Technical Report No. 2012-595

Optimizing application execution on a computational grid by employing task scheduling policies (research proposal)[1]

Student Name: Rizwan Mian

Student Number: 6001533

School of Computing

Queen's University

Kingston, Ontario, Canada

September 5, 2012

Preface: After my Mphil [1], I started working on a research proposal for a PhD program. Majority of this work was done in 2007, with some minor updates in 2009 and 2010. With my current research direction and interests, I won't be able to pursue this actively. Nonetheless, I share the work-so-far in this technical report.

[1] Acknowledgements: I acknowledge input from Dr. Rehan Hafeez at NUST.

Optimizing application execution on a computational grid by employing task scheduling policies (research proposal)

Rizwan Mian
School of Computing, Queen's University
mian@cs.queensu.ca

Abstract

Simulations of complex biological and physical systems have been enabled by advances in computational technologies. This paper aims to study the task scheduling policies that make redeployment decisions for the components of the application to improve application execution performance. The policies are based on a cost model and performance predictions. Time-series and application intrinsic metrics based techniques are used for performance predictions to enable effective decision making.

Keywords: scheduling, prediction, migration.

1. Introduction

PerCo [2] is a performance control system that improves the performance of an application execution by reducing the overall execution time of the application at runtime. Such an application is constructed by composing previously independent or pre-existing components together. These components are seprately deployable computation units of well defined functionality. The application is deemed completed when all the components of the application have finished executing. Application Performance Steering (APS) part of PerCo manages the components on a specified (fixed) set of resources. APS monitors the relative rate of progress of the components on the allocated set of resources; and provides performance control by redeployment of components.

An example application may be to observe the interaction of an ocean simulation and an atmospheric simulation to determine climate. The computation requires interaction between these two simulations through a series of iterations or phases. Note that there are progress points between the phases as depicted in Figure 1. These progress points include APS activity. The APS activity, undertaken by task scheduling policies, is essentially a decision-making process which may lead to a new deployment for the components of the application.

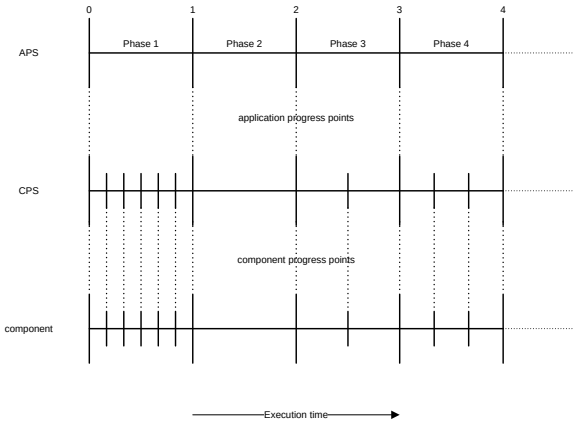

Figure 1: relationship between phases and progress points

There are roughly two classes of task scheduling policies (or just policies); based on heuristics and analytic modelling, respectively [2]:

1. The policies based on heuristics obtain a performance history of the application on certain resources, and then use some heuristic to improve performance. Note that in this method the performance prediction appears to be implicit in the decision procedures and heuristics.
2. In contrast, an analytical *performance model* is developed for modelling based policies. The performance model can be parameterized by application, history and system characteristics. Then the performance model is used to generate performance predictions using specific application, history and system parameter values.

A performance model based APS performance control can have a number of parameters including application, policy, *cost model* and *prediction*. Cost model and prediction are themselves based on other parameters. The goal is to minimize the application execution time. The redeployment decision is based on a cost model. A cost model allows to quantify

performance gains and losses produced by redeployment activities. The more accurate a cost model is, the better it can quantify performance gains and losses. A cost model can have various parameters which may be static (e.g. OS type) or dynamic (e.g. CPU availability). Dynamic parameters, as the name suggests, can vary during the execution of the application. Therefore, an accurate value of the dynamic parameter would give a more accurate cost model. We also need to know the value of dynamic parameter in the near-future. Prediction may be used to forecast the value of dynamic parameter. Figure 2 specifies the relationship between different building blocks discussed aforementioned and below. At every progress point, the APS policy decides whether to continue with the current deployment of components or to redeploy.

