Mining in a Mobile Environment

Sean McRoskey
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, Indiana 46556
smcroske@nd.edu

James Notwell
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, Indiana 46556
jnotwell@nd.edu

Nitesh V. Chawla
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, Indiana 46556
nchawla@cse.nd.edu

Christian Poellabauer
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, Indiana 46556
cpoellab@cse.nd.edu

ABSTRACT

Distributed PRocessing in Mobile Environments (DPRiME) is a framework for processing large data sets across an ad-hoc network. Developed to address the shortcomings of Google's MapReduce outside of a fully-connected network, DPRiME separates nodes on the network into a master and workers; the master distributes sections of the data to available one-hop workers to process in parallel. Upon returning results to its master, a worker is assigned an unfinished task. Five data mining classifiers were implemented to process the data: decision trees, k-means, k-nearest neighbor, Naïve Bayes, and artificial neural networks. Ensembles were used so the classification tasks could be performed in parallel. This framework is well-suited for many tasks because it handles communications, node movement, node failure, packet loss, data partitioning, and result collection automatically. Therefore, DPRiME allows users with little knowledge of networking or distributed systems to harness the processing power of an entire network of single- and multi-hop nodes.

Keywords

Ad-hoc network, classifier, data mining, MapReduce, ensemble.

1. INTRODUCTION

Technology is exhibiting a trend toward wireless. For many, standards such as Bluetooth and Wi-Fi are replacing their wired counterparts, and powerful mobile devices are more popular than ever [1]. In other areas, there is a desperate need for real-time data analysis and extraction [2]. A government agency may need to quickly evaluate camera data from an airport to determine if it is indicative of terrorist activity. In another scenario, a financial institution may need to check a customer's credit card transaction for evidence of fraud. These data mining applications demand an immediate response. Processing large amounts of data on the mobile devices that are present in these areas, however, is taxing on their processors, memory, and batteries.

MapReduce is Google's solution to processing vast amounts of data on many commodity machines. It does so by dividing the machines into a master and many workers; the master splits the work into many small pieces and is responsible for coordinating the processing of each piece. MapReduce is an example of distributed processing; by employing the resources of the entire network, several less-powerful devices can outperform a single device with greater processing capability and resources. Unfortunately, MapReduce possesses several attributes that prevent it from being deployed in mobile environments. For this reason, we developed DPRiME, which utilizes this approach by partitioning the data into manageable pieces and distributing them among available wireless devices.

Much like Google's MapReduce framework, DPRiME represents an abstract framework and associated implementation for processing large data sets in a distributed environment. The nature of mobile environments precludes several important features of MapReduce, such as a fully-connected network and shared storage space, from being implemented on wireless devices. This served as the impetus for developing DPRiME as a framework for addressing the same problem with a mechanism suited for wireless ad-hoc networks. DPRiME retains the relevant features of MapReduce, with several necessary additions. Because DPRiME automates data partitioning, task management, and communications, it only requires the user to define the processing task, data, and processing parameters during run-time. Thus, no knowledge of the underlying workings of the system is required.

1.1 MapReduce

1.1.1 Framework

Google developed MapReduce to simplify distributed computing by separating data processing functions from the underlying distribution and parallelization tools. With MapRe-

SensorKDD'09 June 28, 2009, Paris, France

duce, users simply specify a Map and Reduce function, and the framework handles data partitioning, task assignment, fault tolerance, and other aspects of the distributed computing process. In this way, a user with no knowledge of distributed systems could still harness the capabilities of a large computing cluster [3].

The MapReduce process consists of two main parts: the Map task, which takes a set of input key/value pairs and produces an intermediate set of key/value pairs; and the Reduce task, which merges all of the values for each distinct intermediate key. An example of an input pair could be a word to count and a string in which to search for it; in this case, the intermediate pair would consist of this word and its count. The reduce task would then produce a count for all map tasks that searched for this word.

After the user specifies Map and Reduce functions, the framework designates one machine as the master and the rest as workers. MapReduce begins by partitioning the data into manageable pieces. The master, which coordinates the map and reduce tasks, assigns pieces of the data, as well as the Map task, to idle workers. The Map workers parse the data and pass them to the Map function, which produces intermediate key/value pairs that are periodically written to memory; pointers to the intermediate pairs are then passed back to the master. When the master receives a pointer, it relays it to a Reduce worker; the Reduce worker reads the remotely stored key/value pairs into memory, sorts it by intermediate key, and passes the intermediate values for corresponding keys to the reduce function, which produces an output for each intermediate key.

