Systems Support for High-Performance Scientific Data Mining

Gagan Agrawal  Ruoming Jin  Raghu Machiraju  Srini Parthasarathy
Department of Computer and Information Sciences
Ohio State University, Columbus OH 43210
{agrawal,jinr,raghu,srini}@cis.ohio-state.edu

Abstract

Initial efforts on applying mining techniques on scientific simulation datasets has demonstrated that 1) the algorithms and the algorithmic parameters that give desirable results for a given problem and dataset need to be determined through an iterative process, and 2) scientific simulation datasets are typically very large, and therefore, scalable implementations are required.

Based upon these observations, we believe that there is a need for a tool that will allow us to rapidly create scalable implementations for a variety of mining algorithms. We have been developing a system called FREERIDE (FRamework for Rapid Implementation of Datamining Engines) that provides this functionality. It allows a parallel mining algorithm to be specified using a high-level API, and achieves both distributed memory and shared memory parallelization, besides enabling efficient processing of disk-resident datasets. FREERIDE has been successfully demonstrated on a number of standard mining algorithms, including apriori association mining, k-means clustering, k-nearest neighbor search, and RainForest based decision tree construction.

We have recently initiated an effort to apply FREERIDE for mining scientific simulation datasets. Particularly, we are focusing on a feature detection, tracking, and mining framework that some of us have been developing with aid of application experts in computational fluid dynamics. This paper describes the FREERIDE system and our initial ideas on applying it for the feature-mining framework.

1 Introduction

In recent years, data mining techniques have been extended beyond traditional domains such as business and marketing and have been applied to scientific and biomedical problems [12, 3] However, data mining techniques have not yet achieved the same level of success in scientific domains that they have achieved in commercial domains.

This paper describes our on-going effort on developing system support for scalable data mining, which, we believe, can significantly improve the applicability of mining in scientific areas. Our work is based upon the following observation. Initial work on data mining for scientific problems has shown that algorithms and algorithmic parameters that can yield desirable results on a given problem and dataset need to be determined through an iterative process. At the same time, datasets from scientific simulations are often very large. Therefore, reasonable response time can only be achieved if the algorithms are implemented on parallel machines and can efficiently process disk-resident datasets. Thus, there is a need for tools that will allow us to rapidly create parallel and scalable implementations for a variety of mining tasks. Further, we believe that it is important that the algorithms can be easily ported across a variety of parallel platforms and the parallel implementations are easy to maintain and modify.

Over the last two years, we have developed a middleware system called FREERIDE (FRamework for Rapid Implementation of Datamining Engines) [9, 8, 10, 11]. The basis for this system is a set of techniques for distributed memory as well as shared memory parallelization that apply across a variety of popular data mining algorithms. We believe that this system can be particularly useful while mining on scientific data. This is because our understanding of what kind of algorithms can yield desirable results is relatively lower for these areas. At the same time, the datasets from scientific simulations can be very large, making scalability an even more important factor.

This paper gives an overview of the FREERIDE framework, and then describes our on-going work on parallelizing feature tracking and feature-based mining algorithms using this framework. The feature-based mining approach we focus on has been developed by Machiraju and his associates [18]. We have initiated effort on parallelizing the steps involved in feature detection and tracking, as well as the mining algorithms associated. This paper describes our initial ideas in this direction.

The rest of this paper is organized as follows. The core for our work, the FREERIDE system for rapidly creating
scalable implementations is described in Section 2. Parallelization of a scientific mining approach, i.e. the feature and shaped based paradigm, is described in Section 3. We summarize our work in Section 4.

2 FREERIDE: Middleware for Parallel Data Mining Implementations

Motivated by the difficulties in implementing and performance tuning individual data mining algorithms, Agrawal’s group has developed a system called FREERIDE (Framework for Rapid Implementation of Datamining Engines) [9, 10]. FREERIDE is based upon the observation that parallel versions of several well-known data mining techniques share a relatively similar structure. We have carefully studied parallel versions of apriori association mining [1], Bayesian network for classification [2], k-nearest neighbor classifier [4], artificial neural networks [4], and decision tree classifiers [14]. In each of these methods, parallelization can be done by dividing the data instances (or records or transactions) among the nodes. The computation on each node involves reading the data instances in an arbitrary order, processing each data instance, and performing a local reduction. The reduction involves only commutative and associative operations, which means the result is independent of the order in which the data instances are processed. After the local reduction on each node, a global reduction is performed. FREERIDE exploits this commonality to support a high-level interface for developing parallel implementations. The target environment are clusters of SMPs, which are emerging as a cost-effective, flexible, and popular parallel processing configuration. Clusters of SMP workstations, where each workstation has an attached disk farm, offer both distributed memory or shared-nothing parallelism (across nodes of the cluster) and shared-memory parallelism (within a node).

