HPC Production Job Quality Assessment

Extended Abstract

Omar Aaziz
New Mexico State University
Las Cruces, New Mexico
omarraad@nmsu.edu

Jonathan Cook
New Mexico State University
Las Cruces, New Mexico
joncook@nmsu.edu

ABSTRACT

Users of HPC systems would benefit from more feedback about the quality of their application runs, such as knowing whether or not the performance of a particular run was good, or whether the resources requested were enough, or too much. Such feedback requires more information to be kept regarding production application runs, and requires some analytics to assess any new runs. In this paper we assess the practicality of using job data, system data, and hardware performance counters in a near-zero overhead manner to assess job performance, in particular whether or not the job runtime was in line with expectations from historical application performance. We show over four proxy applications and two real application that our assessment is within 10% of actual performance.

KEYWORDS

runtime monitoring; high performance computing;

1 INTRODUCTION

Most HPC cluster users are not developers or advance users that are capable of operating complex tools to learn about their jobs performance. Moreover, users are interested in instant and meaningful feedback that can help them to tweak their jobs configuration to improve performance.

Without interfering with the application, we can collect information about its execution. We can get basic job configuration information from the submitted script; we can get system-level metrics from the nodes it is running on; we can use the nodes' hardware performance counters to get hardware-level application performance data; and we can observe application output as it is running. Our focus is on evaluating how effective using non-invasive instrumentation might be in providing useful information about application runtimes in production. In particular, the focus in this work is on estimating a specific job's runtime.

2 METHODOLOGY

The non-invasive metrics that we capture are the well known microprocessor hardware counters, available through the PAPI interface [6]. These can be collected at the system level without any application process instrumentation or interaction. We choose the following five counters: cycles, instructions, branch mispredicts, number of branches, and level one data cache misses. From these we calculate the following derived rates: instructions per cycle, branch mispredict rate per instruction, number of branches per instruction, and level one data cache miss rate.

For experimental purposes we divide the entire runtime to 8 equal intervals. We collected the raw hardware counters and then at the end of each interval output the values of the counters and the four metrics for that interval. Thus we have detailed time-series data for each of these metrics. Having this level of detail allows us to explore various analyses. We also record application name, number of processes, number of nodes, problem size, and measured time for each job.

We used the Lightweight Distributed Metric Service (LDMS) [1] component of the Ovis project as the instrumentation framework for our experiments. LDMS has an instrumentation ("sampler") plugin architecture, and for this work we extended a rudimentary PAPI sampler so that it automatically recognizes when an HPC job is started and attaches to the correct processes and collects the PAPI metrics, without needing to run the application using any of the command-line or other PAPI-compatible tools.

Our approach is to create a linear regression model that estimates runtime (dependent variable) from the metrics that we have, both PAPI time-series data and the job attributes described above. The runtime estimation performed by comparing the job actual runtime to the estimated runtime. Building up a history of an application will help users better understand each run's performance and will help them to better configure jobs in the future. We mostly focus on *rate-type* metrics derived from the PAPI counters, because these could be used to begin to estimate a job's runtime while it is still running. In the results Section we present results from various ways of using the PAPI counters.

Because of the volume and times-series nature of the PAPI data, linear regression is not applicable directly to this data. Therefore we apply principal component analysis (PCA) to the PAPI data to reduce the dimensionality down to a level where linear regression makes sense to apply. Note that the parameters that come out of PCA are also part of the visible model that we end up with, and can be used to help characterize the application. Not only are the PAPI metrics time-series, but they are captured *per process*. This results in large volume of data that needs reduced to a manageable number of values to perform PCA over.

3 RESULT

The results in this paper were taken from executing two real and four proxy applications on a 10-node cluster at our institution. Each node in this cluster has two intel 12-core CPUS (e5-2695) and 256GB RAM, and are connected with an Infiniband interconnect.

The six applications we used are MiniFE, SU2, Gadget, MiniGhost, Graph500, and Lulesh.

The most general summary of results is how well our regression models trained on the training runs were able to match the measured runtime of the test runs. Figure 1 and Figure 2 show the summary of the results achieved.

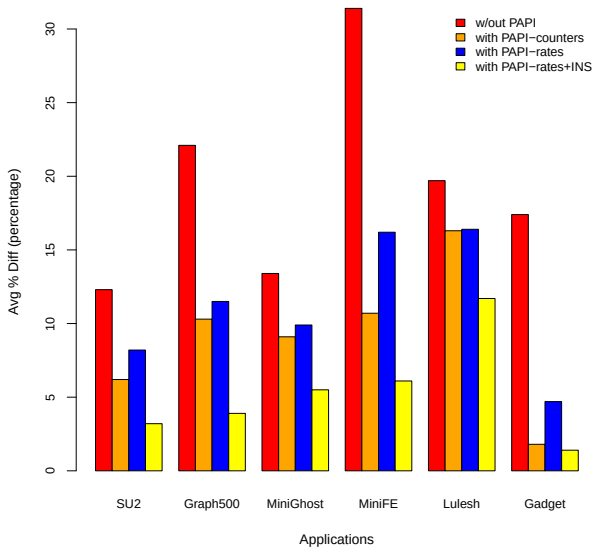

Figure 1: Difference in Estimated vs. Measured Runtime.