Several features within this process distinguish MapReduce as a robust and efficient model: fault tolerance, task granularity, and back-up. To address worker failure, the master periodically pings its workers. If it does not receive a response, it labels the worker as failed, cancels any task the worker is processing, and resets the worker to an idle state; the task the worker was processing is then reassigned. Task granularity encompasses the idea of splitting the data into many fine pieces. Doing so improves dynamic load balancing and also reduces the costs of worker failure. Back-up is implemented when MapReduce nears completion. It consists of the master assigning all currently running tasks to idle workers; a task is completed when either the primary or back-up worker finish. Back-up helps to mitigate the effects of stragglers, machines that take an especially long time to complete one of the final Map or Reduce tasks [3].

1.1.2 Limitations

MapReduce was designed and optimized to operate on large clusters of commodity machines that are fully connected over Ethernet. It reduces bandwidth usage by relaying data locations from Map workers to Reduce workers through the master; the Reduce workers then read the remotely stored data into memory. In doing so, MapReduce relies on shared memory and fully-connected machines. These features are not possible in mobile environments, in which data can only be shared by physically passing data over the network. In addition, every node in the network is not connected to every other node, requiring data to be transmitted to several nodes before reaching its intended destination. For these reasons, we found it unproductive to implement MapReduce for wireless environments.

2. DPRIME

To address the shortcomings of MapReduce in mobile environments, we found it necessary to develop a distributed framework that retained MapReduce‘s key features while incorporating communication and storage optimized for a mobile environment. DPRiME uses a similar approach to MapReduce—a master assigns portions of a data set to available workers, which process them using an assigned function. Several important features, however, distinguish DPRiME and make it ideal for deployment in mobile environments.

For this paper, DPRiME was used to distribute a classification task across a network, but it could easily be used to distribute many other types of jobs. It has the benefit of being able to process data stored on a single node or distributed on nodes throughout the network.

2.1 Master Functionality & Worker Discovery

A mobile device is designated as the master when the user specifies an input data set and processing parameters. Unlike the fully-connected computers that compose a cluster, the ad-hoc network used in a mobile environment prevents the master from knowing how many workers, if any, are available at a specific time. To initiate the process, the master broadcasts a PING message and listens for responses from workers. After an interval where the master waits for responses, it partitions the data according to the number of available workers and assigns a section to each for processing.

Much like in MapReduce, the master maintains data structures which track workers and assigned tasks. Each worker is tagged with a WorkerTag instance that contains information about the worker id, worker address, worker state, task assigned, and elapsed time from the workerÕs last communication . These allow the master to track workers and tasks throughout the DPRiME process, noting failed workers and unfinished tasks for fault tolerance and backup.

2.2 Worker Functionality

Worker nodes, if available, respond to a master PING with an AVAI response. The master then issues the worker a command packet (CMND) containing data and processing parameters. The master controls data processing by passing a task identification number, from a group of pre-programmed tasks, along with any required parameters for that task. In addition, the master can specify whether the worker should process locally stored data by passing the name of the file where the data would be stored.

2.3 Execution Overview

When the user specifies a data set, processing task, and processing parameters, DPRiME proceeds through the following sequence:

1. The master, begins by attempting to open the file containing the data and reading them into memory. If the data are smaller than a threshold value, the master processes them itself.

2. Otherwise, the master pings its one-hop neighbors, waits for responses, partitions the data according to the number of responses, and sends a piece of the data to each available one- hop neighbor.

3. If no neighbors respond to the ping, the master processes the data itself.

4. If there are no data to read into memory, the master sends the name of the file that would contain data to each one-hop neighbor.

5. If there are no data to read into memory and no one-hop neighbors, the task terminates.

When a worker receives a task from its master, it proceeds through the following sequence:

1. If any data were received, the worker begins by examining them. If the data are smaller than a threshold value, it processes them itself.

2. Otherwise, it distributes the task like its master. If there are no one-hop neighbors, the worker processes the data itself.

3. After completing the processing task, the worker processes local data in the same manner as its master.

4. Finally, it returns the results to its master.

Like MapReduce, DPRiME incorporates a back-up functionality; it maintains a list of tasks, through which it iterates. The master continually loops through the list and assigns any unfinished task to idle workers until every task is completed; a task is completed when the first worker replies with its result.

3. IMPLEMENTATION

3.1 DPRiME Protocol

Communication between devices was implemented in User Datagram Protocol (UDP), with several key acknowledgement and retransmission features built in for reliability. The combination of UDP's unreliable packet delivery and the mobility of workers in an ad-hoc environment necessitated a robust system for efficiently handling node failures and dropped packets.

Throughout the process, various scenarios were addressed which could cause long delays or, in some cases, failure of the process. For instance, a master's failure to notice an available worker at the beginning of the DPRiME process is costly, since data partitioning depends on the number of known idle neighbors. The initial PING is thus sent out several times to ensure the first master-worker communication is established. Once the process has begun, the master periodically broadcasts a ping to check for any new available workers.

