A Black-Box Approach to Performance Analysis of Grid Middleware

Per Alexius, B. Maryam Elahi, Fredrik Hedman, Phillip Mucci, Gilbert Netzer, and Zeeshan Ali Shah

Center for Parallel Computers (PDC), KTH, SE-100 44 STOCKHOLM, Sweden
{alexius,elahi,hedman,mucci,noname,zashah}@kth.se

Abstract. We propose a black-box approach to performance analysis of grid middleware and present the architecture of a non-invasive platform-independent evaluation tool that quantifies the effects of the overhead imposed by grid middleware on both the throughput of the system and on the turnaround times of grid applications. This approach is a step towards producing a middleware independent, comparable, reproducible and fair performance analysis of grid middlewares. The result of such performance analysis can be used by system administrators to tune the system configuration and by developers to find the bottlenecks and problems in their design and implementation of the system. It can also be used to devise more optimized usage patterns. As a proof of concept, we describe the implementation details of the evaluation tool for UNICORE 5 and demonstrate the result of initial experiments.

Keywords: UNICORE, Performance Analysis, Grid Middleware.

1 Introduction

Performance analysis and benchmarking are relatively young and emerging areas in Grid computing which itself is only about a decade old. Grids are comprised of a hierarchy of heterogeneous resources that are managed by several interacting software components [1]. This inherent complexity frustrates the direct application of some of well established techniques in parallel or distributed computing, among them performance analysis and benchmarking [2].

A few research groups have presented their interpretation of what the parameters of interest in grid performance assessment are and have also proposed methods and developed tools for evaluation of grid systems [2,3,4,5]. However, benchmarking and performance analysis of grid middlewares are a less explored area of research. The characteristic of grid applications is that they vary in resource requirements. The resource that is ultimately allocated must satisfy the user specified minimum requirements, but could be very different from one run to another. This makes platform-centric performance assessment of any particular application of limited use to grid developers, and grid users. Although resource benchmarking could be used as a tool for assisting better resource ranking and allocation [6], it can hardly be used to reflect on the performance of the grid middleware. This is for two reasons. The first is that platform-centric metrics places

L. Bougé et al. (Eds.): Euro-Par 2007 Workshops, LNCS 4854, pp. 62–71, 2008.

undue restrictions on the underlying execution platform to provide hardware and software components able to provide these specific measurement capabilities; also these components can not be guaranteed to be available. Secondly, using such metrics on a grid system is in direct contrast to the platform independent nature of the grid execution model. This model very purposely abstracts the underlying platform to facilitate a ubiquitous execution environment. Furthermore, performance of the grid middleware affects the throughput of the system with little or bounded dependency on the execution platform. A thorough evaluation of the overhead imposed by the middleware is important and could be leveraged throughout the design cycle; by the developers and grid site administrators as a performance analysis and diagnosis tool and by grid users and grid application developers to find the optimal submission patterns for a particular grid site.

In this paper we present a black-box approach to performance analysis of grid middleware and introduce a non-invasive platform independent design of an evaluation tool that measures the overhead of the middleware and quantifies the effects of this overhead both on the throughput of the system and on the turnaround times of grid applications. The tool measures the overhead imposed by the grid middleware on different job submission patterns. It measures only the time spent on "grid work" for different number of simultaneous job submissions to show the effect of middleware overhead on the throughput of the system and the turn-around time of jobs for different submission patterns. Beside the useful information this tool provides to grid developers, administrators and users, we discuss how this approach could be leveraged to produce comparative performance analysis among different grid middlewares. With this approach we aim at producing middleware independent, comparable, reproducible and fair performance analysis of grid middlewares. As a proof of concept, we present the implementation details of the evaluation tool for UNICORE 5 and present the result of some initial experiments on the OMII-Europe [7] JRA4 internal evaluation test bed.

A discussion on how performance analysis of grid middleware and revealing the overhead imposed by middleware components provide indispensable information for grid developers, administrators and grid users is presented in section 2. Section 3 illustrates the design of our middleware evaluation tool. Section 4 provides implementation details of the tool for UNICORE 5. Section 5 demonstrates the result of initial experiments with the tool on UNICORE 5. Conclusions and future directions are presented in section 6.

2 Measuring the Overhead Imposed by the Middleware

To analyze the performance of the grid middleware rather than measuring the period of time an application spends on a resource, we propose a method to measure the time spent on the "grid work" per job, namely everything that has to be done before and after the job performs its computation on the grid resource. We expose the fixed costs of the grid middleware components-the latency of the components-and show how those costs affect the performance of a single job and

ultimately the throughput of the entire system, i.e. the bandwidth. Although the overhead of grid middleware is not a constant-factor, it is bounded by the resource usage of the job (e.g. storage requirements); so useful patterns can be extracted from submitting different types of jobs with different resource usages.

