A Practical Data-Driven Framework for Parallel Data Mining

Yang, Chunsheng; Létourneau, Sylvain

Proceedings of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI 2005), 2005

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

NRC · CNRC

*A Practical Data-Driven Framework for Parallel Data Mining **

Yang, C., and Letourneau, S.
July 2005

* published in The Proceedings of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI 2005), Orlando, Florida, USA. July 10-13, 2005. NRC 47440.

Canada

A Practical Data-driven Framework for Parallel Data Mining

Chunsheng Yang
National Research Council of Canada
Ottawa, Ontario, K1A 0R6, Canada
chunsheng.yang@nrc-cnrc.gc.ca

Sylvain Létourneau
National Research Council of Canada
Ottawa, Ontario, K1A 0R6, Canada
sylvain.letourneau@nrc-cnrc.gc.ca

Abstract

In many practical applications, data mining results must be quickly delivered. To achieve the required efficiency, without sacrificing the quality of the results, practitioners are now looking at ways to parallelize the most computationally expensive steps of the data mining process. Realizing that a complete rewriting of existing sequential programs into parallel ones is often too tedious and expensive, we propose a framework which re-uses existing sequential programs to perform parallel data mining on a computer cluster. The proposed framework relies on the JavaParty system and can be used to parallelize both Java and non-Java programs. This paper details the framework, illustrates the implementation, and presents early experimental results showing the benefits of the approach.

Keywords: Parallel Data Mining, Feature Extraction, Model Evaluation or Testing, JavaParty.

1. Introduction

Real-world applications of data mining often require quick delivery of high quality results. To meet this need, practitioners are now looking at parallel programming to speed up the computationally expensive tasks. The hardware resources to run parallel programs are increasingly available. For example, several organizations already have access to computer clusters or could build one at relatively low cost. Distributed computing environments based on systems such as Condor [1] [14] are also inexpensive and can be used to simulate the hardware infrastructure. Unfortunately, practical software solutions for parallel data mining are still not widely available. Although significant amounts of research are performed in the area of parallel data mining (e.g., [1][10][11][12]), the proposed solutions are often difficult to adopt; either because they do not rely on realistic hardware infrastructures or do not integrate well with the programs currently used in the organizations. This short paper addresses this problem by introducing a software framework that helps practitioners parallelize computationally expensive data mining tasks while maximizing re-use of existing programs. The framework proposed has been developed and evaluated on a Beowulf computer cluster and relies on JavaParty, an open-source support library for computer cluster programming. Although developed in Java, it can integrate non-Java programs and is also adequate for other distributed computing environments such as Condor.

There are two broad approaches to parallelize data mining tasks [9]: algorithm-oriented and data-oriented approaches. Research on algorithm-oriented approach targets the development of parallel algorithms for existing machine learning techniques such as parallel decision trees [10], parallel genetic algorithms [11], and parallel neural networks [12]. The algorithm-oriented approach typically involves a significant rewrite of existing programs. The data-oriented approach tries to reduce the complexity of a given data mining task by splitting the datasets into several subsets, analyzing each subset independently, and then combining the partial results to form the final one. The infrastructure proposed in this paper provides support to implement this approach on a computer cluster. To maximize benefits, the architecture facilitates reuse of an existing program for the second step (the analysis of the subsets). Previous work on parallel data mining has mostly focused on model building. As we will show through an example, the proposed infrastructure could also be used for other data mining tasks such as feature extraction and model evaluation.

[1] Condor is a specialized workload management system for providing the High-Performance Computing (HPC) environments. Such HPC environments provide a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management.

The next section introduces the framework. Section 3 details the implementation while Section 4 presents experimental results showing the feasibility of the approach. The last section draws conclusions and discusses future work.

2. A Data-driven Framework for Parallel Data Mining

As mentioned above, the proposed framework follows the data-driven paradigm to parallelize data mining on a Beowulf computer cluster. A Beowulf [15] computer cluster is simply defined as a group of computers (named *nodes*) that work together as a unified system. A parallel program in the proposed framework works as follow. First, it partitions the (potentially huge) initial dataset into multiple subsets. Then it creates a task for each subset and progressively dispatches the tasks to the various nodes until they are all completed. Finally, it combines the partial results to create the final one. The potential gain in performance over a sequential solution comes from the parallel execution of the multiple tasks. On the other hand, we note that this procedure includes operations that are not found in an equivalent sequential program: splitting of the initial dataset, creation and dispatch of tasks, and combination of the partial results. To achieve any gain in performance, we must ensure that the time taken by these extra operations does not exceed the amount of time saved through parallel execution of the tasks.