The rest of the paper is organized as follows. Section 2 discusses the building blocks of effective task scheduling policies, namely modelling the cost of migration and prediction, to address the problem at hand. Section 3 discusses existing approaches and systems in related work. A summary and discussion of the future work is provided in Section 4.

2. Building blocks

2.1 Migration

The APS component contains the logic for performance control. The APS policy may decide to redeploy or migrate the components in order to balance their rate of progress and keep overall application execution time minimal, possibly at the expense of increasing execution time of some components. This may involve migrating a slower component to a faster resource and vice versa. Recall, the goal is to minimise the overall execution time of the application.

The current APS policy is driven by detecting *symptoms* in current deployment of components. High *asymmetric* progress rates of components are the symptoms. If the symptom is detected, a migration is considered based on the cost model of migration. This symptom, however, leads to the problem that all the components may not complete execution at the same time. Since an application is not considered completed until all components are deemed completed, the slowest component becomes the bottleneck for the whole application.

Careful deployment of such components is required to achieve performance potential. At each progress point, the APS policy must be able to evaluate the performance characteristics of the application and its components.

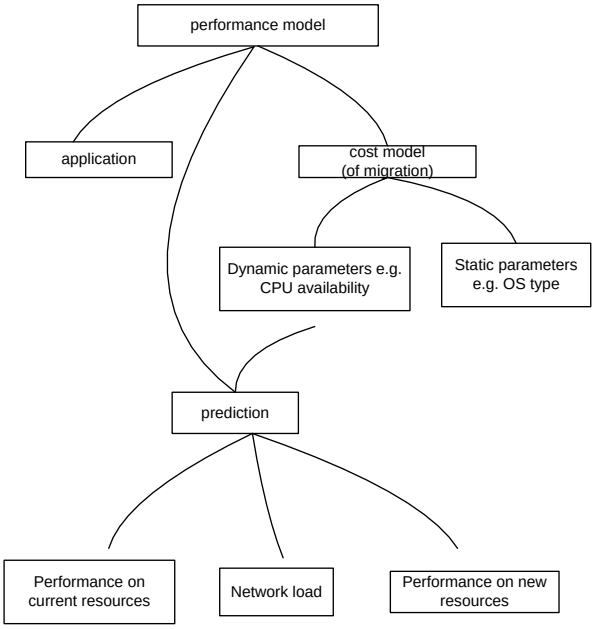

Figure 2: Building blocks

It must be able to quantify the performance gains and losses produced by migration activities. Typically, at every progress point, the APS policy must consider estimating:

1. the performance of the application if a migration is not performed;
2. the performance cost of a particular migration, using cost model; and
3. the performance of the application if the migration is carried out.

We can use a prediction model to estimate (1) and (3), i.e. by component prediction on current resource and component prediction on a new resource, respectively.

2.2 Modelling the cost of migration

A cost model is required to quantify the performance gains and losses produced by a migration. The cost model includes the time to:

a. determine a new deployment,
b. *checkpoint* the current state of the component,
c. transfer the checkpoint to the new resource, and
d. restart the component on the new resource from the check-pointed file.

We expect that (b) and (d) are negligible compared to other two factors, but would be experimentally verified. The time for (c) depends on the network load. Prediction can be used to estimate network load. We can determine (a) by traversing through the *search space*. The search space consists of performance

predictions of components on current and new resources.

The above cost model for migration is based on [3]. We need to take network load into account, since migration strategies that are purely based on CPU load-factors are inferior to strategies that also take network performance into account [4].

2.3 Prediction

First, we need the ability to predict how the components would perform on the current resources given some history. Second, we need the ability to predict how the components would perform on *different* resources. Third, we need the ability to predict the network load before initiating a migration.

2.3.1 Network load prediction: Wolski [5] describes a system, Network Weather Service (NWS), to predict network performance; and available CPU percentage for each resource that is being monitored. Prediction algorithms are based on time-series techniques. These algorithms are generic and can be applied for predicting other parameters. We will use them for predicting component performance on current resources.

2.3.2 Component prediction on current resources: We propose to reuse the algorithms in NWS [5] to predict the performance of a component on the current resource. We will record the history of a component on the resource. Then, the algorithms would be used to make a prediction using history.

