Automatic Tuning in Computational Grids[1]

Genaro Costa, Anna Morajko, Tomás Margalef, Emilio Luque

Computer Architecture and Operating Systems Department.
Universitat Autònoma de Barcelona
08193 Bellaterra, Spain
genaro.costa@aomail.uab.es;{anna.morajko, tomas.margalef, emilio.luque}@uab.es

Abstract. The possibility of have available massive computer resources to users as a market commodity opens the ideas for the future of the interoperability between the multiple infrastructure systems. This wide system should be composed of multiple high performance resource clusters and their users should share them to solve big scientific problems. These resources have a dynamic behavior and to reach the expected performance indexes it is necessary to tune the application in an automatic and dynamic way. The MATE environment was designed to tune parallel applications running on a cluster. This paper presents the key ideas for tracking down application process in a wide distributed environment like Computational Grids, and enables the use of MATE for dynamic application optimizations in such systems.

1 Introduction

With the advent of Internet there was an intensification of information share and services. Another fact is that the commodity computers have the computational power of the old mainframes. To solve new class of scientific problems, new systems like computer clusters became more common and accessible to everyone. The interconnection of clusters with different administrative domains builds a new class of system called Computational Grids [1]. New class of applications can execute in this wide environment spawning processes over more than a single cluster. The control of the application can be done by using a high level scheduler like Condor-G [2]. The composing processes could be submitted using the Globus Toolkit [1] to fulfill the requirements of security and resource localization and execute in a cluster scheduler environment like Condor [2] or OpenPBS [3].

One common use of the Grid processing nodes resource type is to decrease application execution time through parallelization. A user can describe the application resource needs characteristics and the Grid system layers drive the execution requests to locations that can fulfill the expectations. At machine layer, or fabric layer using the Grid terminology presented by Foster in [1], the application execution spawns parallel processes controlled by a batch queue system. The application can use a single cluster to avoid communication overheads or can spawn itself over more than

[1] This work has been supported by the MCyT (Spain) under contract TIN 2004-03388.

one cluster, possibly dealing with high latency message passing communication problems.

We developed a Monitoring, Analysis and Tuning Environment (MATE) that runs at clusters under PVM platform [5]. In our environment, an application execution can be guided based on performance models to obtain better execution time or less resource utilization. MATE uses the concept of tuning components called *tunlets* which encapsulate the monitoring, analysis and tuning logic for specific problems.

This paper discusses how to plug in our Monitoring, Analysis and Tuning Environment in a computational Grid as a fabric layer service or a user application plug-in. Section 2 describes our target Grid environment and characterizes user specific requirements for a dynamic tuning in such system. Section 3 presents three techniques for MATE integration in the Grid Fabric Layer. Finally, Section 4 presents the conclusions of this work.

2 Computational Grids

An oversimplification of Computational Grid could be a wide collection of interconnected resources distributed under different administrative domains. A common way of use the computational power of a Grid is to spawn the processes of a massive parallel application within the available processing node resource. Following the Grid Protocol Stack, the Grid Fabric Layer is where the resources are managed, acquired and controlled. In a typical environment, each resource would be computational host or cluster controlled by a local scheduler. To instrument an application in such environment, a monitoring tool should track down the application process components to gather instrumentation data. This starts to be a problem as resource utilization is indirectly selected through the Grid Protocol Stack.

To track down process in a Grid environment, two approaches can be used: (i) bind the application together with a tracking process in a single binary or (ii) have a tracking process running on the target execution machine waiting for the application process. In the first approach, the execution of the application process is controlled by the bound tracking process, which may be responsible for gathering instrumentation or doing the dynamic tuning. In the second approach, the tool can look for application start in a pooling method or can monitor the scheduler tracing the application startup. The first approach can be done by application developers and users, targeting better application execution time, and the second can be done by site administrators interested in efficiency, since installation requires administrative privileges.

An important problem that the application may deal in a Grid environment execution is that such system generally have a heterogeneous infrastructure and in most cases, a dynamic system configuration. That behavior could lead to performance drawbacks such load imbalance problems, high latencies caused by improper message fragmentation or inadequate buffer sizes. If the target platform is changed (number of processors, processor speed, network bandwidth, etc.) the required optimizations may be different. Grid brings new challenges in performance analysis and tuning making classical approaches less useful. For example, post-mortem analysis that presumes repeatability may not be used as the grid platform is highly dynamic and rarely

repeatable. In this sense,,a dynamic tuning tool seems to be relevant approach that could adapt applications to system changes or runtime configuration.

