Tuning Master/Worker Applications: 
A Practical Use Case with MATE

Andrea Martínez, Anna Sikora, Eduardo César and Joan Sorribes

Computer Architecture and Operating Systems Department, Universitat Autònoma de Barcelona, 08193 Bellaterra, Barcelona, Spain
{amartinez, ania}@caos.uab.es, {eduardo.cesar, joan.sorribes}@uab.cat

Abstract—Programming parallel/distributed applications is a difficult task that requires a high degree of knowledge and expertise, especially to achieve the potential performance offered by HPC. Analysis and tuning tools can be helpful for automatically improving applications performance. In particular, dynamic analysis and tuning tools are necessary for applications that vary their behaviour at execution time. MATE is a tool that, employing performance models, can automatically and dynamically tune parallel applications. This work presents how a theoretical performance model has been integrated into MATE for dynamically tuning the data distribution and the number of workers of Master/Worker applications. The results show the effectiveness of using performance models for dynamically tuning parallel applications, and the achieved reduction in time when the application modifies its behaviour during its execution.

Keywords: dynamic performance analysis; dynamic and automatic tuning; performance models; parallel/distributed computing

1. Introduction

Currently, software applications are used to solve complex problems in several areas of science and engineering. Many of these problems have very high computing requirements that can only be addressed through parallel/distributed processing. Therefore, performance is usually the most important issue related to parallel applications. In this work, we apply a methodology, based on performance models, for automatically and dynamically tuning the performance of parallel applications. In particular, we focus on the implementation and integration of a performance model for Master/Worker applications in MATE [1].

When a programmer develops a parallel application, he or she wishes to achieve a level of performance close to the expected theoretical performance. Unfortunately, this is not usually the case because the development of this type of applications is a difficult task. So, with the aim of increasing the performance of their applications, developers must undertake a performance improvement process. This process includes 3 successive phases: monitoring, analysis and tuning. First, during the monitoring phase, information about the application behaviour is captured. Then, by studying this information, the analysis phase looks for bottlenecks, deduces their causes, and tries to determine what the correct actions to eliminate the problems are. Finally, in the tuning phase these actions are applied to the application to solve the problems and improve its performance. As a result, developers must be familiar with the application, the software layers involved, and the behaviour of the system on which it is executed.

Various approaches and tools have been developed to support the performance improvement process [2] [3]. In particular, one of these approaches is the automatic and dynamic tuning of the application without stopping, recompiling, or rerunning it. This type of performance tuning approach is especially recommended for applications that behave differently depending on input data, or may even change their behaviour during each execution. In such cases, it is not worth carrying out a post-mortem performance analysis and tuning because conclusions based on one execution may be invalid for another. MATE (Monitoring Analysis and Tuning Environment) is a tool that implements this approach. It is able to automatically and dynamically tune a parallel/distributed application using the knowledge provided by a performance model.

The remainder of this work is organised as follows. Section 2 briefly describes MATE. In Section 3, we present an overview of the performance model developed for dynamically tuning Master/Worker applications. Section 4 explains the integration of the performance model into MATE. In Section 5 we present the results of the experiments conducted using MATE to improve the performance of a Master/Worker application. Section 6 presents the related work in automatic and dynamic tuning. And finally, Section 7 details the conclusions of this study.

2. MATE

MATE is a tool that performs monitoring, analysis, and tuning of MPI parallel applications. Its objective is to improve the performance of a parallel application at runtime, by adapting it to the variable conditions of the system.
First, at run-time MATE instruments the application to gather information about its behaviour. During the analysis phase MATE receives events, searches for bottlenecks and specifies solutions for solving the performance problems encountered. Finally, the application is dynamically modified by applying the given solutions. MATE uses dynamic instrumentation [4] to modify the application at run-time, so it does not need to be recompiled or restarted.

