute servers. Once all the results have been scored and saved, the final step (Step 4) can be executed on the client machine and displayed to the user.

### 3.2 Network Aware Discretization

This application can adapt to a bandwidth limited environment (as may be the case when the data servers are

Grid Size(MB)	Classification Error
0.00155	13.5%
0.00625 <i>KB</i>	12.8%
0.025	12.47%
0.1	12.22%
0.4	12.19%
1.55	12.15%
6.25	12.12%

**Figure 2. Network-Aware Discretization**

truly remote over the commodity Internet) in the following way. The pdf can be computed at different levels of granularity<sup>1</sup>. The finer the granularity the greater the accuracy of the results (lower the classification error). However, at finer granularities the amount of data that needs to be transmitted increases proportionally. We evaluated this tradeoff (results in Figure 2) for a pair of attributes from the LL dataset (described in [15]). From the figure we see that the most accuracy at the highest granularity (grid size = 6.25MB). However, the drop-off in accuracy at a significantly smaller granularity (grid size = 0.1MB) is just 0.1%. This demonstrates that this application can adapt to a bandwidth limited environment by trading off quality (classification error) for significant performance gains (reduction in communication). Note that the total amount of data that needs to be transferred from the data server(s) to the compute servers is equal to  $N(N + 1)/2 \times \text{gridsize}$  where N is the total number of attributes in the dataset. Therefore, the total savings in communication is extremely significant, and comes with a minimal sacrifice to quality. Also note that the top-scoring PDF's are transferred a second time from compute servers to the client (Step 4) resulting in further performance benefits (from working with smaller grid sizes).

### 3.3 Hierarchical Dataset Clustering

In [14], we define the similarity between two datasets (say  $D_i$ , and  $D_j$ ) to be a function of the difference between the set of associations ( $A_i$  and  $A_j$  respectively) induced by them ( $Sim(D_i, D_j) = f(A_i, A_j)$ ), weighted by the supports of each association. We also show how such measures of similarity can be used effectively for clustering homogeneous datasets. The basic structure of this clustering algorithm is shown below:

Step 1 is typically computed by the corresponding data server, since this step distills the datasets to a much smaller summary (association sets  $A_i$ ). The summaries are then

<sup>1</sup>The granularity of the pdf estimate is a function of the number of discrete locations at which the pdf is estimated, i.e., the finer the granularity the more the number of discrete locations.

Step 1: Compute associations sets  $A_i$  for each dataset.  
 FOR each pair of datasets  $D_i, D_j$   
 Step 2: Compute  $Sim(D_i, D_j)$  from  $A_i, A_j$   
 END FOR  
 REPEAT (initially treating each dataset as a cluster)  
 Step 3: Merge closest pair of clusters  
 UNTIL Desired number of clusters  
 Step 4: Display data clusters and similarity matrix

communicated to the compute servers. Steps 2 and 3 are naturally computed by the local compute servers. Once the dataset clusters have been computed, Step 4 can be executed on the client machine and displayed to the user.

### 3.3.1 Network Aware Dataset Clustering

TIE (MB)	Accuracy
36	100%
18	99.3%
9	99.0%
7.2	98.9%
5.4	98.4%
3.6	97.8%
1.8	91.5%

**Figure 3. Network-Aware Dataset Clustering**

In step 2 it is possible to sample the association sets, to limit the amount of information that is being communicated at a cost to accuracy. Table 3 shows the impact of using sampling (which affects the total information exchanged (TIE)) to compute the similarities amongst 12 datasets [14]. Clearly, when the total information exchanged (TIE) is more, the similarity metric is more accurate. However, the loss in accuracy when a tenth of the data is exchanged (3.6MB) is about 2%, which may be tolerated by the user if it does not affect the actual clustering of the datasets (which happens to be the case for this experiment). The merging procedure (step 3) also involves exchanging of complete or partial information (in the spirit of step 2) to compute (or estimate) the association sets for the merged clusters. Similar results, from trading off quality for communication efficiency, was observed for this step as well.