Figure 1 illustrates the proposed framework which has five modules: main program, data partition, task management, task execution, and results fusion. The main program coordinates the overall process for the given data mining task. It also simplifies communications by providing inputs to the various modules of the framework and by collecting results. Before presenting an example of a main program (Section 3), let us discuss the other modules in some detail.

2.1 Data Partition

The data partition module partitions a dataset ***S*** into *n* non-overlapping subsets noted $(D_1, D_2 ....., D_i, ..., D_n)$. It is the main program that specifies (through the input parameter *partition_algo)* how to partition the initial

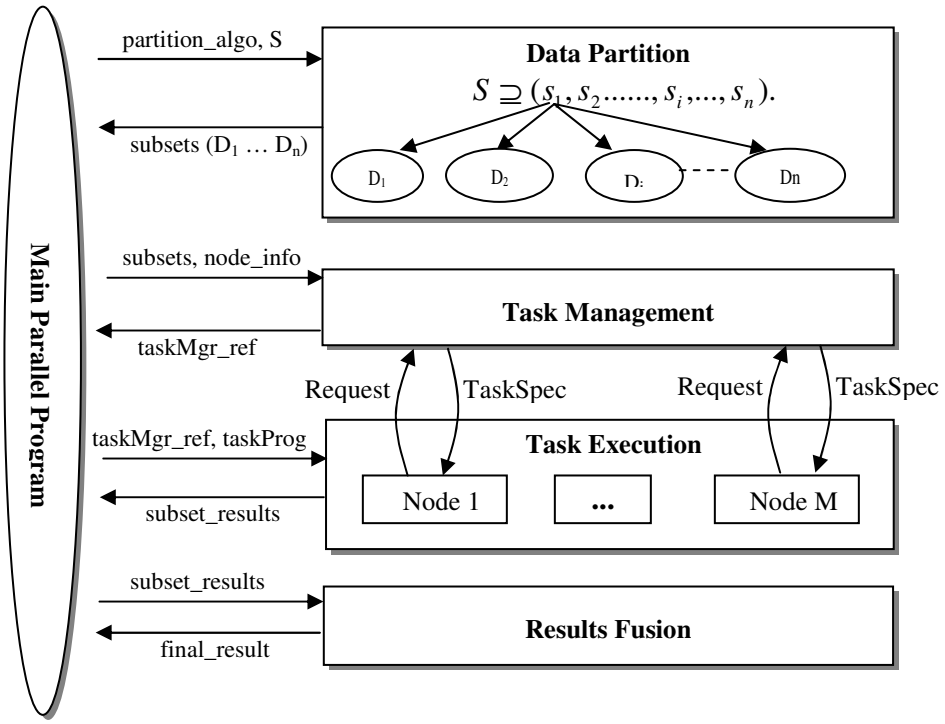

Figure 1. Overview of the proposed framework for parallel data mining.

dataset for that task at hand. Three commonly used partition methods are: random partitioning, sequential partitioning, and attribute-based partitioning. The random method constructs each subset through random sampling (without replacement) of the instances from the initial dataset. The sequential method simply selects the instances in the order that they appear in the initial dataset. These two methods create subsets of equal-size and by default they generate as many subsets as there are CPUs in the cluster. For example, if there are *M* computer nodes and each has *m* processors then these two methods will generate $n = M \bullet m$ subsets of equal-size. Equal-size partitioning is generally optimal as it distributes the work equally among the CPUs and simplifies the management of the tasks.

The attribute-based partitioning method splits the initial dataset based on the values of one or more attributes. This method is useful when each instance from the initial dataset belongs to a particular group and all the instances from the same group need to be analyzed together. For example, when performing feature extraction from a time-series dataset covering several entities of a given type (e.g., system, patient, stock), one would want all instances for a given entity to form a subset (e.g., a time-series). In such cases, attributes such as *patient_id* or system*_serial_number* would be used to perform the partitioning. In the case of attribute-based partitioning, the subsets may have different sizes and the number of subsets may not correspond to the number of CPUs in the cluster. There is therefore a potential for non-optimal use of the cluster; some of the nodes may be overloaded while others may be idle or have very little processing to perform.