MATE is composed of the following modules which cooperate to control and improve the application’s performance [5]:

- The **Application Controller** (AC) is a daemon that controls the execution and the dynamic instrumentation of each individual MPI task.
- The **Analyzer** is a centralised process that carries out the application performance analysis, and decides on monitoring and tuning. It automatically detects existing performance problems on the fly and requests appropriate changes to improve the application’s performance.
- The **Dynamic Monitoring Library** (DMLib) is a shared library that is dynamically loaded by the AC in the application tasks to facilitate collecting data and delivering it to the Analyzer.

Performance models constitute the knowledge used by MATE to conduct the performance analysis process. Each performance model is encapsulated in MATE in a piece of software called a tunlet. Each tunlet implements the logic to overcome a particular performance problem by encapsulating information concerning to the measurement points to insert instrumentation in the target application to gather performance information, the performance functions, which are a set of expressions that model the application’s behaviour, and the tuning points, which are the points of the applications that can be changed by a tuning action to improve its performance.

### 3. Master/Worker Performance Model

The goal of performance analysis is to identify and solve the application performance problems. This process may be supported by a performance model that can be a combination of analytical expressions and heuristics. The parameters needed for evaluating the model correspond to the measurements gathered during the application execution. We have implemented and integrated into MATE a tunlet with the Master/Worker performance model described in [6]. It is designed for Master/Worker iterative applications, where all process repeatedly performs all operations. The condition of the iteration-based application behaviour implies the existence of a significant number of iterations and persistent performance problems between iterations.

This performance model includes two phases to solve Master/Worker application performance problems: a load balancing strategy, and an analytical model to evaluate and predict the appropriate number of workers for the application. In the following subsections we summarise both phases, and how to represent them in terms of the knowledge organisation required by MATE.

#### 3.1 Load Balancing

Load balancing techniques try to avoid that some processes complete their processing before others. Some of these techniques are based on distributing the tasks in portions of decreasing size called batches.

In particular, we have implemented the strategy called Dynamic Adjusting Factoring (DAF) [7]. This technique divides the task set into different sized batches using a partition factor \( x_i \) whose value is dynamically adapted to the current load conditions of the application through expressions (1) and (2) located at Table 1. This factor depends on the mean \( \mu_C \) and standard deviation \( \sigma_C \) of task processing \( C \), and the number of workers \( N \). Table 1 shows the Dynamic Adjusting Factoring strategy definition, represented according to the MATE knowledge requirements.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>- ( N ), number of workers</th>
<th>- ( C ), task processing time, ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance functions</td>
<td>Partition factor of the first batch of the iteration: ( x_0 = 1 + (\sigma_C \sqrt{N/2}) / \mu_C ) (1)</td>
<td>Partition factor of the remaining batches of the iteration: ( x_i = 2 + (\sigma_C \sqrt{N/2}) / \mu_C ) (2)</td>
</tr>
<tr>
<td>Tuning points/actions</td>
<td>Partition factor. Its value can be modified throughout the iteration.</td>
<td></td>
</tr>
</tbody>
</table>

### 3.2 Adapting the Number of Workers

For determining the appropriate number of workers of the application, we have used the performance index \( P_i \) proposed in [6]. This index relates the execution time to the efficient use of resources in order to maximise the performance without wasting resources. Following the requirements of knowledge representation in MATE, the definition of this tuning strategy is presented in Table 2.

The parameters \( m_0 \) and \( \lambda \) are statically configured taking into account the characteristics of the computing platform. \( \alpha \) is calculated as the sum of task sizes sent to workers while the total communication volume, \( V \), is the sum of task sizes sent/receive to/from workers. Finally the total computation time, \( T_C \), is obtained by adding the computation time of workers in an iteration.
4. Tunlet Implementation

To dynamically tune the performance of Master/Worker applications, we have developed a tunlet that integrates the tuning strategies presented in Section 3. A tunlet is a library that encapsulates the information about a performance problem, implementing a particular tuning technique. Its implementation must use the Dynamic Tuning API [1] provided by the MATE’s Analyzer module.