### 3.4 Other Applications

Classical parallel implementations of various data mining tasks where the data is either centralized or distributed and there is no notion of a separable data server can also adapt to system resources. For association mining [1] researchers have described I/O sensitive techniques that:

trade off disk traffic [18] or communication [23] for extra computation (it is important to realize that quality need not always be sacrificed); trade off disk traffic for quality [22]. Since sequence mining is essentially association mining over temporal data, the same trade-offs can be made for it as well. The same is true for clustering algorithms as well. Recently, we described methods [16] by which the *Expectation Maximization* algorithm can adapt to various resource constraints. We described how this algorithm: can adapt to bandwidth limited environments via lossy and lossless compression; can adapt to available cache and memory sizes via program and data transformations, and can adapt to computational resource constraints via appropriate program transformations.

### 3.5 Traffic Patterns

In this section we isolate the traffic patterns of distributed data mining algorithms in general while paying particular attention to the two applications we have described. More importantly we identify how these applications differ from typical media applications. As pointed out earlier media applications usually have an easy to specify communication pattern. The communication patterns of data mining applications differ in the following respects:

- **Unpredictable Message Sizes:** The amount of information communicated depends largely on the data characteristics, and input parameters (or user interaction). In some cases the exchange of information is minimal and in yet others it is a large amount. For instance, between Step 1 and 2 of the hierarchical dataset clustering algorithm, the association sets are communicated from the remote data server(s) to the compute cluster. The number of actual associations in a given association set is completely dependent on the characteristics of the dataset<sup>2</sup>, as well as on the input parameters (minimum support), and is therefore difficult to predict in advance. Distributed versions [3] of the Apriori [1] algorithm that exchange candidate itemsets also exhibit this property. Here, the number of candidate itemsets, and therefore the amount of information communicated, depends on the characteristics of the dataset and input parameters.
- **Bursty Communication:** As we see from the algorithm descriptions above, data mining applications may compute for a bit, then communicate information, and then compute again. In many applications (such as discretization; when multiple pdf's are mapped to the same compute server) communication (receiving pdf's from data servers) can be overlapped

<sup>2</sup>It depends on the average number of items per transaction, total number of items in the dataset, and actual associativity of the dataset.

with computation (computing the optimal discretization on the pdfs that have already been received). In other cases computation ceases until communication completes. This results in bursty communication patterns that can overwhelm network resources, especially in the case of data intensive applications like data mining, resulting in poor performance.

- **Varied Communication Models:** In media applications the normal communication model is a simple synchronous, and continuous producer-consumer or one-on-one streaming communication pattern. Data mining applications exhibit a range of communication patterns such as synchronous, asynchronous, pairwise (one-on-one) and one-many (broadcast) and many-one (slave-master).
- **Dependency on Reliable Communication:** Data mining applications by and far rely on reliable network protocols unlike media applications. Communication in such applications is typically achieved via reliable protocols such as TCP/IP. TCP's flow control and congestion control mechanisms while critical to the effectiveness of TCP in shared networks have the unfortunate consequences of making TCP traffic sensitive to the loss of individual packets.

The above issues notwithstanding data mining applications are typically more *flexible* than media applications and can often place slightly less stringent QoS requirements on the network resources. For example in video-on-demand applications if a certain frame rate cannot be guaranteed, then the quality fallout may be unacceptable. On the other hand in data mining applications, there is often, and specifically in the case of the two examples we outlined above, no strict bandwidth requirements (as in requiring X bits per second). Therefore, the system can be more flexible in terms of bandwidth guarantees, and admission control. As a result even when the traffic patterns are bursty, mining applications are conducive to fairly aggressive shaping policies. The caveat here is that some mining applications, especially active mining applications, that operate on time-varying data might require strict bandwidth, and anti-shaping guarantees. Therefore, data mining applications can benefit *from a system that can support both conventional (strict) and more flexible classes of QoS.*

## 4 System Details

In this section we consider the details of the system under development to support network-aware data mining. We first briefly overview how we envision our system will process and control resource reservations. We then discuss the flexible QoS classes that we plan to support, for data

mining applications, within the local compute cluster. We then describe how we support QOS for remote data access.