Whether a large or a small job is submitted, the fixed costs contribute to the overhead of the grid system. From a user point of view, jobs that run for a long time amortize this cost versus the amount of computation that they perform. However, a frequent usage model of grids is to submit thousands of small jobs as part of a particular experiment (i.e. signal analysis, pattern recognition, Monte Carlo simulations). In order to facilitate better scheduling and resource utilization, users submit the jobs individually instead of all at once. However, this usage model is highly prone to the overheads induced by individual grid components and those components can contribute significantly to the run-time of a series of jobs representing a body of work that a user would like to perform.

Arguably, it is important to be able to record, diagnose and address this latency issue early in the design cycle. Grid developers can use this information for evaluating design choice trade-offs. Consequently, the analysis of the performance assessments that reflects the overhead imposed by the middleware could be beneficial to both developers to improve the design and implementation of the middleware components and to grid users to try to find the optimal usage model for the submission of their grid applications to a particular grid infrastructure. Given some estimates of the performance patterns of a standard installation of a middleware are published along with the middleware distributions, system administrators could also compare the result of performance measurements on their grid site and diagnose possible problems in their system configurations.

2.1 Comparison Across Different Middlewares

One of the things that contribute to the originality of our approach is the ambition to facilitate the production of performance data about grid components that is comparative among different middleware stacks. However, due to the differences among the middlewares there is no obvious way to produce this type of data with a high level of detail in a non-invasive manner. On the other hand, since such performance comparison has not been previously done, even somewhat crude measurements that can reveal differences in performance among the middlewares should be of interest to the Grid community.

A grid may from the point of view of this test be considered a black box, into which jobs are submitted and from which results are returned. Having this view, an initial thought is to put each of the middlewares in the place of the black box and record the submit time and the completion time, i.e. the time when the results are returned. Assuming that the middlewares are set up in a fair way (running on the same hardware, using the same number of worker nodes etc.) the results from such measurements provide a basis for comparison of different grid middlewares and also show which grid middleware handles a certain number of jobs in the shortest time-all in the scope of a particular set of use cases.

Obviously to provide a basis for sound and valid comparison, it is necessary to create reasonably fair conditions when performing comparisons between middlewares. However, fair conditions are very difficult, not to say impossible, to achieve since there are such large differences in how the middlewares are designed. We argue that adopting a best effort approach can still give us some comparable performance measurements across different grid middlewares.

3 Design of the Middleware Evaluation Tool

As an initial attempt to create an evaluation tool for grid middleware with the goals mentioned in the previous section, we propose a simple architecture for a non-invasive, middleware independent performance analysis tool that treats the middleware as a black box and gathers and visualizes performance data. This tool consists of scripts for submitting jobs to the middleware, recording the necessary timing information and for processing the output into graphs and charts. On the grid system, jobs are submitted, executed and the output is returned. Recording timestamps for when jobs are submitted, started, terminated and for when the output is returned, makes it possible to measure the overhead imposed on jobs by a middleware prior to and after execution (we call this pre-work and post-work). Information extracted from this measurement includes how much overhead the grid imposes on a submitted job and how this overhead changes when the number of submitted jobs varies.

We can assume that somewhere inside the grid, each job is executed and thus has an actual start and end time. Recording these times in addition to the submit and completion time, adds to the level of detail of the performance assessment. However, it is not obvious that these times are easily available, at least not considering the fact that the submit and completion times as well as the start and end times may be recorded on different computers on which clocks may differ significantly. In order to circumvent this problem, we use call-backs from the submitted jobs to the submitting computer. When jobs make call-backs at the time of starting and finishing, the times when these call-backs are received at the submitting computer are recorded. Obviously this approach introduces other possible sources of error, such as network latency, but initially it is assumed that the impact of these potential errors is negligible in a controlled environment. A schematic view of this execution case is illustrated in Fig 1. The performance measurements extracted via this approach gives a view of how different settings of a middleware performs and how different design choices affect the overall performance of the system and can also be further advanced to provide comparative performance data between different grid middlewares. It can be used as a starting point for where to instrument for more detailed measurements and could also help developers decide where to focus for improving performance.