2.2 Task Management

The task management module receives from the main program the list of subsets to be analyzed and the information on compute nodes that are going to participate in the computation. Given this information, it dispatches the tasks to the compute nodes. The process works as follows. When a CPU becomes idle (e.g., right after initialization or between two tasks), it sends a request to the task management module to receive a new task. The task management module responds by sending back a *TaskSpec* object which defines the task specific parameters, the task data, and a result container to store a partial result. The process stops when all subsets have been processed.

2.3 Task Execution

The actual execution of the tasks is done on local processes running on the various compute nodes. The main program launches these processes after initializing the task manager. As indicated in Fig 1, these processes take two input parameters at creation time. The first one (named *taskMgr_ref*) is a reference to the task manager which allows them to request new tasks when they are available. The second parameter (named *TaskProg*) specifies the program to run for the analysis along with any additional information required to execute this program. When the local processes receive a new *TaskSpec* object from the task manager, they execute the specified *TaskProg* on the subset defined in the *TaskSpec*. The *TaskProg* may directly refer to an existing application that is used for sequential data analysis, therefore maximizing re-use of existing code. For maximal benefits, the local processes need to know how to efficiently launch and control various types of applications. In our case, most of our sequential programs were written in Java. Accordingly, we decided to rely on a framework named JavaParty that allows tight interactions with Java programs. We have also interfaced our framework with the R[2] statistical system in a tightly coupled-manner to ease integration of statistical routines. Finally, the framework can also run any command line programs through an exec call to the underlying operating system.

2.4 Result Fusion

The result fusion module takes the partial results computed during the task execution step and generates the final result. Although not illustrated in Fig.1, this module can send partial results to the remote processes to allow re-use of previous results during computation of new ones if needed.

3. JavaParty-based Implementation

This section discusses the implementation of the proposed framework and presents an example of a main parallel program for a key data mining task: feature extraction. Traditional MPI (Message Parsing Interface) [4] and PVM (Parallel Virtual Machine) [5] approaches could be used to implement the framework. However, both MPI and PVM were designed for C/C++ programs and cannot easily work

[2] R is a statistical tool and requires the R-Java interface to interact between the Java application and the R system.

with existing Java programs. Recent works such as JPVM [8] try to address this problem by developing Java libraries for MPI and PVM. These libraries provide a Java interface allowing Java programs to interact with MPI or PVM. Java itself could be another solution to implement the framework, since it provides distributed programming support [7] such as Threads, RMI and CORBA. But these are low-level mechanisms with no support for cluster computing. After evaluating various open-source alternatives, we decided to use JavaParty [2][3] to implement the proposed framework. JavaParty provides a distributed Java virtual machine on top of a set of regular Java virtual machines cooperating on a common task in the computer cluster. JavaParty has two main features. Firstly, JavaParty allows the various programs to access remote information just like they access local ones; addressing issues, communications, and network exceptions are all taken care of by JavaParty internally. This feature, named *location transparent environment,* greatly reduces the level of details that the programmers need to take into account. Secondly, JavaParty supports object migration to help developers optimize communications. JavaParty implements the location transparent environment by adding *remote* objects to Java. The modifier "remote" is the only extension of the Java language. By declaring a class to be "remote" and instantiating it, JavaParty will automatically and transparently distribute the tasks to remote computer nodes. We use "remote" classes to define data subsets and the parallel processes.

Figure 2 illustrates the use of the proposed framework to parallelize feature extraction. The figure shows pseudo Java code to implement the main parallel program that performs parallel Fast Fourier Transforms (FFT) of a time-series dataset. The FFT for the subsets are computed using an existing R routine. The parallel program (named

