Issues in Applying Data Mining to Grid Job Failure Detection and Diagnosis

Lakshmikant Shrinivas and Jeffrey F. Naughton
Dept. of Computer Science
University of Wisconsin-Madison, USA
pachu@cs.wisc.edu, naughton@cs.wisc.edu

ABSTRACT

As grid computation systems become larger and more complex, manually diagnosing failures in jobs becomes impractical. Recently, machine-learning techniques have been proposed to detect a variety of application failures in grids. While this is a promising approach, there are many options as to how to apply machine learning to this problem, and it not always obvious which approaches are feasible or effective. We explore some issues that arise when we try to apply existing implementations of data mining algorithms to diagnose as well as predict job failures in grids. We demonstrate that a) it is feasible to gather enough data in real-time to train useful classifier algorithms, using only a small fraction of the grid's computational resources, b) it is important to choose the features used for classification with care, and c) it is useful to have both per-user and system-wide classifiers, as they diagnose different kinds of problems. We illustrate all these issues using a prototype system that runs over the Condor grid computation platform [3].

Categories and Subject Descriptors: D.4.7 [Operating Systems]: Organization and Design—Distributed systems; H.2.8 [Database Management]: Database Applications—Data mining

General Terms: Experimentation, Performance, Measurement.

Keywords: grid computation systems.

1. INTRODUCTION AND MOTIVATION

Recently, grid computation systems have become very popular as an aid to researchers and businesses interested in running large, complex applications. By allowing users the ability to spread out their computation over thousands of machines, grid computation systems allow for massive increases in productivity. However, this ability comes at a price — if any problems occur, it is very hard for users to diagnose why their jobs are failing, since the traditional way of determining problems (such as debuggers or profilers) cannot be applied at such a large scale. Recently, machine learning techniques have been applied to diagnose such large-scale job failure problems, as indicated by [2] and [4], and it seems to be a promising approach. However, machine learning can be applied in many ways, and it is not always obvious which approaches are computationally feasible or effective. We report our experiences with using specific machine learning techniques to diagnose failures in jobs being run on a cluster managed by the Condor system [3].

The authors of [2] demonstrated that data mining (specifically classification) algorithms are quite useful in identifying commonalities among job failures. The benefits of this technique are two-fold — a) allowing users to take corrective actions based on information gleaned from the mining, and b) predicting impending failures, in order to provide an "early warning system" for users and administrators. For example, if a user finds out that their jobs seem to fail on machines with limited memory, they can take corrective action by targeting machines that have sufficient memory. Likewise, if a user is told beforehand that a significant proportion of their jobs are predicted to fail, then they can stop and reexamine their submission before wasting time and system resources. In commercial grids with a pay-for-resources-consumed kind of economic model (such as Amazon EC2 [1]), such early warning might save the user money as well. Thus, data mining techniques may prove to be very useful for users and administrators of a grid computation system.

However, if a developer were to try to build a classification based fault detector/predictor for a grid, they would quickly find themselves faced with a number of questions, the answers to which are not immediately apparent. Some such questions are: What kind of information about the grid needs to be collected? Which classification algorithm(s) should be used? What are the trade-offs between the amount of data collected and the performance of the algorithm(s)? Is a single, central classifier enough, or are multiple classifiers needed? What issues arise when an existing classifier is adapted for use on a grid computation system?

In this project, we explore the answers to these questions. Our main contributions are as follows: we show that a) it is feasible to gather enough data in real-time to train useful classifier algorithms, with reasonable frequency, using only a small fraction of the grid's computational resources, b) it is important to train the classifier using only those attributes that can predict the outcome of a job submission, and omit attributes that are closely correlated with the success of the job, but are only available after job completion and c) it is useful to have both per-user and system-wide classifiers, as they diagnose different kinds of problems. Our goal is not to develop new machine learning algorithms, but rather to experiment with the application of existing algorithms to diagnosing job failures in grids.

2. SYSTEM ARCHITECTURE

In this section, we describe our system architecture, which is based on the Condor grid-computing platform [3].

Figure 1 shows the various components of Condor, consisting of various agents (known as *daemons*), a database and an instance of a job. When users want to submit a job to the system, they interact with the submission agent, which advertises the job requirements and supplies the program binary. The scheduling agent handles the allocation of jobs to machines. The machine agent manages the state of a machine in the grid. Once a job has been allocated a machine, the execution sandbox takes care of the execution of

HPDC'08, June 23–27, 2008, Boston, Massachusetts, USA.
ACM 978-1-59593-997-5/08/06.

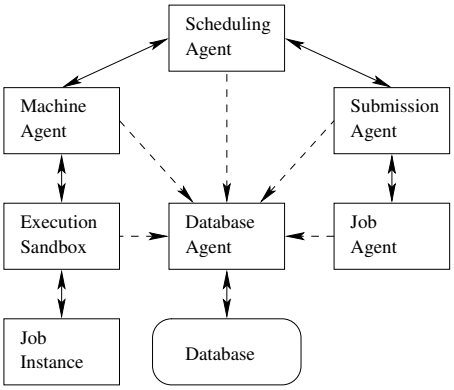

Figure 1: Overview of Condor

the particular instance of the job on that machine. For each job instance, a job agent is spawned on the machine from which the job was submitted. The job agent is responsible for supplying the necessary input files that the job needs, and receiving the output from the job instance. All these daemons communicate with the database agent, which records a variety of operational data from the daemons into a central database.