For the measurements presented in this paper, the jobs that have been submitted are 'no-op' jobs, i.e. jobs with a negligible execution time. A natural continuation would be to extend this to include also other types of jobs, in order

to stress particular components of the system. Further development of this approach would be to increase the number of jobs that are submitted and measure how the turnaround time changes. This can provide a view of how the middleware handles large workloads and also give a hint of where potential bottlenecks are to be found. The level of detail in this approach may not be high enough to point out exactly which components are the main performance bottlenecks, but gives an indication of where to spend effort on further investigation. These tools are being developed and tested on four different grid middlewares within the OMII-Europe Joint Research Activity on benchmarking [7]. As a proof of concept, the result of some initial experiment on UNICORE 5 is presented in section 5.

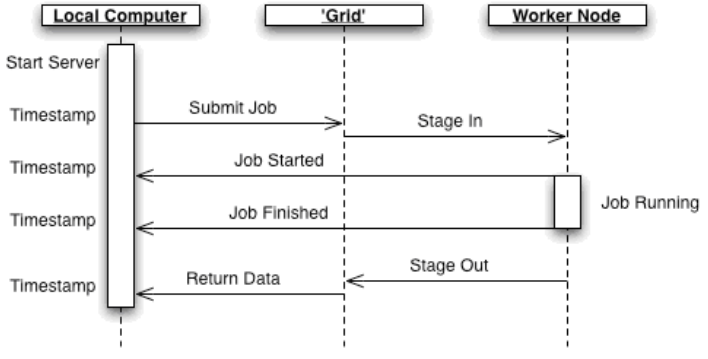

Fig. 1. Schematic view of the test script execution; jobs make call-backs when starting and finishing and the times when these call-backs are received at the submitting computer are recorded

3.1 Components of the Evaluation Tool

In this section, we describe the components of our evaluation tool:

Submit Script. A script to encapsulate the use of the scripting tools for the middleware and the recording of timestamps at relevant times. The submit script makes it possible for the user to submit a specified number of jobs to the middleware. The script produces the timestamps from the jobs as well as some metadata about the run. The output processor further processes this data to a more descriptive and graphic format.

Server. As described above, the submitted jobs issue call-backs to a server on the submitting computer when starting and stopping. The server listens for these call-backs and records timestamps when they are received. It is started by the submit script before submitting any jobs and when the last call-back has been received the server stops and hands over its output to the script.

Jobs. To perform initial measurement of the middleware overhead a very simple job without in-data or out-data is used. This job does not do anything else

but issue the two call-backs to the server before it exits. This job can be easily extended to stress test or for more fine-grained performance analysis of the components.

Output Processor. A script to process the timestamps and produce different graphs to show the turnaround time, throughput and the pre-work overhead versus post work overhead imposed by the middleware.

4 Proof of Concept on UNICORE

As a proof of concept, we provide the implementation details of the test tool for UNICORE 5 and present the result of running some initial experiments with the tool on the OMII-Europe JRA4 internal evaluation infrastructure.

4.1 UNICORE

The UNICORE (Uniform Interface to Computer Resources) grid middleware is a software infrastructure that aims at providing the user with a smooth, secure, and intuitive access to their heterogeneous and distributed computing resources. The key objective of UNICORE is to enhance abstraction, security, site autonomy, ease of use and ease of installation. UNICORE, which was initially developed by the UNICORE and UNICORE Plus projects, is now one of the major projects in Grid computing [8,9,10]. Recently UNICORE 6 has been released in a beta version; it is a web service based implementation. In the near future we will extend our evaluation tool to perform tests also on UNICORE 6. This will produce comparative measurements between the two versions of the middleware.

4.2 Evaluation Testbed Setup

The UNICORE server has three main components: the Gateway, the Network Job Supervisor (NJS) and the Target System Interface (TSI). These may be placed on different nodes or on the same node in any combination. We have chosen to place each component on its own node and also put the client interface, "Command Line Interface Queue" (CLIQ), on yet another node. SSL is used for the communication between the client and the gateway as well as between the gateway and the NJS. The NJS and the TSI communicate over plain sockets. The TSI interfaces to a default installation of Torque.

4.3 Middleware Specific Details for the Test Tool

When submitting a large number of jobs, the proposed way to interact with UNICORE 5 is through CLIQ. It is a tool that handles both job submission, monitoring and output retrieval. It lets you specify jobs or entire workflows in an XML format. When copied to the CLIQ submission folder, the information in the XML job specification file is used to create an Abstract Job Object (AJO),

which is then submitted to the UNICORE Gateway for execution. CLIQ keeps track of the status of the job and retrieves output files when the job is done.