Earlier works featuring MATE show applying separate tuning techniques to load balancing [8] or to adapting the number of workers [9]. It is worth noting the complexity of the developed tunlet as it encapsulates two tuning phases, taking into account the interactions between them. In particular, the phase for adapting the number of worker considers that the application is balanced.

For the proper development of the tunlet, its definition should include the identification and interpretation of a set of elements related to the performance model and the type of the applications under study. From the point of view of the performance model, the following must be defined: measurement points, analytic performance functions and tuning points/actions. With respect to the application, in our work we have taken into consideration:

- The programming model followed by the applications.
- The variables or values that can be manipulated, with the aim of locating variables to tune.
- The functions whose execution must be detected to gather behavioural information.

In order to implement the tunlet based on the presented Master/Worker performance model, we have followed a tunlet design and development process [10] consisting of four steps which are explained in the following subsections.

4.1 Identify Application Actors

The designed tunlet needs information about the different types of application processes that cooperate to solve a concrete problem. This knowledge is required because each type of process should be instrumented depending on the role that it plays in the application. The application to be tuned follows a Master/Worker paradigm, so, two types of process can be identified: the master and N workers.

4.2 Identify Measurement Points

Performance model evaluation requires determination of which points in the application execution - measurement points - must be monitored in order to collect behavioural information about the application to calculate the parameters of the model’s analytical expressions, which are shown in Tables 1 and 2.

The measurement points are located in either the entry to or exit from a function. One value is extracted of each of these points. However some parameters require multiple values and multiple measurement points for being calculated.

4.3 Identify Events

Events are messages in which the values extracted at the measurement point are sent to MATE’s Analyzer module. These events are explicitly defined within the tunlet. Multiple values obtained at the same measurement point can be encapsulated in a single event and these values will be used by the Analyzer module for calculating the parameters for evaluating the performance model.

Table 3 presents the relationship between events and measurement points required by the analysis process. For each measurement point the table shows the actor, the function where it is situated, whether it is the entry to or exit from this function and the value which will be obtained.

4.4 Identify the Tuning Points and Actions

The last step consists of identifying the specific variables that will be modified by MATE at runtime. Consequently, a Master/Worker application must include a variable indicating the partition factor to be applied to the set of tasks for the load balancing strategy, and a variable indicating the current number of workers. Once MATE has taken all measurements to calculate the parameters of the analytical expressions, the performance model can be evaluated, and depending on the results of this evaluation, the adequate point to modify the associated variable should be determined.

For the load balancing strategy, the evaluation of the expressions is triggered when two separate events are received by the Analyzer: Start Iteration and End computing worker. At the beginning of the iteration, the tunlet has gathered all the information for calculating the mean $\mu_C$ and standard deviation $\sigma_C$ of the task processing time for the previous iteration. This allows the calculation of the partition factor values for the first and second batch of the current iteration. On the other hand, when a worker has ended computing, the tunlet can verify if the information about the processing time of each worker that has participated in the computation.
The experiments were performed using 2, 4, 8, 16 and 31 workers. Each worker, the master, and the Analyzer were executed on a dedicated node. We have conducted our experiments using four scenarios:

1) Xfire was executed for different numbers of workers without tuning.
2) Xfire was executed with MATE, but only tuning the load balancing following the DAF algorithm. The initial partition factor was 0.5, and during the execution this value was adjusted to the load balancing conditions.
3) Xfire was executed with MATE, but only tuning the number of workers. The application started with two workers, and during the execution this number was changed according to the model described in Section 3.
4) Xfire was executed with MATE applying the entire developed tunlet, i.e., Xfire was tuned using the load balancing strategy and adjusting the number of workers.

