High-level Interfaces and Abstractions for Grid-based Data Mining

Gagan Agrawal

Department of Computer and Information Sciences
Ohio State University

(joint work with Ruoming Jin, Liang Chen, Xiaogang Li, Leo Glimcher, Ge Yang, Xuan Zhang)
Our understanding of what algorithms and parameters will give desired insights is often limited.

The time required for creating scalable implementations of different algorithms and running them with different parameters on large datasets slows down the data mining process.
A data mining application in a grid environment -
- Needs to exploit different forms of available parallelism
- Needs to deal with different data layouts and formats
- Needs to adapt to resource availability

We should be targeting users who are used to
programming systems like matlab / SQL / …
What do we need?

- The ability to exploit different forms of architectures/parallelism without hard-coding
  - Distributed memory, shared memory, combination of two
- Self adaptation based upon resource availability and need for interactivity
- Support for high-level schemas on datasets, without losing performance
- Support for processing data streams in a grid environment
Research Projects

- FREERIDE (Framework for Rapid Implementation of Datamining Engines)
  - High-level specification of a parallel data mining algorithm
  - Flexibly exploit different forms of parallelism
- GATES (Grid-based AdapTive Execution on Streams)
  - OGSA based
  - Support for processing distributed streams in a grid environment
  - Self Adaptation to meet real-time constraints
- XML-based high-level abstractions of datasets
  - XQuery/XPath for application development
  - Use of compiler techniques for program transformation and efficiency
FREERIDE Overview

- Framework for Rapid Implementation of datamining engines
- Demonstrated for a variety of standard mining algorithm
- Targeted distributed memory parallelism, shared memory parallelism, and combination
- Can be used as basis for scalable grid-based data mining implementations
- Published in SDM 01, SDM 02, SDM 03, Sigmetrics 02, Europar 02, IPDPS 03, IEEE TKDE (to appear)
Key Observation from Mining Algorithms

- Popular algorithms have a common canonical loop
- Can be used as the basis for supporting a common middleware
- Parallelism of different forms and execution on disk-resident datasets

```c
While( ) {
    forall( data instances d) {
        I   =   process(d)
        R(I)  = R(I) op d
    }
    ......
}
```
Shared Memory Parallelization Techniques

- **Full Replication**: create a copy of the reduction object for each thread
- **Full Locking**: associate a lock with each element
- **Optimized Full Locking**: put the element and corresponding lock on the same cache block
- **Fixed Locking**: use a fixed number of locks
- **Cache Sensitive Locking**: one lock for all elements in a cache block
Trade-offs between Techniques

- **Memory requirements**: high memory requirements can cause memory thrashing
- **Contention**: if the number of reduction elements is small, contention for locks can be a significant factor
- **Coherence cache misses and false sharing**: more likely with a small number of reduction elements
Combining Shared Memory and Distributed Memory Parallelization

- Distributed memory parallelization by replication of reduction object
- Naturally combines with full replication on shared memory
- For locking with non-trivial memory layouts, two options
  - Communicate locks also
  - Copy reduction elements to a separate buffer
Apriori Association Mining

Relative Performance of Full Replication, Optimized Full Locking and Cache-Sensitive Locking

500MB dataset, N2000,L20, 4 threads
K-means Shared Memory Parallelization

![Bar Chart]

- **full repl**
- **opt full locks**
- **cache sens. Locks**

- **x-axis**: 1 thread, 4 threads, 16 threads
- **y-axis**: 0 to 1600
Performance on Cluster of SMPs

Apriori Association Mining
Results from EM Clustering Algorithm

- EM is a popular data mining algorithm
- Can we parallelize it using the same support that worked for other clustering algo (k-means) and algo for other mining tasks
Results from FP-tree

FPtree:
800 MB dataset
20 frequent itemsets
A Case Study: Decision Tree Construction

• Question: can we parallelize decision tree construction using the same framework?
• Most existing parallel algorithms have a fairly different structure (sorting, writing back ...)
• Being able to support decision tree construction will significantly add to the usefulness of the framework
• Focused on Gehrke’s RainForest framework
Shared Memory Parallelization Strategies

- **Pure approach:** only apply one of full replication, optimized full locking and cache-sensitive locking
- **Vertical approach:** use replication at top levels, locking at lower
- **Horizontal:** use replication for attributes with a small number of distinct values, locking otherwise
- **Mixed approach:** combine the above two
Results