## 4.1 Overview of the System

Static information available to the system includes the type of connection within the compute cluster (peak bandwidth, minimum latency), and similar information for connections to the data servers. Dynamic information available to the system include current resource utilization, and future resource utilization (from application reservations). At any given point the network *resource scheduler* can receive a resource reservation request from a client. The scheduler can also simultaneously receive similar information from other clients. Based on all the new requests, existing client priorities (briefly described in the next section) and the available resources at the cluster, the scheduler needs to decide on the optimal set of services to be scheduled. Depending on the optimization function (some combination of maximizing throughput of the system, best response time for prioritized clients, and guaranteeing QOS) the scheduler needs to derive the final set of resources (and corresponding QOS) that can be reserved. Once the scheduling decision is made, the admitted client applications are notified of the resources that are available and the corresponding QOS available. The scheduler also informs the *resource enforcer* about the admitted jobs which in return enforces the respective QOS for the clients. Based on this information, especially if the reserved resource is not at the desired QOS requested by the client, the application can adapt to the available service from the cluster. Once the reservation related to a particular resource is no longer required, the resource is released back to the cluster. We next describe the QOS interface for the compute cluster and the QOS interface for remote data access.

## 4.2 QOS for Local Networks

The QOS classes that we intend to support for data mining applications are *short low latency*, *premium (2)*, *best-effort* and *0-priority*. *Short low latency* messages are for short high priority messages that can be used for sending signals or for collective synchronization operations like barriers. *Premium messages* can be used for high priority messages that are too large to be supported by the low latency class of service. Based on the discussion presented in Section 3.5 we plan to support two forms of premium messages, i.e., *with and without bandwidth guarantees*. Data mining applications, requiring a strict QOS requirement, can use the former class (with bandwidth guarantees). Other applications, where the bandwidth requirements are not as stringent (or unknown), can use the more flexible premium message class (without bandwidth

guarantee) while still getting a reasonable level of QOS. Under the latter class, the system also has the flexibility (within moderation) to police and shape according to dynamic network information. *Best effort messages* can be used to indicate which messages do not require QOS service and can therefore be aggressively shaped or policed by the system. *0-priority* messages are messages that can be dropped when contention in the system becomes an issue. Such messages can be used in data mining in applications where the arrival or non-arrival of a message does not affect the produced results. For instance in association rule mining, pruning of candidate itemsets can improve the search speed [1], however, if there is heavy contention in the network then the information required by the pruning algorithm may be delayed, thus delaying the overall algorithm. If instead one were to send all pruning information using a one-way asynchronous 0-priority message, then if there is heavy contention this information would not get through else it will. The algorithm can tolerate this message loss, with no loss to result quality at the cost of extra computation.

There are other parameters involved when placing a network resource reservation. The *bandwidth* parameter is required, for one class of premium messages, and it reflects the amount of information that needs to be sent per unit time. The *total message size* (if known or if it can be estimated<sup>3</sup>), can allow the system to compute the projected reservation time based on current system state. This is useful for admitting other reservations (much like how restaurant reservation systems work).

There is also a need to provide mechanisms which would allow applications to monitor reservation requests. If the requested class of service is unavailable the system will return the next best available class of service that it can currently guarantee. The application can query the reservation, and if the desired QOS level is not available, then it can decide how to adapt. A sample piece of code below describes how a programmer could take advantage of the current interface.

```

QOS_OBJECT *QO = QoS_adv_reservation(PREMIUM1, bw, message_sz,...)
/* ... other information such as source, destination etc. */
.....
QOS_RESERVATION_DATA *QOR = QO->get_reserved_info()
.....
IF (!QOR->got_requested_reservation())
    Read what we got
    Adapt accordingly
    QO->Send(*msg_ptr);
ELSE
    Continue as per plan
    QO->Send(*msg_ptr);
ENDIF
.....
QO->relinquish(); /* required for PREMIUM1 service, implicit for others */

```

<sup>3</sup>We have pointed out in Section 3.5 that the total message size is difficult to guess apriori in many data mining applications. However, in some cases this information can be estimated, either via sampling or from historical performance data, slightly in advance of when the message is actually sent.