Because the command packets (CMND) could be very large in some cases, they are sliced at roughly 8KB intervals and sent as a series to the worker. The worker records each received piece and sends a "command complete" (CMPL) packet to the master upon receiving the entire CMND. If a short timeout elapses before the complete command is received, the worker notifies the master with a receipt packet (RECV) enumerating the missing pieces; the master then resends these fragments.

Just as workers can come into the master's range in the middle of the process, they can just as easily leave. Worker mobility was thus addressed in several ways. When the

Figure 1: Execution Overview.

master sends a command, it defines a maximum silence period within which it requires a response from the worker. Workers begin working on a task and send "progress reports" (PROG) according to the maximum silence defined by the master. If a master does not hear from a worker within the maximum silence period, it issues a "hello" packet (HELO) and waits one more timeout period before marking the worker as missing.

Broken connections are expensive in time and processing power if a worker is unable to send the results of a tedious task back to the master. Thus, Dynamic Source Routing was implemented to address this challenge.

3.2 Dynamic Source Routing (DSR)

Especially in an ad-hoc environment, workers may frequently move away from their master while working on a task. It is infeasible, however, to simply reassign lost tasks. Besides costing precious time, there is no guarantee that the next worker will finish the same task within range. Therefore, DSR was implemented as a simple means for workers to find a route back to their master, to send both progress reports and results.

After sending a progress report or a results packet, workers expect an acknowledgement from the master (RECV). After three unacknowledged packets, the worker assumes it has lost connection to the master and begins building a new route using DSR. After a new route has been established, all communication from the worker to the master and vice versa includes a "DSR-data" (DSRD) header and is sent along this route. If the worker eventually determines it has lost the master again, it will simply restart the route-building process. In this way, the framework relies on multi-hop routing to ensure delivery of results and to avoid reassigning completed tasks.

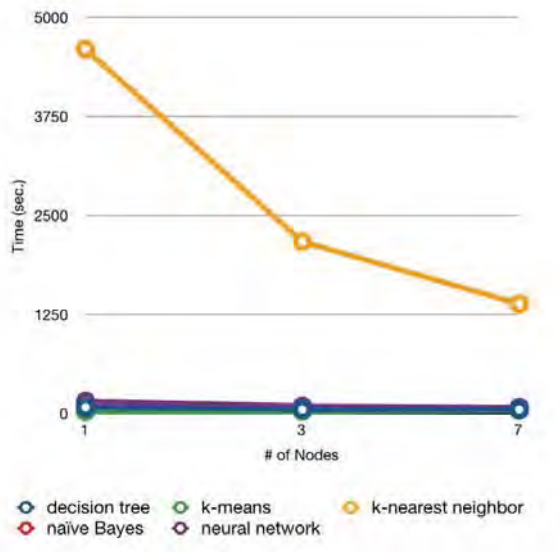
4. MINING

To simulate data mining in the field, we implemented five classifiers: Iterative Dichotomiser 3 (ID3) decision trees, k-means, k-nearest neighbor, Naïve Bayes, and artificial neural networks. On their own, these classifiers have been studied and are well-understood [4], [5]; implementing them in parallel, however, is the subject of ongoing research [6], [7], [8].

Initially, we considered developing a parallel implementation of each classifier. For decision trees, each branch would be parallelized as in [8]. Developing a parallelization for each classifier is non-trivial though, and would also limit DPRiME ' s capacity to accommodate new classifiers. For these reason, we chose to split the data, train a classifier on each split, and treat the classifiers as an ensemble.

Ensembles were selected for their ability to generate highly accurate classification results [9]. In our framework, after a classifier had been trained, it would classify each of the testing examples it had received; this took place on the worker nodes. Because only two-class data-sets were used (the framework is capable of processing data sets with any number of classes), the classification results were stored in a 2 x n array, where each row represented a class and each column (0, ..., n) represented a testing example. This array was then transmitted to the worker's master, where the arrays representing the classifications from different workers were added together. If this master was a sub-master, it would transmit the sum array to its master, which would do the same. When the results reached the master where the task originated, it would compare the number of votes for each of the two classes and assign each test example the class with the majority vote; ties were decided by randomly assigning a class.

Figure 2: Classifier completion times.



5. RESULTS

For our experiment, DPRiME was implemented in C# using the .NET Compact Framework. It was deployed on Motorola MC35 Windows Mobile devices. Figure 2 and Figure 3 display the time taken to perform classification tasks for three different scenarios. Both figures display the same information, but Figure 3 does not include the k-nearest neighbor classifier results. Experiments were conducted using three different configurations: one master, one master with two workers, and one master with two workers and two workers for each of the master's workers.

