

This is a preprint of an article accepted for publication in Concurrency and Computation: Practice & Experience, copyright 2003-2004 John Wiley & Sons Ltd. The preprint can be shared for personal or professional use, but not for commercial sale or for any systematic external distribution by a third party.

Middleware Benchmarking: Approaches, Results, Experiences¹

Paul Brebner
Emmanuel Cecchet
Julie Marguerite
Petr Tůma

Octavian Ciuhandu
Bruno Dufour
Lieven Eeckhout
Stéphane Frénot
Arvind S. Krishna
John Murphy
Clark Verbrugge

Abstract

The report summarizes the results of the Workshop on Middleware Benchmarking held during OOPSLA 2003. The goal of the workshop was to help advance the current practice of gathering performance characteristics of middleware implementations through benchmarking. The participants of the workshop have focused on identifying requirements and obstacles of middleware benchmarking and forming a position on the related issues. Selected requirements and obstacles are presented in section 2 of this report, together with guidelines to adhere to when benchmarking in section 3, open issues of current practice in section 4, and perspectives on further research in section 5.

1. Introduction

For over a decade, middleware has been a well-established part of numerous applications. The spectrum of middleware implementations is diverse, ranging from communication-oriented middleware such as CORBA or SOAP libraries to component-oriented application servers such as the EJB or CCM containers. Naturally, an important characteristic of these implementations is their performance under various conditions. Such a characteristic is used both by the middleware developers to improve and advertise their middleware and by the middleware users to decide on their middleware of choice.

The natural way to obtain performance characteristics of a system is through benchmarking. But although middleware benchmarking is a relatively frequent endeavor [1][2][3][4][10][11][12][18][22][23], the practice is rather fragmented. Middleware developers use proprietary testing suites with results that are rarely comparable across the spectrum of middleware implementations. Middleware users rely on simplistic testing suites whose results are often prone to misinterpret-

¹ A report produced from the OOPSLA 2003 Middleware Benchmarking Workshop

ation when related to specific usage scenarios. Most standardization efforts in the area have so far also failed to fruit [24].

To remedy this situation, the participants of the Middleware Benchmarking Workshop held during OOPSLA 2003 shared their experience with designing, running and evaluating the existing benchmarks. As the next step, the participants identified some of the most significant obstacles encountered in the current practice and proposed approaches to tackle these obstacles. This report presents the results of the workshop in four sections, dedicated to the reasons for benchmarking, guidelines for benchmarking, open issues in benchmarking and perspectives on further research.

2. Why Benchmarking ?

An important statement emphasized repeatedly during the workshop debates was that a nature of a benchmark depends strongly on the intended use of the results. A benchmark that provides continuous data to an automated load balancing system will be different from a benchmark that provides static data to a system developer. The somewhat obvious character of this statement is outweighed by the less obvious nature of its implications, which touch all aspects of benchmarking from the benchmark design through the measurement mechanisms to the processing of results. The following sections focus on the use of benchmarking for the design and the evaluation of middleware and outline the implications of the considered use on the nature of the benchmark.

2.1. Benchmarking to Design Middleware

The first use of a benchmark considered important by the participants of the workshop was benchmarking to aid in the design of middleware. Essential to this use is benchmarking real applications to see how they stress the supporting system, collecting data to create models of supporting system usage typical for the selected application domains. Examples of such models are the workloads defined by the ECperf, RUBiS, SPEC jAppServer, SPEC JBB and TPC-W benchmarks, all focusing on the online business domains. Additional models are needed for other domains.

Model-based techniques can be applied to visually represent techniques for defining entities and their interactions in application domain using domain-specific building blocks. An example of a model-based benchmarking tool is the CCMPerf benchmarking tool [23], which is a model-based benchmarking suite that allows developers and end-users of the CORBA Component Model (CCM) [19] middleware to evaluate the overhead that CCM implementations impose above and beyond the CORBA implementation, as well as model interaction scenarios between CCM components using varied configuration options, to capture software variability in higher-level models rather than in lower-level source code. The model-based techniques used in CCMPerf help domain experts visualize and analyze the results of performance experiments more effectively since they focus on higher-level modeling abstractions, rather than wrestling with low-level source code.