The submit script interacts with CLIQ according to the following scheme:

1. Start CLIQ
2. For each job that should be submitted:
   (a) Create job description file.
   (b) Copy job description file to the CLIQ submit folder. This is where the submit timestamp is recorded.
3. Poll the folder where the output files from the jobs will be written. When the output files for a particular job is found, consider that job done. This is where the completion timestamp is recorded.
4. Stop CLIQ when all jobs are finished.

By default, CLIQ is restricted to not submit more than ten jobs at the same time and to poll for job status every fifth second. To stress the system slightly more we have set the maximum number of jobs to 1000 and to get more fine-grained results we have chosen a polling time of 100 ms. In next the section we present some results from initial tests as a proof of concept. We also categorize the results by interfacing UNICORE with different batch systems.

5 Experimental Results

The computer platform used consists of four nodes each with a 2.8 GHz Intel Pentium D CPU and 2 GB RAM connected through gigabit Ethernet. Scientific Linux is the operating system that has been used. The version chosen was 3.0.8. We used version 5 for UNICORE and for batch systems we used version 2.1.6 of Torque from its originator (we call it vanilla Torque) and the Torque version 2.1.6, which is packaged with gLite 3.0.2 (we call it gLite Torque).

5.1 Initial Test Scenarios

Below is the test scenarios for this paper:

- Measurements on Unicore-5 with direct submission (we call it Fork submission or unicore-nobatch),
- Measurements on Unicore-5 with submissions through Torque (gLite flavor Torque) and
- Measurements on Unicore-5 with submissions through Torque (vanilla).

For the scenarios above we produce the following graphs:

- Pre/Post work bar chart
- Turnaround time per job
- Throughput of the system (number of submitted jobs divided by the total time it takes for them to complete)

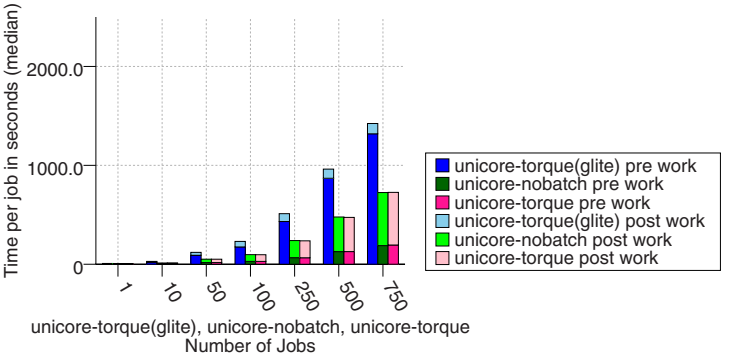

Fig. 2. UNICORE 5 Pre/Post work using Fork, Torque (vanilla) and Torque (gLite)

The figures in this section are generated using test data recorded by the testing tool. The data is then processed by a python script which then converts them into different graphs. From Fig 2. we see that the pre-work in case of Torque (vanilla) and no-batch is lower than the post work. For Torque (gLite) it shows that the pre work took more than half of the time of the post work. In Fig 3 we can notice that UNICORE performs equally when interfaced with Torque (vanilla) and with out any batch system (Fork)-measurements overlap in the figure. The line marked with triangles shows that UNICORE performs more slowly when interfaced with the gLite flavor of Torque. Fig 4 shows the throughput time and it is clearly shown the difference between UNICORE 5 when interfaced with Torque (vanilla) and when interfaced with Torque (gLite).

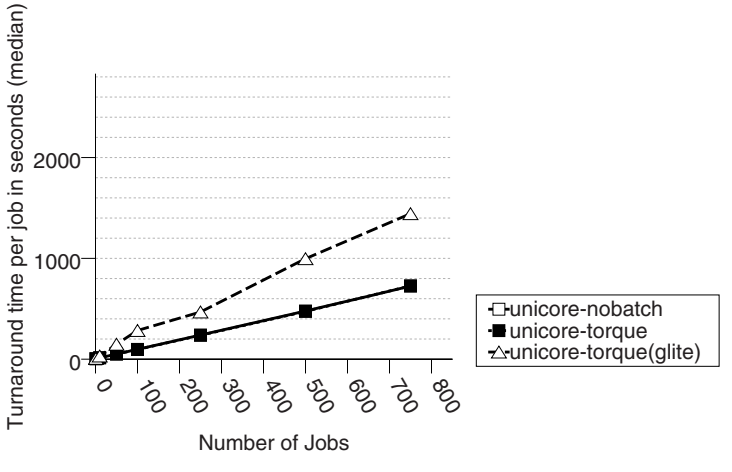

Fig. 3. UNICORE 5 turnaround time per job using Fork, Torque (vanilla), Torque (gLite). Note that lines from Unicore-torque and Unicore-nobatch are overlapping.