The code first places an asynchronous advance QOS request (in the spirit of asynchronous I/O requests) with the desired class of service (*PREMIUM1*), the desired bandwidth (*bw*) and the estimated size of the message (*message\_sz*). Placing advance reservations is recommended especially if requesting premium or low-latency class of service as the system has more time to make a decision. Such requests will return a communicator object handle (*QOS\_OBJECT*), along with the exact QOS that will be guaranteed. The application can view the guaranteed reservation (using the *get\_reserved\_info()* function). This information can be used by the application if it needs to adapt to a lower than expected guarantee of service. The *Send()* function is used to send the corresponding message(s). Relinquishing reservations are implicitly handled by the system once the message has been sent, for all classes of service except premium messages with bandwidth requirements. For this class of service the reservation will have to be relinquished explicitly (as shown in the code fragment using the *relinquish()* function).

### 4.3 QOS for Remote Data Access

QOS over the commodity Internet is difficult to guarantee. Perhaps the most challenging aspect of a remote storage server design lies in the bandwidth barrier represented by the wide area link protocols. Theoretically the bandwidth on the (wide-area) Gigabit Ethernet is comparable to that of a local (campus-wide) cluster. However, the well documented overhead of conventional TCP/IP protocol stacks limits the link data rate to values around 50 MB/s, no matter how fast the wire.

To overcome this problem, we have adopted a parallel communication approach in the spirit of [10] and [2]. The basic idea is to communicate by striping data over parallel TCP/IP connections. Our scheme improves on the above in that each of the connections can use distinct endpoint nodes since we envision that each individual data server is basically a cluster architecture. Basically, by taking advantage of the presence of clusters on both ends of the geographical link, we get around the bottleneck by effectively parallelizing the TCP/IP protocol processing. The scheme scales up with the number of nodes used for each cluster, up to the physical limit of the intervening link. An additional software layer is in charge of striping and de-striping on the two ends, while taking advantage of the fast user-level communication available on the local clusters.

The above approach also enables applications to specify a quality of service in terms of the number of striped connections. The quality of service can be controlled by increasing or decreasing the number of striped connections in response to changes in application requirements. The two classes of service that we support here are *best effort* and *premium* (with striping parameters). When requesting

the premium class the application can specify the striping parameters that will correspond to the number of striped connections per node as well as the number of nodes involved for each cluster. The network scheduler will determine whether the request can be met (which will involve communicating with its corresponding unit at the data server end).

Table 1 presents some preliminary experiments in which we evaluate the the impact of the above approach. The first column (#CS), represents the number of compute server nodes involved in the experiment. The second column (#RS) represents the number of remote data server nodes involved in the experiment. The third column (#PC), represents the number of parallel communication channels between the remote server(s) and compute server(s). The second row of the table represents an experiment involving four parallel connections between one compute server node and one remote server node. Finally, the fourth column represents the total time for the data transfer. This includes the time for communication, de-striping, and storing on the local file system. We assume that the data is already striped on the remote servers.

#CS	#RS	#PC	Total Time (s)
1	1	1	26.4
1	1	4	24.1
1	2	4	22.6
1	4	4	21.1
2	1	4	22.1
2	2	4	21.3
2	4	4	20.1
4	1	4	21
4	2	4	20.3
4	4	4	18.6

**Table 1. Evaluating Striping for Remote Data Access**

Our experimental set-up consisted of compute servers located in the Ohio-State University. The data servers were located at Rochester, NY. The communication was over the commodity Internet. In all the experiments 16MB of data was transmitted. The overall performance was severely inhibited by the fact that compute cluster has to make connections to the outside world via a single gate-way node. In spite of this limitation we observed encouraging results. On going from 1 (row 1 (*best-effort*)) to 4 (row 2) parallel connections for one compute node and one data node we see that there is a 10% improvement in performance. As we pointed out earlier, increasing both the number of remote server nodes (rows 2-4), and the number of compute nodes (rows 5-11) resulted in further improvements in performance (10-35%).