During middleware design, benchmarking should be used to determine the optimal architecture for given constraints, such as memory capacity, processing power, or network throughput and latency. Important in this aspect is the task of validating models created during the design and thus predicting the real behavior of the architecture. Contemporary middleware is too complex to be understood purely through the analysis of its architecture, without benchmarking its real behavior.

Another important feature of the use of benchmarking to design middleware is the ability to capture and document consequences of using a specific architecture. An example of this approach is the work on design patterns for server side threading and dispatching models of TAO [16].

2.2. Benchmarking to Evaluate Middleware

The second use of a benchmark considered important by the participants of the workshop was benchmarking to evaluate middleware. This use of benchmarking includes comparing performance of various middleware architectures, as well as comparing performance of specific middleware configurations. Besides the obvious use of the comparisons for selecting architectures and configurations, perspective uses include gathering data for system sizing and for determining system state and devising autonomic computing policies.

An important task of the benchmarks is to evaluate the scalability of middleware. This implies benchmarking the behavior of the middleware and the supporting system when stretched to the limits of their scalability, as well as when used within the limits of their scalability.

3. Benchmarking Guidelines

The participants of the workshop also noted that many benchmarking projects tend to repeat common mistakes that devalue the results. Prominent among the common mistakes was an incorrect choice of metrics. Timestamps alone are not necessarily sufficient for understanding the results, annotating timestamps with system utilization and resource consumption data for the supporting system is useful.

A pitfall that can lead to useless results is using benchmarks that rely on an artificial workload such as microbenchmarks of an isolated feature of the middleware. It is difficult to conclude anything about the behavior of real applications from the results of such benchmarks, and their use should therefore be limited and well justified.

Measurement technique in benchmarking is equally important. Particular performance measures or metrics may have a variety of intertwined effects, and a change or improvement in one measurement will usually affect many other quantities as well. A reduction in the number of method calls executed due to a method-inlining optimization for instance, will increase code size which may alter cache performance. In order to make valid overall performance judgments, evaluations based on quantification must focus not only on providing a comprehensive representation of the benchmark that reveals important qualities, but also on understanding the inevitable dependencies between measurements.

Building a comprehensive view through measurement is itself challenging. A small, fixed suite of measurements is desirable since it makes comparisons simple and human comprehension feasible. Any reduced set of measures, however, may not give a true indication of benchmark activity, and is certainly unlikely to capture all possible quantities of interest. A more reasonable approach is to permit a rough assessment through a few, general measurements, but also calculate more specific measures in order to allow exploration of an apparent behavior through other perspectives and finer-grained detail. Evaluating a benchmark numerically is then a process of drilling down until one is satisfied that a clear understanding of the behavior has been determined [6].

Another pitfall is related to the interpretation of the results. The results must reflect the working of the measured feature and not the interference of other features. That is especially true for warming up the system under test to a steady state to filter out interference such as cache priming, which is known to take minutes for simple benchmarks and can stretch to hours when complex mechanisms such as garbage collection or database access are involved. A similar concern applies to the interference of the workload injection limits with the system under test.

It may not be easy or straightforward to assess performance. Execution activity at various levels from the lowest to the highest will alter performance, and so must be understood in order to give an accurate evaluation. This is particularly true of low-level concerns ; while the impact of higher level algorithm design, and even of code generation choices may be well-understood by an application developer, low level issues such as cache sizes, branch-predictor strategy, and so on can have a significant impact on the actual performance cost of various benchmark actions. For instance, an interpretation that a benchmark is making a lot of method calls may not be indicative of excessive execution overhead if each call is correctly predicted by the processor. A benchmark evaluation based on numerical data must therefore be careful to account for all possible system features.

As a general guideline, the participants of the workshop agreed that it is important to release exhaustive configuration information together with any results. Besides the obvious need for reproducibility of the results, the information can be used for assessing the impact of configuration changes on the results.

Note that interpretation of numerical data will only be valid when the measurement is precisely defined. Many analyses fail to include enough information about how the measurement is actually calculated, making it not only difficult to reproduce and verify results but also difficult to make comparative assessments. Lines of code, for example, a traditional measurement of static application size is easily perturbed by programming style and a variety of potential measurement possibilities (should it count blank lines, comment lines, etc). Choices by different experimenters will often differ, and so exact metric definitions have an obvious necessity.