3 Changes in MATE

MATE consists of execution environment that permits dynamic tuning of application without need of code modification, compilation or linkage, based on Dyninst [5]. The two main components of MATE are the Application Controller (AC) and the Analyzer. The AC is the component which interacts with the application process, inserting instrumentation code and doing the dynamic tuning modifications [6]. The Analyzer consists of a container of tuning components called tunlets. Each tunlet can encapsulate the logic of what should be measured, how data can be interpreted by a performance model and what can be changed to achieve better execution time or better resource utilization. The Analyzer interacts with the tunlets, which request for application instrumentation. These requests are forwarded to the AC. The AC receives the requests, instruments the application and forward the trace back to the Analyzer and it is dispatched to the desired tunlets. The tunlet decides, based on its performance model, what should be changed to tune the application and requests to the Analyzer application changes. These requests are forwarded to the AC and the dynamic changes are made on the application.

In the current version of MATE, the application is started under the control of the Analyzer component. The AC bounds itself as PVM Tasker, so, when the Analyzer spawns the application the AC is notified to spawn each process. By this sequence MATE has total control of the application and does steering execution of its processes [6]. To work in a Grid environment, the first change is that, the Analyzer is started independently of the application and the AC is in charge to start the tuning session.

Using the first approach to track down application processes, as presented in Section 2, is to generate a composed binary using three applications: a glue code, the current application and the AC binary. That preparation step can be done by the developer or even by the application users before the execution. By doing that composition, at runtime, the first executed code it's the one that is loaded on memory. The key idea is to put the glue code as the first application in the composed binary. In startup phase, the glue code locates the application and AC code within the composed binary, does some checkup initializations and executes the AC code. When the AC executes, it starts the application child process using the Dyninst library. The information needed at runtime by the glue code is a fixed size information record append at the end of the composed binary in preparation phase. When the glue code runs, it uses the information record to un-pack the AC and application code. After that, the glue code verifies if the environment has Dyninst installed by checking its environment variables. It the environment permits dynamic instrumentation, the glue code execs the AC code; elsewhere, it execs the application without instrumentation. If the AC is started, it creates the child process. The glue code acts as a wrapper process to AC or the application based on target system execution properties.

Using the second approach of process tracking presented in section 2, the AC is installed as an operation system daemon by the site administrator. The key idea is to

enable the machine with dynamic tuning services that can be used by any registered application. With the AC daemon running on each machine of a cluster, this cluster is ready to do tuning sections by user request. The application to tuned should be registered to the Analyzer by the user in order to start the tuning session. By doing this, the Analyzer broadcasts the location request containing application identification with the name and binary checksum hash to all registered AC process. This is necessary due possible process name coincidences. The AC can operate in pooling mode or pull mode. In pooling mode, it monitors changes in proc file system to detect applications process startup. When an application process name is found, AC ensures that the binary belongs to the application, attaches to the found process and starts the steering execution. In pull mode, the AC instruments the batch scheduler with Dyninst and waits for the callback event generated by exec system call. This precises application startup detection and control. We currently support OpenPBS [3] as proof of concept. In each of presented models of execution, the target execution machine should support Dyninst, without that, the dynamic tuning cannot be done.

4 Conclusions

Performance tuning of Grid application becomes very complicated due to unique Grid characteristics such as time varying resource demands, heterogeneous resources, geographic distribution and network sharing. The dynamic behavior of Grid environment reinforces the need of dynamic tuning tools since the user has less control about the application target execution hosts.

The literature is not clear about the requirements that make a tool enabled for Computational Grids, although, if this tool could be used transparently within the Grid Protocol Stack, it is part of the system. We presented three alternatives that enable MATE to be used transparently in such system within the Fabric Layer of the Grid Protocol Stack.

5 References

1. Foster, I. Kesselman, C., eds, “The Grid 2: Blueprint for a New Computing Infrastructure”, Morgan Kauffman, San Francisco (2003).
2. Frey, J., Tannenbaum, T., Foster, I., Livny, M, Tuecke, S. "Condor-G: A Computation Management Agent for Multi-Institutional Grids", HPDC10, 2001.
3. OpenPBS Team, “A Batching Queuing System”, Software Project, Altair Grid Technologies, LLC, www.openpbs.org.
4. Morajko, A., Morajko, O., Margalef, T., Luque, E.. “MATE: Dynamic Performance Tuning Environment”. LNCS, 3149, pp. 98-107 (2004).
5. Buck, B., Hollingsworth, J.K. “An API for Runtime Code Patching”. University of Maryland, Computer Science Department, Journal of High Performance Computing Applications. 2000.
6. Morajko, A.. “Dynamic Tuning of Parallel/Distributed Applications”, Phd Thesis, TDX-1124104-171625, 2004.