2.2.3 Component prediction on new resources: Initially, we do not have history available for a component on a new resource. Therefore, we cannot use prediction based on history. We need a performance model that can be parameterized by a number of factors. An example of such a performance model is described in [6]. *Application intrinsic metrics* are solely dependent on the application code and problem parameters. These metrics express the demands of application on resources, and are independent of resources. Therefore, given that we can generate application intrinsic metrics and can obtain predicted CPU availability of a resource from NWS, we can predict how a component would perform on that resource.

3. Related Work

Paton *et al.* [7] proposed an abstract methodology for optimizing utility (could be execution time) in Cloud Computing through Autonomic Workload Execution. There is no discussion on the possible candidates for the building blocks. This makes the suggested methodology so abstract that it can be applied to grids and clusters – though the target domain is Cloud Computing. Nevertheless, the building blocks would be the key differentiating factors that would determine the effectiveness of a methodology to a particular paradigm.

Migration has been studied in [3]. This study successfully demonstrates the improvements in (*Jacobi*) application execution time when migration is used over one-time assignment of deployment. However for Jacobi, every processor needs to obtain information from its neighbouring processors at every iteration, therefore, the processors need to be synchronized after every iteration. Whereas PerCo assumes that inter-component communication is somewhat less and, therefore, does not require the components on different resources to synchronize at every iteration. Consequently, components can proceed with different rates of iteration cycles.

Job scheduling policies deals with the allocation of programs or jobs to processors according to the specified priority of the jobs and availability of resources [8]. These balance the workload among resources and optimizes system throughput – a system-oriented metric. It assumes no relationship between the jobs.

In contrast, task scheduling policies has a global view of an application. These prioritize and schedule components of an application to minimize application execution time – an application-oriented metric. These take into account dependencies between components, and offers co-scheduling when there is inter-component communication [8].

MARS [4] and *Dome* [9] both support check-pointing and migration by providing a custom runtime systems. The schedulers use migration to achieve *load-balancing* of unrelated and independent *jobs* [10], which is a system-oriented metric. In contrast, PerCo uses the underlying infrastructure i.e. Globus [11] for an initial deployment of related components, and then migration. The metric here is to minimise application execution time, which is an application-oriented metric.

Similarly, systems such as *Phish* [12] and *CARMI* [13] allow *workload* rebalancing at runtime. Such systems focus primarily on redistribution of jobs to utilize idle workstations, which is, again, a system-oriented metric. Resources dynamically join and leave the pool of available resources. PerCo assumes that a list of resources stays constant during the execution of the application.

GrADS [14] provides a redeployment mechanism, which is activated by a *performance contract* violation; which happens when the difference in the actual execution time and predicted execution time is beyond certain tolerance. The monitoring of execution times in GrADS is based on *Autopilot* toolkit. Autopilot assesses the application's progress using performance contract [6], which also uses application intrinsic metrics and resource capabilities to make a prediction. *Benchmark* executions are required to capture application intrinsic metrics prior to *real runs* of application. However, we intend to use: (a) application intrinsic metrics and resource capabilities to make a prediction, without the need for benchmarking new resources; and (b) time-series techniques studied in NWS to make a prediction on current resources.

Condor [15] is a hunter for idle resources. The system aims to maximize utilization of resources, which is a system-oriented metric, with minimal interference. When Condor detects interactive usage of a resource, it migrates the job from that resource to another resource. Condor assumes the job is autonomous and the focus is local. Notice, the trigger for redeployment is interactive usage of the resource not the different rate of progress of related components, as in the case of PerCo.

4. Conclusion and Future Work

The proposed research has many parameters that can be used for evaluation. Following parameters are of particular interest: application, policy, cost and prediction model, heuristics, performance and efficiency, and meta-data. In particular, we can

1. compare policies, cost models and prediction techniques in terms of application performance improvements they provide,
2. evaluate PerCo in terms of performance and efficiency. In this context, performance is a measurement of how well the resources are being used, whereas efficiency is a measurement of how much overhead does performance control exhibit,
3. compare against similar systems (e.g. Dome and MARS), and
4. do performance improvements with introducing policies in CPS

Scheduling has been studied for a long time in distributed computing [16]. As far as the author knows, time-series techniques have not been studied to predict performance of components on resources. Moreover, a technique for establishing application intrinsic metrics without benchmarking has not yet been established. Specifically, these techniques have not been used for migrating components to reduce the application execution time on a standard grid infrastructure i.e. Globus.