4. Open Issues

Paramount of the open issues was the question of resources needed to conduct benchmarks, in terms of machinery, time to run the benchmarks, and expertise needed to understand the complete setup of the benchmarks. The participants of the workshop agreed that the resource requirements are often prohibitive to conducting benchmarks, especially in a research environment with stringent requirements on research results. An additional complication is that the lifetime of the expertise and the results is short, which increases the cost of benchmarking.

The results of a benchmark should be indicative of the performance of a real application. This implies the need for real workloads, but such workloads tend to be unsuitable for benchmarking because of their size. Determining what substitute workloads are representative of real workloads is an important open issue.

During microprocessor design, computer architects face similar problems due to extremely long simulation times – it is not exceptional that simulating one second of a real system takes 24 hours. Several techniques have been proposed to address this issue in the context of uniprocessor performance modeling. *Sampling* [5][17][20] refers to selecting samples from a complete program run. Since the number of sampled instructions is smaller than the total number of instructions in a complete program run, significant simulation speedups can be obtained. Important issues that need to be dealt with are which samples to select and how to guarantee a correct hardware state at the

beginning of each sample. A second technique is to use *reduced input sets* [8][13]. These reduced input sets are derived from longer reference inputs by a number of techniques: modifying inputs, truncating inputs, etc. The benefit of these reduced inputs is that the dynamic instruction count when simulating these inputs can be significantly smaller than for the reference inputs. A third approach is to apply *statistical simulation* [7][14][15]. The idea of statistical simulation is to define and collect a number of important program characteristics, to generate a synthetic workload from it which is several orders of magnitude smaller than a real application, and to finally simulate this synthetic workload. If the synthetic workload captures the right characteristics, the behavior of the synthetic workload and the real application will be similar.

A key issue that needs to be dealt with for all these techniques is to determine whether the substitute workload is representative for the real workloads. Previous work has shown that statistical data analysis techniques could be useful for that purpose [9]. We believe that adapting these methods to the context of middleware benchmarking could be extremely useful in speeding up measurement time while preserving the representativeness of the substitute workload. How to do this, however, remains an open issue.

With the middleware being just one part of a layered architecture, typically supported by the operating system or even another middleware, the results of middleware benchmarks depend not only on the middleware itself, but also on the supporting architecture. An open issue is how to abstract from the supporting architecture and characterize only the middleware layer, perhaps together with the interactions between layers.

A representative example of a layered architecture is the CORBA Component Model (CCM) [19], where the CCM applications sit on top of a CORBA ORB, which comprises of three layers. The *host infrastructure layer* is the middleware layer that provides a uniform abstraction over the underlying operating system. The *distribution middleware layer* is the middleware layer such as CORBA that abstracts the data marshaling and connection management facilities. The *common middleware services layer* is the middleware layer that provides services. Benchmarks for CCM implementations are thus heavily influenced by the underlying layers, and it is necessary to characterize the influence of these layers in isolation.

An open issue of an entirely different nature is dealing with the publication of results for commercial products. Such results can often damage any of the concerned sides through real or perceived bias, or through simple misinterpretation.

5. Perspectives

With the numerous issues of technical nature, it might be beneficial to keep a knowledge base of benchmarking expertise that would list such issues and thus help keep research free of engineering mistakes. The question that remains unanswered is how to keep such a knowledge base up to date. Similar reasoning supports the perspective of an open database of benchmarking results similar to [22]. Such a database could help extract different information from the same results depending on the nature of the research that uses the results. The database would require solutions to the issues of anonymity, reliability and credibility of the results. The related technical issue of common data format would also need to be solved to facilitate sharing the results and the tools to process them.

Another step beneficial to the benchmarking efforts would be to interest the middleware designers in supplying the recommended settings for specific applications and workloads. With representative workloads for selected application domains, this will help avoid publishing results

that are incorrect because of misconfiguration, especially for commercial products that often have complex or undocumented settings that influence benchmark results.