The execution of a particular instance of a job on a machine is known as a *run*. For each run, the central database contains all the job information, as well as the state of the machine that the run occurred on. In our current system, each run is treated as an instance (for purposes of classification) and is classified as either a Success or a Failure. A run is labeled a "Failure" if the exit code is not zero, or the program crashes due to a signal, or if it is kicked out and restarted on another machine, and "Success" otherwise.

3. RESULTS AND DISCUSSION

We now look at a couple of experiments we conducted, in order to shed more light on the questions raised Section 1. For our experimental setup, we used the version 6.9.3 of the Condor grid-computing platform [3] running over a cluster of 50 machines. All the machines had Intel x86 based processors and ran CentOS 4.5, a flavor of Linux. We submitted approximately 40,000 jobs, which resulted in approximately 140,000 runs. These jobs simulated various kinds of failures such as file transfer errors, program crashes on certain arguments and program crashes on insufficient memory.

We tested three implementations of classifiers — C4.5 [6], J48 [8] and vfdt [5]. C4.5 and J48 are both based on the C4.5 algorithm proposed by Quinlan [6], while vfdt is part of VFML toolkit [5], which is used for high speed data streams and very large data sets.

Figure 2 shows the training times of the three implementations as a function of the number of training instances. The timing experiments were performed on a computer with a 2.4 GHz Intel Core 2 Duo E6600 processor, with 2 GB of RAM, running CentOS 4.5.

We see that the training times are more or less linear in the number of instances for J48 and vfdt, and super-linear for C4.5. More important, though, is to note the absolute times taken to train the classifiers. We see that for 140,000 instances, even the slowest implementation (J48) took slightly less than 25 seconds to train, while C4.5 and vfdt took less than 15 seconds to train. Even the busiest grids managed by Condor today have less than 100,000 thousand jobs a day, and since the 140,000 runs in our experiments were generated from roughly 40,000 jobs, a single modern desktop computer can easily sustain the classification of jobs over a one or two day sliding window of jobs.

Another finding from our study was that it is important to omit certain attributes while training the classifiers, viz. attributes that

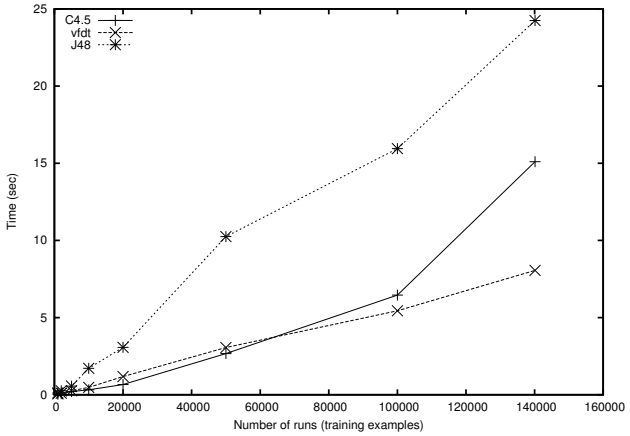

Figure 2: Training Times

are available only after job termination. Including such attributes makes the classifiers very accurate, but completely useless; for example, including the exit code attribute while training resulted in a classifier that essentially said "if the exit code is non-zero or null, the job is a Failure". Many of these attributes are closely correlated with the success or failure of a job, but cannot be used for prediction, since they are unavailable while the job is running.

The interested reader can refer to [7], which contains a more detailed analysis, as well as experiments exploring the trade-offs between system-wide and per-user classifiers.

4. CONCLUSION

In this project, we have attempted to shed some light on the issues that arise in applying machine learning algorithms and their implementations to the problem of grid job problem diagnosis. We demonstrated that it is feasible to gather enough data to train useful classification algorithms and that the classifiers can be trained with reasonable frequency using only a small fraction of the grid's computational resources. We also showed that it is important to train the classifiers using only those attributes that can predict the outcomes of jobs, and omit attributes that, while closely correlated with the success of jobs, are only available after job completion. Finally, we showed that it is useful to have both user-specific and system-wide classifiers, as they diagnose different kinds of problems and provide different insights to users and administrators. Even though data mining is not a solution to finding all problems in grids, overall it is a promising approach that deserves further investigation.

5. REFERENCES

[1] Amazon. Amazon Elastic Compute Cloud (Amazon EC2). Website. `http://www.amazon.com/b?ie=UTF8&node=201590011`.

[2] D. Cieslak, D. Thain, and N. Chawla. Short Paper: Troubleshooting Distributed Systems via Data Mining. *High Performance Distributed Computing, 2006 15th IEEE International Symposium on*, pages 309–312, June 19-23 2006.

[3] Condor. Condor Project. Website. `http://www.cs.wisc.edu/condor/`.

[4] J. Hofer and T. Fahringer. Grid Application Fault Diagnosis Using Wrapper Services and Machine Learning. In *International Conference on Service Oriented Computing*, Dresden, Germany, June 2007. Springer Verlag.

[5] G. Hulten and P. Domingos. VFML – a toolkit for mining high-speed time-changing data streams. 2003.

[6] R. J. Quinlan. *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning)*. Morgan Kaufmann, January 1993.

[7] L. Shrinivas and J. F. Naughton. Issues in Applying Data Mining to Grid Job Failure Detection and Diagnosis. Technical report, University of Wisconsin-Madison, 2008.

[8] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, October 1999.