Another form of QoS that we plan for remote data access is *data-preprocessing time* at the data-server end. As we pointed in Section 3, many data mining applications can use data reduction operations like wavelet transforms, pdf generation, various time/frequency transforms, sampling, etc., to reduce the amount of data that needs to be transferred thereby ensuring that even with lower bandwidth reservations the reduced data can be obtained in a timely fashion. However, these data reduction operations can be computationally intensive. In order for the data reduction operations to be effective the sum of the time taken to compute the reduction and the time taken to send the reduced data must be less than the time to send the original data. Therefore, the application needs to place an appropriate QoS reservation for computational resources [12, 21] on the data server.

#### 4.4 Current Implementation Status and Issues

We are clearly at the very early stages of implementing the ideas presented in the previous few sections. We are in the process of implementing the interface for local area networks based on the Differentiated Services-based system, where the QoS class labels (premium, short-low latency etc.) are placed in packet headers, so that policing can be enforced and shaping of messages can be carried out where appropriate. After the basic interface has been implemented we will implement the meat of the system, viz., the resource scheduler. For remote data access, we have implemented the quasi-QoS protocol as outlined above and are evaluating it.

One interesting research issue we are investigating is to identify a mechanism that can handle premium messages with bandwidth guarantees effectively. One possible solution we are investigating is to estimate and dynamically adjust the optimal depth of the token bucket at the edge routers. We are evaluating the viability of estimating this information using current premium bandwidth reservation information coupled with dynamic network performance data.

### 5 Translation of user level QoS to System-level QoS

To provide a particular level of user/application level QoS, there is a need to translate the user QoS to the underlying system-level QoS presented in the previous section. This translation will naturally have to be done in an application-specific manner. Below we discuss the research issues involved in this translation process.

For data mining applications, the user's requirements are mostly specified in terms of performance and quality of results (Table 2 and Table 3). Performance is usually

described as the amount of time that is allowed to fulfill the user's request, i.e., *response time*. Based on the algorithm that is involved, this response time specification can be translated into the deadline for completing the requested operation. This "execution deadline" information can be used by the scheduler to *prioritize* among competing clients requesting network resource reservations. To specify the quality requirement, the user can typically specify the amount of error (E) that is tolerated. The relationship (network resources =  $f(E)$ ) between the error tolerance and the required network resources is application-specific. This relationship can often via static or run-time profiling on sampled data be estimated. For instance on knowing the maximum tolerated classification error for building a decision tree one can, after sampling the data at runtime, estimate the amount of data that needs to be communicated in order to guarantee this error tolerance requirement. By knowing the amount of data that needs to be communicated, and in conjunction with response time requirements of the client, one can compute the appropriate bandwidth requirements needed, which can then be reserved.

## 6 Related Work

Several systems have been developed for distributed data mining. The JAM [20](Java Agents for Meta-learning) and the BODHI [11] system assume that the data is distributed. They employ local learning techniques to build models at each distributed site, and then move these models to a centralized location. The models are then combined to build a meta-model whose inputs are the outputs of the various models and whose output is the desired outcome. The Kensington [9] architecture treats the entire distributed data as one logical entity and computes an overall model from this single logical entity. The architecture relies on standard protocols such as JDBC to move the data. The Id-Vis [21] architecture is a general-purpose architecture designed with single data mining applications in mind to work with clusters of SMP workstations. Both this system and the Papyrus system [8] are designed around data servers, compute servers, and clients as is the system presented in this work. The Id-Vis architecture explicitly supports interactivity through the interactive features of the Distributed Doall programming primitive. However, the interactions supported are limited to partial result reporting and bare-bones computational steering. Our work is complementary to the above distributed data mining systems. Their focus is on how to build data mining systems or specific data mining applications when the data and processing capacity is distributed. Our focus is on how to build systems support for resource-aware data mining.