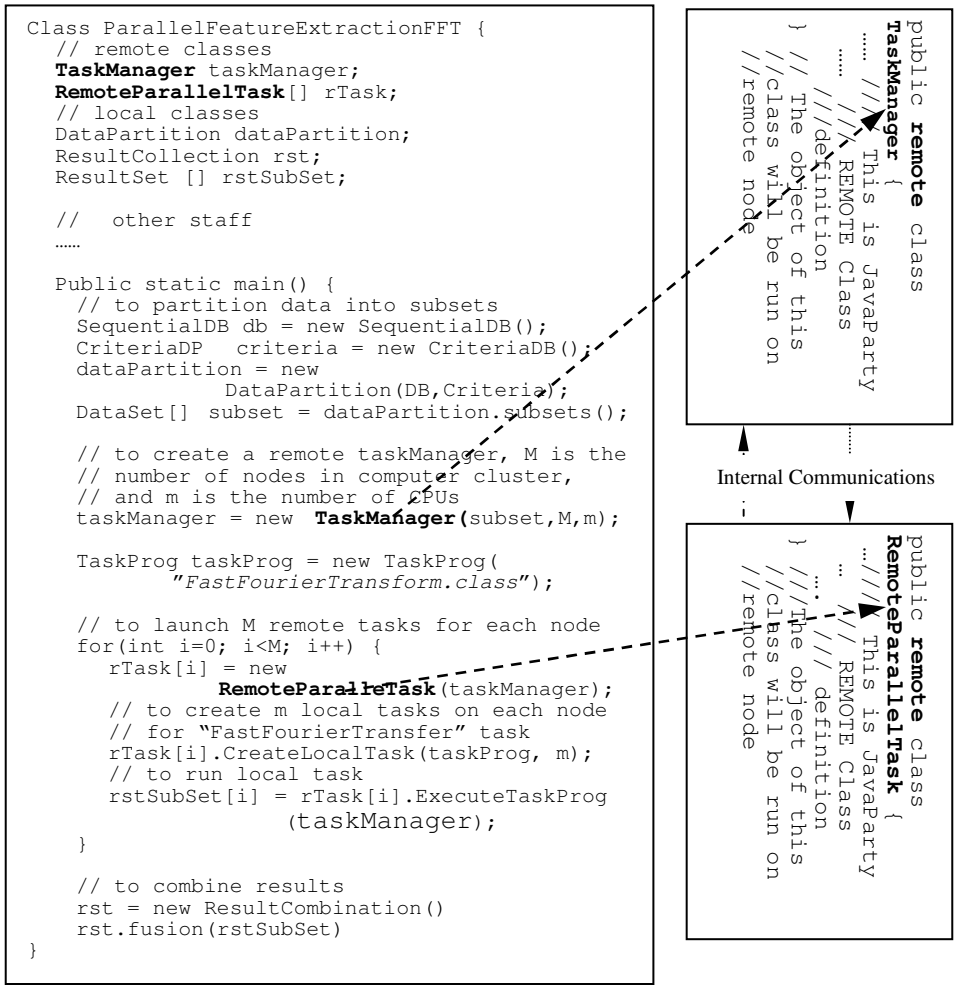

Figure 2. An example of a main program for parallelizing feature extraction.

ParallelFeatureExtractionFFT) first reads the full dataset and partitions it using the given criteria. Secondly, the program instantiates an instance of the remote class *TaskManager*. Thirdly, the program iteratively launches *M* remote tasks on the compute nodes by instantiating the remote class *RemoteParallelTask*. These parallel tasks will be executed on the compute nodes automatically and transparently. Meanwhile, the program lets each remote task fork *m* local tasks by executing the method, CreateLocalTask(*taskProg, m*), where *taskProg* specifies the program for the data mining task (in this example, it is the Fast Fourier Transform program named *FastFourierTransform.class*) and *m* is the number of CPUs on each remote node. All of these local tasks will execute the specified task program by calling the method, ExecuteTaskProg(*taskManager*), where *taskManager* is the reference to the remote task manager object. Through communication with *taskManager,* each local task will get a *taskSpec* object specifying the dataset to be processed. Finally, the program combines the results from all parallel tasks and generates the final result which contains the FFT for all observations in the initial dataset. We do not detail the remote classes *TaskManager* and *RemoteParallelTask* since they are as described in Section 2.2 (Task Management) and Section 2.3 (Task Execution), respectively.

4. Experimental Results

To illustrate the efficiency of the proposed framework, we report on experiments to parallelize the model evaluation task in a real-world application. The implementation is identical to the one described above except that we replaced the task program *FastFourierTransform.class* by one for model evaluation (named *ModelEvaluation.class*). We first constructed a high performance Beowulf computer cluster using the NPACI Rocks software [13] on the Linux operating system. Each computer node is configured with same hardware and software. This means that the nodes should have the same performance. The data comes from the WILDMiner project[3]. The training and testing datasets used contain 22083 and 2190980 instances, respectively. Using the training data and the WEKA system, we built four models: one NaiveBayes, one Decision Tree, and two Instance Based models (k=2,3). We then used our parallel framework to evaluate each of these models on the full testing dataset. For comparison, we performed the evaluation on a single computer, a 6 nodes computer cluster, and a 12 nodes cluster. Table 1 shows the experimental results.