Table 4 summarises the results obtained. The comparison of the execution times obtained for scenario 1 and 2 shows that dynamic tuning of the partition factor improves the Xfire performance because MATE is able to detect the load imbalance and correct the factor to reduce the execution time.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>#Workers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>48.08</td>
</tr>
<tr>
<td>2</td>
<td>37.19</td>
</tr>
<tr>
<td>3</td>
<td>Starting with two workers 7.49</td>
</tr>
<tr>
<td>4</td>
<td>Starting with two workers 6.48</td>
</tr>
</tbody>
</table>
Regarding scenario 3, MATE starts with two workers and then, upon receiving data from each iteration it adjusts the number of workers being employed. As it can be seen in Figure 1, as time passes, computational load variations cause changes in the number of workers in the application in order to achieve an optimal performance, making efficient use of the available resources. These changes in the computational load are due to varying condition in the weather, wind, vegetation or topology as the fire line progresses, and consequently the calculation of the new fire front may have a greater or lesser complexity. It can be observed that the execution time of Xfire with MATE is close to the best execution time obtained by different fixed number of workers, however with a better user of resources.

Figure 2 shows the execution time of Xfire application considering different number of workers and in the last column the execution time of Xfire under MATE applying the entire developed tunlet. It can be seen that tuning both the partition factor and the number of workers gives a lower execution time than when applying just one of the tuning policies, while at the same time making efficient use of the available resources.

6. Related Work

MATE presents an approach that automatically and dynamically improves the performance of parallel applications. This approach is based on the use of dynamic instrumentation and performance models as the intelligence engine of the analysis process. Currently, there are other tools which perform dynamic tuning processes.

Autopilot [13] is a project for dynamic performance tuning in heterogeneous environments. It is based on the use of real-time techniques, which dynamically adapt the system to different demands and resource availability. Similar to MATE, Autopilot monitoring process is based on dynamic integration of sensors, which extract information about the application. The information analysis and decision procedures are performed using fuzzy logic. The application tuning is done by dynamically inserting actuator processes that adjust the application behaviour. This requires knowledge about the application.

Active Harmony [14] is a framework, which allows dynamic adaptation of an application to the network and available resources using automatic adjustment of algorithms, data distribution and load balancing. Its structure is based on a client-server model. The client is the harmonised application, which sends performance information to the server. The server performs the tuning based on this information. In this tool, the monitoring process gathers measures for various libraries with the same functionality. Then, it uses heuristic techniques to explore the application optimisation space and to adjust the tuning values. The tuning process is based on choosing the best implementation among the libraries.

PerCo [15] is a framework for performance monitoring in heterogeneous environments. It is able to manage the distributed execution of applications using migrations, for example, in response to changes in the runtime environment. PerCo monitors execution times and reacts according to a control strategy to adapt the performance when significant changes occur in the application behaviour. The performance analysis and tuning process is performed using historical data, and combining time series and data adjustment methods.

The main difference between MATE and presented tools is in the analysis phase. In MATE, the analysis is based on performance models, whereas Autopilot, Active Harmony and PerCo use fuzzy logic, heuristic techniques, and historical data and time series respectively.
7. Conclusions and Future Work

Achieving high performance for parallel applications is a complicated task that requires a high degree of experience, especially when dealing with applications with dynamic behaviour, or those running on heterogeneous systems. In these cases, the dynamic tuning performance is the most adequate approach. MATE is a tool that implements this approach for tuning applications.

In this work, the implementation of a theoretical performance model for Master/Worker applications and its integration into MATE has been presented. MATE has been extended to improve application performance by balancing the load and determining the appropriate number of workers. The performance model has been encapsulated in a MATE component called a tunlet. To correctly design and develop the tunlet, it has been necessary to identify and interpret the relation between the performance model, the type of tuned application, and the tuning tool. The developed tunlet can be used to tune other applications based, as in the case of Xfire, on iterations and a Master/Worker paradigm. It would only be necessary to adapt the application to the tuning process, adjusting the names of certain functions and tuning variables.

The experimental results present the applicability of the MATE dynamic tuning environment and performance models to reduce the execution time significantly adapting the application to changing conditions and using the resources efficiently.

Acknowledgment

This research has been supported by the MICINN-Spain under contract TIN2011-28689.

References