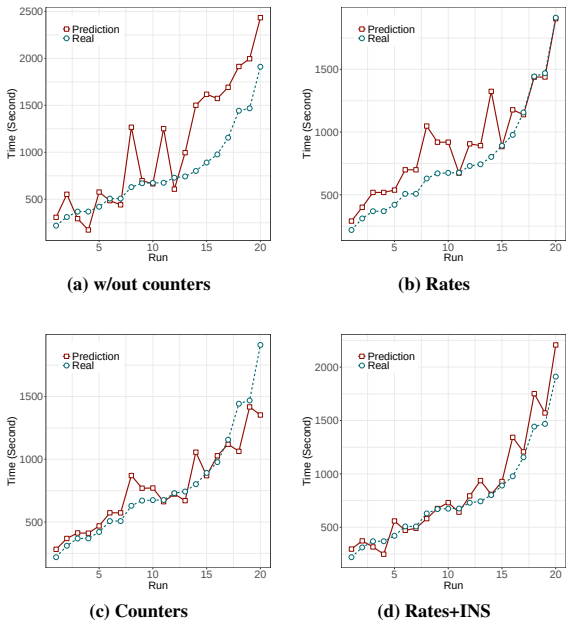

Figure 2: MiniFE: Difference in Estimated vs. Measured Runtime for All Estimation Modules.

The first observation is that for all applications except Lulesh, any form of hardware counters metrics significantly improves the estimation of runtime. Adding hardware counters metrics improves the estimations by about 50-75% for the other five applications. Most estimations are below 10% different from measured runtimes, and thus this lends support that we can get usable estimations in a framework that has near-zero overhead.

The second observation is that, while the hardware counters rate metrics are always worse than the direct counters, for four applications (SU2, Graph500, MiniGhost, and Lulesh), the hardware counters rate metrics are quite close in explanation power to the counters. This is a good result, supporting the possibility of doing estimations. With the focus on evaluating applications, this estimation calculation could be presented to users along with the confidence interval or with the variance shown for the application, so that a user would not have overconfidence in it when it was not justified.

The third observation is that adding the raw instruction counts to the rates metrics adds significant accuracy to the estimation models, though at the cost of introducing a counter metric dependency. It does make sense that this should do better than even all counters alone, since the other rate metrics essentially give an explanation of how to transform that raw instruction count into runtime. Because it uses raw instruction count, we view this model not as a useful estimation model but rather as a baseline of expectation for each application. For example, even this model averages more than 10% different than measured for the proxy application Lulesh, and so this gives a context for understanding the higher error rate for the other regression models.

4 RELATED WORK

Bhatele *et al.* [2], Calotoiu [3], and Jain *et al.* [5] collected MPI communication data and hardware counters then used regression to predict the execution time of MPI applications. Jain and Bhatele collected network hardware counters (e.g. the number of packets sent on each link) with analytic data (e.g. FIFO length) and applies regression models to predict the application execution time. Calotoiu predicts the scalability of MPI applications using Scalasca [4] by recording two sets of performance metrics, the application runtime with hardware counters and MPI communication information

These methods mainly designed for MPI applications analysis because of the MPI communication data collection high overhead. The overhead of these methods can be acceptable in the development environment but not for production. While our technique can be used in the production and development environment because of the low overhead.

5 CONCLUSION

We have presented a method of estimating the runtime of a particular run of an application through basic job parameters and PAPI metrics captured during the run itself. By using rate metrics derived from the raw PAPI counters, we leave open the possibility that we can adapt this methodology to be an online job statistic that can be queried as the job runs, just as other job statistics are. Our methodology used a standard statistical model rather than a potentially more powerful but opaque machine-learning method; this allowed us to explore what the model says about the application itself and certainly there is much more work to do in this area. We would like to also use these models to characterize application similarities, and whether proxy applications are similar to the applications they claim to represent. One additional future work will be to compare how a deep learning technique will perform compared to our statistical model.

REFERENCES

[1] Anthony Agelastos et al. The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications. In *SC'14: Proc. International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 154–165, Piscataway, NJ, USA, 2014. IEEE Press.

[2] Abhinav Bhatele, Andrew R Titus, Jayaraman J Thiagarajan, Nikhil Jain, Todd Gamblin, Peer-Timo Bremer, Martin Schulz, and Laxmikant V Kale. Identifying the Culprits Behind Network Congestion. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 113–122. IEEE, 2015.

[3] Alexandru Calotoiu, Torsten Hoefler, Marius Poke, and Felix Wolf. Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 45. ACM, 2013.

[4] Markus Geimer, Felix Wolf, Brian JN Wylie, Erika Ábrahám, Daniel Becker, and Bernd Mohr. The Scalasca Performance Toolset Architecture. *Concurrency and Computation: Practice and Experience*, 22(6):702–719, 2010.

[5] Nikhil Jain, Abhinav Bhatele, Michael P Robson, Todd Gamblin, and Laxmikant V Kale. Predicting Application Performance Using Supervised Learning on Communication Features. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 95. ACM, 2013.

[6] Philip J Mucci, Shirley Browne, Christine Deane, and George Ho. PAPI: A Portable Interface to Hardware Performance Counters. In *Proceedings of the department of defense HPCMP users group conference*, volume 710, 1999.