2.1 Runtime Techniques

We now discuss the runtime techniques implemented in the middleware for I/O performance enhancement, shared-nothing parallelization, and shared-memory parallelization. Shared-nothing Parallelization: As we described earlier, the structure of parallel algorithms for common data mining techniques makes shared-nothing parallelization relatively simple. After data has been distributed (de-clustered) between different nodes, each node can execute initial processing and local reduction functions on data items it owns. After each invocation of local reduction function, local copies of reduction objects on each node are broadcast to all other nodes, and local copies of reduction objects from all other nodes are received on each node. This communication is facilitated by the middleware. After the communication phase, global reduction function is invoked on each node. I/O Performance Enhancement: Efficient I/O is critical for mining over disk resident datasets. For this purpose, a preprocessing phase, known as task planning, is used to create a schedule for each invocation of local reduction. Task planning is somewhat similar in nature to the inspector used as part of the inspector/executor paradigm for improving communication performance on distributed memory machines [15] or performing runtime parallelization [16]. A schedule or task plan specifies the order in which data items locally owned are accessed and processed. The task plan is created to minimize disk seek time on each disk, while maximizing parallelism in accessing each disk. Within each disk, the data items that are within the subset of data to be processed are accessed in the order in which they are stored. For achieving high bandwidth retrieval, the dataset owned by each node is partitioned into a set of chunks. A chunk is typically of the size of a small number of disk blocks and is the unit of I/O. During task processing, I/O and processing is overlapped as much as possible. This is done by maintaining explicit queues for both kind of operations (data retrieval and data processing) and switching between them as required.

Runtime Shared-memory Parallelization: We now describe the framework we have implemented for efficiently using multiple processors available on each node of a SMP workstation for data mining applications. As we discussed earlier, in most of the parallel data mining algorithms, local reductions on data items are independent operations, except for race conditions in updating the same reduction object. In avoiding these race conditions, the main complication is that the particular element(s) in the reduction object that need to be modified after processing a data item is not known until after performing the computation associated with the data item.

The problem of SMP parallelization of mining algorithms has many similarities with the problem of parallelizing aggregation algorithms. Two obvious approaches are, full replication or the two phase algorithm, and full locking or the simple parallel aggregation [17]. In the full replication approach, each processor can update its own copy of the reduction object and these copies are then merged together later. In the full locking approach, one lock or latch is associated with each aggregated value.

We have observed that supporting a large numbers of locks results in overheads of three types. The first is the high memory requirement associated with a large number of locks. The second overhead comes from cache misses. Consider an update operation. If the total number of elements is large and there is no locality in accessing these elements, then the update operation is likely to result in two cache misses, one for the element and second for the lock. This cost can slow down the update operation significantly on modern machines with deep memory hierarchies. The third overhead is of false sharing. In a cache-coherent shared memory multiprocessor, false sharing happens when
two processors want to access different elements from the same cache block. In the full locking scheme, false sharing can result in cache misses for both reduction elements and locks.

To overcome these overheads, we have designed two new schemes for parallelizing mining algorithms. These techniques are, optimized full locking, and cache sensitive locking [10].

**Optimized Full Locking:** Optimized full locking scheme overcomes the large number of cache misses associated with full locking scheme by allocating a reduction element and the corresponding lock in consecutive memory locations. By appropriate alignment and padding, it can be ensured that the element and the lock are in the same cache block. Each update operation now results in at most one cold or capacity cache miss. The possibility of false sharing is also reduced.

**Cache-Sensitive Locking:** Another techniques we have developed is cache-sensitive locking. Consider a 64 byte cache block and a 4 byte reduction element. We use a single lock for all reduction elements in the same cache block. Moreover, this lock is allocated in the same cache block as the elements. So, each cache block will have 1 lock and 15 reduction elements. Cache-sensitive locking reduces each of three types of overhead associated with full locking.

### 2.2 Preliminary Experimental Data

We now present experimental results demonstrating parallelization on both shared memory and distributed memory configurations.

For distributed memory parallelization, we present results from apriori association mining, k-means clustering, and k-nearest neighbor search. The experiments were conducted on a cluster of workstations. We used 8 Sun Microsystems Ultra Enterprise 450’s with 250MHz Ultra-II processors. The datasets we used were all disk-resident, and were 3 GB, 2 GB, and 2 GB for apriori, k-means, and k-nearest neighbors, respectively.

The results are presented in Figure 1. To focus on speedups, we only give normalized performance numbers, i.e. the execution time with 1 processor is normalized to 1. As can be seen from the figure, these three applications all achieve high-speedups. The speedups on 8 processors are 7.2, 6.9, and 7.7 for apriori, k-means, and k-nearest neighbor search, respectively. More detailed experimental data is available from a recent publication [9].

We have evaluated these parallelization techniques for a number of mining algorithms [10]. Because full locking consistently performed below optimized full locking and cache-sensitive locking, we only show results from full replication, optimized full locking, and cache-sensitive locking.

Figure 2 shows the performance of apriori association mining as the support level is decreased, with parallelization on 4 threads. Decrease in support level increases the size of the reduction object. Initially, full replication gives the best performance, but with increasing memory requirements, locking based schemes outperform replication. Moreover, after a certain point, cache-sensitive locking clearly outperforms optimized full locking. The performance of full locking was consistently significantly below the performance of the other two locking schemes. This study shows that each of full replication, optimized full locking, and cache-sensitive locking can outperform each other depending upon the size of the reduction object and machine parameters like available cache and memory.