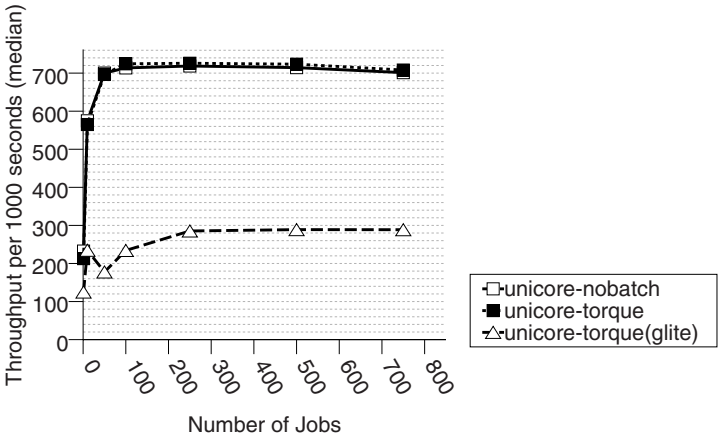

Fig. 4. UNICORE 5 throughput per 1000 seconds using Fork, Torque (vanilla), Torque (gLite). Note that lines from Unicore-torque and Unicore-nobatch are almost overlapping.

6 Summary and Future Work

In this paper we have presented a black-box approach to performance analysis of grid middleware and introduced a non-invasive platform independent architecture for evaluation tools to measure the overhead of the grid middleware and to quantify the effects of this overhead both on the throughput of the system and on the turnaround times of grid applications. Producing middleware component overhead measurements in a middleware independent and non-invasive manner and with high level of detail is currently difficult due to the differences among the middlewares. However, a use case that is relevant for all grids is the submission of jobs, their execution and the retrieval of their output data. Recording timestamps for when jobs are submitted, started, terminated and for when the output is returned, makes it possible to measure the overhead imposed on jobs by a middleware prior to and after execution and observe how this overhead changes when the number of submitted jobs varies.

As a future direction we intend to produce comparative performance data across different middlewares. Work is on-going for running tests using other middlewares including gLite, Globus and CROWN Grid in the context of the OMII-Europe Joint Research Activity on Benchmarking [7]. Extending the evaluation tool for UNICORE 6 and comparing the performance with that of UNICORE 5 is also planned.

We have provided a discussion on how exploiting this simple approach can produce performance analysis data that is beneficial to grid developers, users and grid site administrators. This approach can be further extended to provide comparative performance analysis across different grid middlewares. For proof of concept we presented the implementation details of the evaluation tool for

UNICORE 5 and the results obtained from running initial experiments with this tool. This approach is easily extended by introducing different types of jobs with specific resource requirements that would stress particular components of the middleware. To conclude, we have shown one approach to middleware independent and uniform performance evolution suites for grid middlewares that produce comparative performance measurements.

Acknowledgement

This work is supported by the European Commission through the OMII-Europe project-INFSO-RI-031844. For further information please refer to [7].

References

1. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Elsevier, Amsterdam (2004)
2. Nemeth, Z., Gombas, G., Balaton, Z.: Performance evaluation on grids: Directions, issues and open problems. In: 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2004), p. 290 (2004)
3. Snavely, A., Chun, G., Casanova, H., der Wijngaart, R.F.V., Frumkin, M.A.: Benchmarks for grid computing: a review of ongoing efforts and future directions. SIGMETRICS Perform. Eval. Rev. 30(4), 27–32 (2003)
4. Dikaiakos, M.: Grid benchmarking: Vision, challenges, and current status. Concurrency and Computation: Practice and Experience 19(1), 89–105 (2007)
5. Nmeth, Z.: Grid performance, grid benchmarks, grid metrics. In: Proceedings of the 3rd Cracow Grid Workshop, Cracow, pp. 34–41 (October 2003)
6. Tsouloupas, G., Dikaiakos, M.: Ranking and performance exploration of grid infrastructures: An interactive approach. In: Proceedings of the 7th IEEE/ACM International Conference on Grid Computing, Barcelona, September 28-29, 2006, pp. 313–315. IEEE Computer Society, Los Alamitos (2006)
7. Open Middleware Infrastructure Institute for Europe. Project no: RI03, –OMII-Europe (1844), `http://omii-europe.org`
8. The UNICORE project, `http://www.fz-juelich.de/zam/grid/unicore`
9. The UNICORE Plus project, `http://www.fz-juelich.de/unicoreplus`
10. UNICORE, Official website, `http://www.unicore.eu`