Combining full replication and cache-sensitive locking
SPIES On (a) FREERIDE

- Developed a new communication efficient decision tree construction algorithm – Statistical Pruning of Intervals for Enhanced Scalability (SPIES)
- Combines RainForest with statistical pruning of intervals of numerical attributes to reduce memory requirements and communication volume
- Does not require sorting of data, or partitioning and writing-back of records
Applying FREERIDE for Scientific Data Mining

- Joint work with Machiraju and Parthasarathy
- Focusing on feature extraction, tracking, and mining approach developed by Machiraju et al.
- A feature is a region of interest in a dataset
- A suite of algorithms for extracting and tracking features
FREERIDE Summary

- Demonstrated a common framework for parallelization of a wide range of mining algos
  - Association mining – apriori and fp-tree
  - Clustering – k-means and EM
  - Decision tree construction
  - Nearest neighbor search
- Both shared memory and distributed memory parallelism
- A number of advantages
  - Ease parallelization
  - Support higher-level interfaces
Outline

- FREERIDE (Framework for Rapid Implementation of Datamining Engines)
  - High-level specification of a parallel data mining algorithm
  - Flexibly exploit different forms of parallelism
- GATES (Grid-based Adaptive Execution on Streams)
  - OGSA based
  - Support for processing distributed streams in a grid environment
  - Self Adaptation to meet real-time constraints
- XML-based high-level abstractions of datasets
  - XQuery/XPath for application development
  - Use of compiler techniques for program transformation and efficiency
Grid-based AdapTive Execution on Streams

Targets (distributed) processing of (distributed) data streams

Built on OGSA model

Self adaptation to meet real-time constraint on processing
GATES: Motivation

- Processing of streams widely studied in data mining algorithms / database systems
  - Focus on centralized processing of centralized streams
  - Most work to date on algorithms (particularly in data mining)
- Many applications involve high-volume data streams
  - Data from large scale experiments / simulations
  - Digitized images from a movie camera
  - Network traffic
- Data may arise from distributed sources
- Analysis / consumption of results may be distributed
  - Many users wanting different analyses/results
  - Insufficient compute power at one site
- Improving wide-area bandwidth / QoS can allow grid-based real-time processing of data streams
GATES Requirements

For application developers
- Relieve from complexities of using grid resources
- Automatic resource discovery and resource/requirement matching
- Simple interface for enabling self-adaptation to meet real-time constraints

For application deployer
- Simple deployment – deploy only at the application container and distribution of processing is handled automatically

For application user
- Dynamic adaptation to meet real-time constraints
  - Adaptation to resource requirements and resource availability
GATES Processing Structure

- Processing is in a set of stages
  - First stage is at or close to data source, last stage is close to where results are desired
- Each stage can have up to three threads
  - Input Stream Thread: creates and listens to a socket, connect to stream users
  - StreamService Provider: Extracts and executes the processing associated with this stage
  - Output Stream Thread: Creates and monitors a socket, send write possible event to stream users
Observation: Online (one-pass) analysis algorithms are typically approximate

Goal: Achieve the best accuracy with available resources, subject to real-time constraint

GATES approach:

- Programmer exposes certain parameters in processing of each stage
  - Examples include: rate of sampling, size of summary structure
  - Programmer also specifies direction of sensitivity e.g. larger summary structure means more computation/communication
- Parameters adjusted at runtime
  - Currently based upon size of buffers: signal previous stage to become faster/slower if buffer too small / too large
- Future possibilities: use profiling / performance models …
Outline

- FREERIDE (Framework for Rapid Implementation of Datamining Engines)
  - High-level specification of a parallel data mining algorithm
  - Flexibly exploit different forms of parallelism
- GATES (Grid-based Adaptive Execution on Streams)
  - OGSA based
  - Support for processing distributed streams in a grid environment
  - Self Adaptation to meet real-time constraints
- XML-based high-level abstractions of datasets
  - XQuery/XPath for application development
  - Use of compiler techniques for program transformation and efficiency
Project Overview
Our goals

Support datasets of different formats
- HDF5
- Netcdf
- Chunked multi-dimensional datasets

 Ease of programming
- provide high level abstraction of datasets
- physical details are hidden from application developers
- Use XQuery/XPath for application development

 Compiler optimizations for performance
- physical details are exposed to compiler
- optimizations at both high level and low level

System Architecture

External Schema

XML Mapping Service

logical XML schema
physical XML schema

Compiler

C++/C

XQuery/XPath
Summary

- Developing data mining applications for a grid environment is hard
  - Need independence from architectures and data formats
  - Need high performance
  - System software tools are needed
    - Flexibly exploiting parallelism
    - High-level abstractions on datasets
    - Self-adaptation

- Data stream processing is going to be an important problem for grids
  - Distributed streams and/or distributed processing