Figure 3 presents results from parallelization of decision tree construction. We created 5 different versions. The first three simply use full replication, optimized full locking, and cache-sensitive locking, respectively. The last two versions combine locking and replication. We use replication for categorical attributes and for upper levels of the tree and locking for numerical attributes at the lower levels of the tree. The results show that combining replication with cache-sensitive locking gives the best performance on 2, 4, and 8 threads.

Overall, our work in this area has shown that we can choose and combine from a set of techniques to parallelize different mining algorithms, with different parameters, and on different machine configurations.

### 3 Parallel Feature Mining Using FREERIDE

In this section, we outline our on-going work on using the FREERIDE framework for scientific data mining. Par-
particularly, we are focusing on a feature mining approach. Initially, we describe this feature-mining approach, and then describe how FREERIDE can be used for its parallelization.

3.1 Feature Mining Paradigms

We describe our two feature-mining paradigms below. More description can be found in [18]. Our approach was shaped as a response to the large data visualization problem [13]. We extract and use volumetric regions to represent features. Voxels and their aggregations consume memory; however, they allow many operations to be conducted robustly. Both of these paradigms essentially consist of six steps that analyze features. It should be noted that at least three of these steps are common to both. The two paradigms exist to allow for the analysis of features with both local and global definitions. A systematic approach to feature detection can aid immensely in providing analysis tools to computational researchers.

3.1.1 Classify-aggregate Paradigm

This paradigm employs purely a local approach. The most important step in this paradigm is point classification. The classify-aggregate paradigm is best suited for analyzing features with local definitions and extents. Features like shocks arising in fluid dynamics simulations are ideal candidates for this paradigm. The realization of this paradigm requires the following operations in sequence:

1. Detection by application of a local sensor at each point in the domain
2. Binary classification (verification) of point based on some criteria
3. Aggregation of contiguous regions of like-classified points
4. Denoising to eliminate aggregates that are of insufficient extent, strength, and so on
5. Ranking based on feature or region saliency
6. Tracking of features

This approach identifies individual points as belonging to a feature and then aggregates them into regions. The points are obtained from a tour of the discrete domain and can be in many cases the vertices of a physical grid. The sensor used in the detection phase and the criteria used in the classification phase are physically based point-wise characteristics of the feature. Vertices can also be detected through the application of this paradigm. However, it is often the case that false positives are generated.

The next step is to denoise (filter) and rank the ROIs systematically. The denoising step eliminates regions which are too small or deemed insignificant. The ranking process should not accord significance to features that are weak or of small spatial extent. Multiscale methods if used check the persistence of the regions across scales. Ranking is essentially a sorting algorithm. The size, or a functional quantity (swirl in case of vortices), or integral value of swirl over all elements of the region.

3.1.2 Aggregate-classify Paradigm

We can best incorporate the global information needed to define a feature like a vortex into our second feature detection paradigm, the aggregate-classification approach. Aggregate-classification follows a somewhat different sequence of operations:
1. Detection by application of a local sensor at each point in the domain
2. Aggregation of contiguous regions of probable candidate points
3. Binary classification (verification) of each aggregate based on some criteria
4. Denoising to eliminate aggregates that are of insufficient extent, strength, and so on
5. Ranking based on feature saliency
6. Tracking of features

This approach identifies individual points as being probable candidate points in a feature and then aggregates them. The classification algorithm is applied to the aggregate using physically based regional criteria to determine whether the candidate points constitute a feature. A cheaper and efficient classification operator often suffices for this paradigm. We recently reported an efficient detection technique [6] that quickly identifies core candidates. Our technique segments candidate core regions by aggregating all probable points identified from the detection phase. We then classify (or verify) these candidate core regions based on the existence of swirling streamlines surrounding them [7]. Thus, false positives can be eliminated.

3.2 Parallelization Using FREERIDE

We have initiated work on parallelization of the above set of algorithms using FREERIDE. We have observed that many of the steps in both frameworks involve generalized reductions, and therefore, FREERIDE is well suited for parallelizing them. However, feature tracking involves significantly different computations. We are currently investigating how FREERIDE could be extended for supporting them.

4 Summary

This paper has described our on-going work on a new project addressing the scalability and programmability needs for scientific data mining. Our basic premise is that there is a need for a tool that will allow us to rapidly create scalable implementations for a variety of mining algorithms. We have been developing a system called FREERIDE (FRamework for Rapid Implementation of Datamining Engines) that provides this functionality. It allows a parallel mining algorithm to be specified using a high-level API, and achieves both distributed memory and shared memory parallelization, besides enabling efficient processing of disk-resident datasets.

This paper has presented the FREERIDE system and our initial ideas on applying it for the feature-mining framework. Particularly, we are focusing on a feature detection, tracking, and mining framework that some of us have been developing in conjunction with application experts in computational fluid dynamics.

References