There is already a test bed for experimenting with policies, and that is the PerCo prototype. Various policies, cost models and prediction methods can be developed and evaluated using the PerCo prototype. Thereafter, the policies can also introduced in Component Performance Steerer (CPS), which improves the performance of an individual component. The APS can also be enhanced to perform dynamic selection, exhibiting *adaptive*-ness, from these policies based on, say, dynamic history as is done in [5]. Heuristics based on meta-data can be used to start the initial deployment with previously best performing policy. In the absence of PerCo prototype, various grid simulators may be considered such as GridSim [17] and SimGrid [18].

5. References

[1] R. Mian, "Managing distributed information for performance control of Grid-based applications," Department of Computer Science, University of Manchester, Manchester, 2005.

[2] K. Mayes, G.D. Riley, R.W. Ford, M. Lujan and T.L.Freeman, "The Design of a Performance Steering System for Component-Based Grid Applications.," *Performance Analysis and Grid Computing*, V. Getov, et al., eds., Kluwer Academic Publishers, 2003, pp. 111-127.

[3] G. Shao, R. Wolski and F. Berman, "Modeling the cost of redistribution in scheduling," *Proc. Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, 1997, pp. 9-17.

[4] J. Gehring and A. Reinefeld, "MARS -- a framework for minimizing the job execution time in a metacomputing environment " *Future Generation Computer Systems*, vol. 12, no. 1, 1996, pp. 87-99.

[5] R. Wolski, "Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service," *Proc. Proceedings of the Sixth IEEE International Symposium on High Performance Distributed Computing.*, 1997, pp. 316-326.

[6] F. Vraalsen, R.A. Aydt, C.L. Mendes and D.A. Reed, "Performance Contracts: Predicting and Monitoring Grid Application Behavior " *Proc. Proceedings of the Second International Workshop on Grid Computing* Springer-Verlag, 2001 pp. 154-165

[7] N.W. Paton, M.A.T. Aragão, K. Lee, A.A.A. Fernandes and R. Sakellariou, "Optimizing Utility in Cloud Computing through Autonomic Workload

Execution.," *IEEE Data Engineering Bulletin*, vol. 32, no. 1, 2009, pp. 51-58.

[8] J. Cao, A.T.S. Chan, Y. Sun, S.K. Das and M. Guo, "A taxonomy of application scheduling tools for high performance cluster computing," *Cluster Computing*, vol. 9, no. 3, 2006, pp. 355-371.

[9] J.N. Arabe, et al., *Dome: Parallel Programming in a Heterogeneous Multi-User Environment*, CMU-CS-95-137, Carnegie Mellon University, 1995. http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA295491

[10] F. Berman, et al., "Adaptive computing on the grid using AppLeS," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 4, 2003, pp. 369-382.

[11] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, no. 2, 1996, pp. 115-128.

[12] R.D. Blumofe and D.S. Park, "Scheduling large-scale parallel computations on networks of workstations," *Proc. Proceedings of the Third IEEE International Symposium on High Performance Distributed Computing.*, 1994, pp. 96-105

[13] J. Pruyne and M. Livny, "Parallel Processing on Dynamic Resources with CARMI " *Proc. Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, 1995 pp. 259-278.

[14] F. Berman, et al., "New Grid Scheduling and Rescheduling Methods in the GrADS Project," *International Journal of Parallel Programming*, vol. 33, no. 2, 2005, pp. 209-229; DOI 10.1007/s10766-005-3584-4.

[15] M.J. Litzkow, M. Livny and M.W. Mutka, "Condor-a hunter of idle workstations," *Proc. 8th International Conference on Distributed Computing Systems.*, 1988, pp. 104-111

[16] K. Krauter, R. Buyya and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing," *Software: Practice and Experience*, vol. 32, no. 2, 2002, pp. 135-164.

[17] R. Buyya and M. Murshed, "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, 2002, pp. 1175-1220.

[18] A. Sulistio, C.S. Yeo and R. Buyya, "A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools," *Software: Practice and Experience*, vol. 34, no. 7, 2004, pp. 653-673.