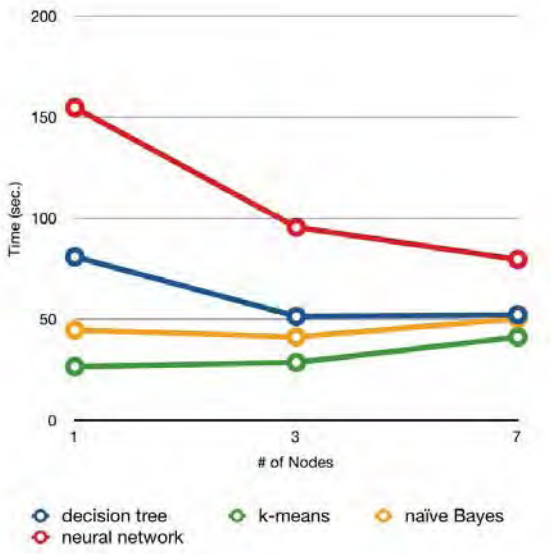
5.1 Discussion

Different parameters were used for the mining algorithms. The minimum leaf size for decision trees was two. K-means used five centroids for classification. K-nearest neighbor used three neighbors. Neural networks used 3 layers with 10 iterations and the learning rate was 0.1.

The same imbalanced data set was used for each classifier. It consisted of thirteen thousand examples with sixteen continuous features and two classes. One thousand testing examples were classified.

Not surprisingly, k-nearest neighbor exhibited the greatest improvement when distributed across this framework. This is because, unlike the other classifiers, no model was created for classification. Instead, classification required the entire data set split to be processed for each testing example making it the longest-running classifier so the communication overheads were only a small fraction of total time. K-means, which took the least amount of time, performed worse when distributed across our framework. This is because the additional communication costs outweighed the benefits of using multiple devices to process a task. We conclude that the

Figure 3: Classifier completion times (without k-nearest neighbor).



more computationally intensive a task, the more it benefits from using this framework.

5.2 Future Work

There is a great deal of work to be done in evaluating the effectiveness of this framework as well as determining parameters for the most effective deployment of DPRiME. Thresholds need to be established for the maximum file size that a worker will process and the size of splits made by a master. In addition, extensive testing will need to be performed to evaluate the time requirements versus accuracy of the different classifiers used in this framework. Finally, the performance gains made by using different combinations of single-hop and multi-hop neighbors need to be compared.

6. CONCLUSION

With the advent of ubiquitous wireless sensor networks and improvements in wireless technology, there will be an increasing demand for processing large amounts of data at their source. DPRiME simplifies this process by providing an abstraction that separates the more complicated distribution and communications tasks from the user-defined function that processes the data. By incorporating the important features of MapReduce into a robust framework capable of handling the shortcomings of MapReduce outside of a fully-connected network, we believe DPRiME will become a valuable tool for simplifying the data-intensive tasks in mobile environments.

7. ACKNOWLEDGMENTS

We would like to thank Notre Dame's Computer Science and Engineering department for the support, facilities, and equipment used in the development of DPRiME. This work was supported by National Science Foundation Grant CNS-0754933.

8. REFERENCES

[1] C. Pettey, "Gartner Says Worldwide Smartphone Sales Grew 29 Percent in First Quarter of 2008," Gartner, 6 June 2008, 30 July 2008 <http://www.gartner.com/it/page.jsp?id=688116>.

[2] B. Thuraisingham, L. Khan, C. Clifton, J. Maurer, and M. Ceruti, "Dependable Real-Time Data Mining," *Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing* (2005).

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large CLusters," *Sixth Symposium on Operating System Design and Implementation* (2004), 15 July 2008 <http://labs.google.com/papers/mapreduce.html>.

[4] T. M. Mitchell, *Machine Learning*, Ed. E. M. Munson, Singapore: The McGraw-Hill Companies, Inc., 1997.

[5] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. San Francisco: Morgan Kaufmann, 2005.

[6] K. W. Bowyer, L. O. Hall, T. Moore, and N. Chawla, "A Parallel Decision Tree Builder for Mining Very Large Visualization Data Sets," *IEEE International Conference on Systems, Man, and Cybernetics* 3 (2000): 1888-893.

[7] R. Agrawal and J. C. Shafer, "Parallel Mining of Association Rules." *Ieee Trans. On Knowledge And Data Engineering* 8 (1996): 962-69.

[8] G. J. Narlikar, "A Parallel, Multithreaded Decision Tree Builder," (1998).

[9] T. G. Dietterich, "Ensemble Methods in Machine Learning." *Lecture Notes in Computer Science* (2000).

[10] C. Chu, S. K. Kim, Y. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Oluktun, "Map-Reduce for Machine Learning on Multicore."