## 7 Conclusions

Many data mining algorithms have unpredictable performance on shared computational environment, due to resource constraints. In this paper we present preliminary results showing how data mining algorithms can effectively adapt to network resource constraints if proper systems support and a flexible interface to control and enforce network resource acquisition were available. We have argued that existing QOS mechanisms for media applications are not appropriate for data mining algorithms since the communication needs and application properties are very different. We have then outlined a desired (wish-list) set of QOS mechanisms that ought to be supported in order that adaptable data mining algorithms can monitor and acquire resources to get predictable and desired performance and have described how we are attempting to achieve this goal.

## References

- [1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *20th VLDB Conf.*, Sept. 1994.
- [2] S. Bailey, E. Creel, R. Grossman, S.Gutti, and H. Sivakumar. A high performance implementation of the data space transfer protocol. In *Workshop on Parallel and Distributed KDD Systems*, 1999.
- [3] D. Cheung, J. Han, V. Ng, A. Fu, and Y. Fu. A fast distributed algorithm for mining association rules. *4th Intl. Conf. Parallel and Distributed Info. Systems*, Dec. 1996.
- [4] B. et al. An architecture for differentiated service. In *IETF Network Working Group, RFC2475*, 1998.
- [5] B. L. et al. A resource query interface for network-aware applications. In *HPDC*, 1998.
- [6] Z. et al. A new resource reservation protocol. In *IEEE Network*, 1993.
- [7] I. Foster and C. Kesselman. The grid: Blueprint for a future computing infrastructure. In *Morgan Kauffman Publishers*, 1999.
- [8] R. Grossman, S. Bailey, S. Kasif, D. Mon, A. Ramu, and B. Malhi. Design of papyrus: A system for high performance, distributed data mining over clusters, meta-clusters and super-clusters. In *Proceedings of Workshop on Distributed Data Mining, alongwith KDD98*, Aug 1998.
- [9] Y. Guo, S. Rueger, J. Sutiwaraphun, and J. Forbes-Millot. Meta-learning for parallel data mining. In *Proceedings of the Seventh Parallel Computing Workshop*, 1997.
- [10] J.D.Touch. Parallel communication. In *INFOCOMM*, 1993.
- [11] H. Kargupta, I. Hamzaoglu, and B. Stafford. Scalable, distributed data mining using an agent based architecture. In *KDD*, Aug 1997.
- [12] C. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *Morgan Kauffman Publishers*, 1999.
- [13] K. Nahrstedt, H. Chu, and S. Narayan. Qos-aware resource management for distributed multimedia applications. In *Journal on High-Speed Networking*, 1998.
- [14] S. Parthasarathy, and M. Ogihara. Clustering Homogeneous Distributed Datasets. In *Fourth Practical Applications of Knowledge Discovery and Data Mining (PKDD)*, 2000.
- [15] S. Parthasarathy, R. Subramonian, and R. Venkata. Generalized discretization for summarization and classification. In *preprint*, Jan. 1998.
- [16] S. Parthasarathy, and R. Subramonian. Adaptive EM-Clustering. Technical Report OSU-CISRC-12/00-TR26, Ohio State University, Nov. 2000.
- [17] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 5(1):71–100, 1996.
- [18] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *21st VLDB Conf.*, 1995.
- [19] J. Shafer, R. Agrawal, and M. Mehta. Sprint: A scalable parallel classifier for data mining. In *VLDB*, pages 544–555, 1996.
- [20] S. Stolfo, A. Prodromidis, and P. Chan. Jam:java agents for meta-learning over distributed databases. In *KDD*, Aug 1997.
- [21] R. Subramonian and S. Parthasarathy. A framework for distributed data mining. In *Proceedings of Workshop on Distributed Data Mining, alongwith KDD98*, Aug 1998.
- [22] M. J. Zaki, S. Parthasarathy, W. Li, and M. Ogihara. Evaluation of sampling for data mining of association rules. In *7th Intl. Wkshp. Research Issues in Data Engg*, Apr. 1997.
- [23] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New parallel algorithms for fast discovery of association rules. *Data Mining and Knowledge Discovery: An International Journal*, to appear, Dec. 1997.