6. References

- [1] L. Boszormenyi, A. Wickener and H. Wolf. "Performance Evaluation of Object Oriented Middleware - Development of a Benchmarking Toolkit." Proceedings of the Fifth International Euro-Par Conference (EuroPar-99), Springer Verlag LNCS 1685, 1999.
- [2] H. R. Callison and D. G. Butler. "Real-time CORBA Trade Study." Boeing, 2000.
- [3] E. Cecchet, A. Chanda, S. Elnikety, J. Marguerite and W. Zwaenepoel. "Performance Comparison of Middleware Architectures for Generating Dynamic Web Content." Proceedings of the Fourth ACM/IFIP/USENIX International Middleware Conference, 2003.
- [4] E. Cecchet, J. Marguerite and W. Zwaenepoel. "Performance and Scalability of EJB Applications." Proceedings of the 2002 Object-Oriented Programming, Systems, Languages and Applications (OOPSLA-02), ACM SIGPLAN Notices, Vol. 37, No. 11, 2002.
- [5] T. M. Conte, M. A. Hirsch, and K. N. Menezes. "Reducing state loss for effective trace sampling of superscalar processors." Proceedings of the 1996 International Conference on Computer Design (ICCD-96), IEEE CS Press, 1996.
- [6] B. Dufour, L. Hendren and C. Verbrugge. "Problems in Objectively Quantifying Benchmarks using Dynamic Metrics." Sable Technical Report 2003-6, McGill University, 2003.
- [7] L. Eeckhout, S. Nussbaum, J. E. Smith, and K. De Bosschere. "Statistical Simulation: Adding Efficiency to the Computer Designer's Toolbox." IEEE Micro, Vol. 23, No. 5, 2003.
- [8] L. Eeckhout, H. Vandierendonck, and K. De Bosschere. "Designing Computer Architecture Workloads." IEEE Computer, Vol. 36, No. 2, 2003.
- [9] L. Eeckhout, H. Vandierendonck, and K. De Bosschere. "Quantifying the Impact of Input Data Sets on Program Behavior and its Applications." Journal of Instruction-Level Parallelism, Vol. 5, 2003.
- [10] S. Gokhale and D. C. Schmidt. "Measuring and Optimizing CORBA Latency and Scalability Over High-speed Networks." IEEE Transactions on Computers, Vol. 47, No. 4, 1998.
- [11] M. B. Juric and I. Rozman. "RMI, RMI-IIOP and IDL Performance Comparison." Java Report, Vol. 6, No. 4, SIGS/101 Publications, 2001.
- [12] M. B. Juric, I. Rozman, A. P. Stevens, M. Hericko and S. Nash. "Java 2 Distributed Object Models Performance Analysis, Comparison and Optimization." Proceedings of the 2000 International Conference on Parallel and Distributed Systems (ICPDAS-00), IEEE CS Press, 2000.

- [13] J. KleinOsowski, and D. J. Lilja. "MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research." *Computer Architecture Letters*, Volume 1, 2002.
- [14] S. Nussbaum and J. E. Smith. "Modeling superscalar processors via statistical simulation." *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques (PACT-2001)*, IEEE CS Press, 2001.
- [15] M. Oskin, F. T. Chong, and M. Farrens. "HLS: Combining statistical and symbolic simulation to guide microprocessor design." *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA-27)*, ACM Press, 2000.
- [16] D. C. Schmidt and C. O'Ryan. "Patterns and Performance of Distributed Real-time and Embedded Publisher/Subscriber Architectures." *Journal of Systems and Software*, Special Issue on Software Architecture Engineering Quality Attributes, 2002.
- [17] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. "Automatically characterizing large scale program behavior." *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, ACM Press, 2002.
- [18] P. Tůma and A. Buble. "Open CORBA Benchmarking." *Proceedings of the 2001 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS-01)*, SCS, 2001.
- [19] N. Wang, D. C. Schmidt and C. O'Ryan. "An Overview of the CORBA Component Model." In G. Heineman and B. Councill. "Component-Based Software Engineering." Addison-Wesley, 2000.
- [20] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling." *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA-30)*. ACM Press, 2003.
- [21] Center for Distributed Object Computing. "The ACE ORB (TAO)." Washington University, <http://www.cs.wustl.edu/~schmidt/TAO.html>.
- [22] Distributed Systems Research Group. "CORBA Benchmarking Project." Charles University, <http://nenya.ms.mff.cuni.cz>.
- [23] Institute of Software Integrated Systems. "CCMPerf: Model Integrated Test & Benchmarking Suite." Vanderbilt University, <http://www.dre.vanderbilt.edu/~arvindk/MIC/ccmperf.htm>.
- [24] Object Management Group. "White Paper on Benchmarking." OMG document bench/99-12-01, 1999.