[3] More information on the WILDMiner project is available at http://iit-iti.nrc-cnrc.gc.ca/projects-projets/wildminer_e.html

Table 1. The performance of parallel model evaluation (in seconds)

	Single Node	6 Nodes	12 Nodes
NaiveBayes	548	96	54
Decision Trees	453	80	44
IBk (k=2)	420844	64954	35051
IBk (k=3)	415760	64932	34795

As we increase the number of nodes, we observe an almost linear decrease in computation time. This near optimal performance clearly shows the potential of the approach and the appropriateness of JavaParty to realize the implementation.

5. Conclusions and Future Work

In this paper, we proposed a practical data-driven framework to parallelize data mining processes on a computer cluster. The framework capitalizes on readily available resources by allowing re-use of existing sequential programs. This characteristic should benefit organizations already involved in data mining that are in need of greater efficiency. We illustrated the implementation of the framework through a simple example and reported highly convincing experimental results. In short, the proposed framework provides a simple and effective way for programmers to implement parallel data mining processes that can significantly speed up the data mining process. We did not compare the performance of our implementation with other parallel programming techniques such as MPI, PVM and JPVM. This is part of our future work. Finally, we also want to explore specializations of the framework for particular data mining tasks. Hopefully, these will further increase applicability and performance.

Acknowledgements

Many thanks go to Bob Orchard for his comments on earlier drafts of this paper. Special thank is for Marc Leveille for his support on computer cluster and Silvain Bériault for his work on this project during a student coop

term at the National Research Council. We are also grateful to Bernhard Haumacher for the JavaParty system and for his technical support.

References

[1] Omer Rana, David Walker, Maozhen Li, Steven Lynden, and Mike Ward, "PaDDMAS: Parallel and Distributed Data Mining Application Suite", Proc. Of the Fourteen Inte'l Parallel and Distributed Processing Symposium, 2000, pp.387-392

[2] Michael Philippsen and Matthias Zenger, "JavaParty—Transparent Remote Objects in Java", Concurrency: Practice & Experience, Vol. 9, No. 11, 1997, pp.1225-1242

[3] Bernhard Haumacher and Michael Philippsen, "Exploiting Object Locality in JavaParty—A Distributed Computing Environment for Workstation Cluster", 2000, http://wwwipd.ira.uka.de/JavaParty

[4] Message Parsing Interface Forum: MPI: a Message-Parsing Interface Standard. University of Tenessee, Knoxville, TN. http://www.mcs.anl.gov/mpi.

[5] PVM: Parallel Virtual Machine, http://www.netlib.org/pvm3.

[6] Bryan Carpenter, Geoffrey Fox, Sung Hoon Ko, and Sang Lim, "Object Serialization for marshaling data in java interface to MPI", Concurrency: Practice and Experience, Vol. 12 No.7, 2000, pp.539-553

[7] Matthias Gimbel, et al, "Java as a Basis for Parallel Data Mining in Workstation Clusters", in the Proceedings of 7th International Conference on High Performance Computing and Networking HPCN Europe, April, 1999, Amsterdam, The Netherlands.

[8] Adam J. Ferrari, "JPVM: Network Parallel Computing in Java", ACM 1988 Workshop on Java for High-performance Networking Computing, February 1998, pp. 11-13

[9] Alex A. Freitas, "Mining Very Large Databases with Parallel Processing", pressed by Kluwer Academic Publishers, 1998

[10] R.A. Perrson, "A Coarse-grained Parallel Induction Heuristic: in " Parallel Processing for Artificial Intelligence" Edited by H. Kitano et al, Elsevier Science, 1994, Vol. 2, pp.207-226

[11] I. W. Flockhart and N.J. Radcliffe, " GA-MINER: Parallel Data Mining with Hierarchical Genetic Algorithm", EPCC_AIKMS-GA-MINER report1.0, University of Edinburgh, 1995

[12] S.K. Foo et al, "Parallel Implementation of Backpropagation Neural Networks", IEEE Trans. Systems, Man and Cybern – Part B: Cybern, Vol. 27(1), pp. 118-126, 1997

[13] P. M. Papadopoulos, et al, "NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters", Concurrency and Computation: Practice and Experience, 2002

[14] "Condor – High Throughput Computing", http://www.cs.wisc.edu/condor/description.html

[15] "A Beowulf Overview" --http://www.beowulf.org/overview/index.html