

Current Trends in Middleware Benchmarking

L. Bulej, P. Tůma

Charles University Prague, Faculty of Mathematics and Physics, Prague, Czech Republic.

Abstract. The paper provides a brief overview of the state of the art in middleware benchmarking. The overview is used to highlight some of the outstanding issues in the area, specifically the issues related to implementing, running and evaluating regression benchmarks. The issues are analysed in depth and the course of our work towards resolving them is presented.

Introduction

Middleware benchmarking is an integral part of the distributed computing area, where it serves to satisfy an obvious need to evaluate and compare performance of numerous implementations of communication and application middleware standards, such as CORBA [*Object Management Group*, 2000], RMI [*Sun Microsystems*, 2002a], or EJB [*Sun Microsystems*, 2001].

Benchmark results are regularly used both by the middleware users, to compare performance and functionality of software products from different vendors, running on different hardware and software platforms, and by the middleware developers, to evaluate the performance of their product in critical areas, to assess implementation changes that may have an impact on performance, and to determine the extent of such impact. In addition, benchmarking can be also used for monitoring and diagnostics, and in less common cases, for estimation of middleware performance under different, previously untested, conditions [*Buble et al.*, 2000].

State of the Art

The area of middleware benchmarking is lacking a standard that would unify the process of middleware benchmarking in general, so that it would cover benchmarking of different middleware platforms and provide a common methodology for obtaining the data and guidelines for presentation of the results. For specific middleware platforms, the situation is better due to demand for comparable and easy to understand results.

The most advanced in this regard appears to be the area of CORBA benchmarking, mostly through the efforts of OMG and its White Paper on Benchmarking [*Object Management Group*, 1999], which standardizes the terminology, methodology and outlines the basic requirements for open, easy to measure and understandable benchmarks. As for other middleware platforms, such as EJB, the standardization exists in the form of widely accepted benchmarks, which are used throughout the industry for performance evaluation and comparison.

At present, we are aware of two principal types of benchmarks being widely used in the area, termed model-application benchmarks and feature-specific benchmarks. These differ substantially in the methodology for obtaining the data and in the presentation of results, but their popularity suggests that both types of benchmarks have its use and audience. Depending on the expected type of results, the audience can be split into the group of middleware users and the group of middleware developers, as described in [*Tůma et al.*, 2001].

Model-application benchmarks

Model-application benchmarks usually simulate a model application and yield a set of numeric values expressing the performance of the benchmark application. Often, the results consist of mere two values, of which one represents the number of model-application operations per second achieved on a given hardware and software platform, while the other amounts to the cost of a single operation. This practice is consistent with that of the Transaction Processing Council (TPC) benchmarks for database systems.

While the results produced by such benchmarks are easy to understand, it is very difficult to make any assumptions about performance of a system under different workloads or for other application types, since benchmarks of this kind do not collect data concerning the lower-level application-independent functionality of the middleware platform used.

To provide an example, consider the EJB platform, which does have an officially standardized benchmark suite for measuring performance and scalability, called ECperf [*Sun Microsystems*, 2002b]. The suite measures performance of a model application simulating real-world business problems with respect to product manufacturing, order, inventory, and supply chain management. The benchmark results in a pair of values representing operations per second and a cost of a single operation.

RUBiS [*ObjectWeb Consortium*, RUBiS] is another model-application benchmark, this time based on a prototype auction site, modelled after eBay.com. The benchmark is supposed to be used to evaluate application

design patterns and application servers performance scalability. SPECjbb2000 [*Standard Performance Evaluation Corporation, SPECjbb2000*] is a benchmark emulating a 3-tier system with business logic engine in the middle tier, which is currently the most common type of server-side Java application. This enumeration is by no means complete and serves for illustration purposes only.

Feature-specific benchmarks

The other type of benchmarks is intended to provide very detailed coverage of individual aspects of given middleware platform, including aspects specific to a particular implementation of that platform. The content of the results produced by such benchmarks usually consists of very detailed and technical data, supposed to be read and understood by the developers of the particular middleware platform.

In measuring performance of isolated aspects typical for a specific middleware platform, these benchmarks provide information about behaviour of the individual aspects on a specific hardware and software platform and under various situations, such as heavy parallelism, network distribution, etc. Middleware exhibiting no anomalies in the behaviour of its individual aspects is expected to be performing well in the overall scope.

An example of such feature-specific benchmarking suite is the Open CORBA Benchmarking Suite [*Tůma et al., 2001*] for performance evaluation of CORBA middleware. In dealing with the issues mentioned in the White Paper on CORBA Benchmarking, the suite serves as a reference for conducting CORBA benchmarks.

To our best knowledge, there is no comparable benchmark suite for other middleware platforms, especially RMI and EJB. As for CORBA, the distribution of TAO [*Distributed Object Computing Group, TAO*], an open source implementation of the OMG CORBA specification, contains benchmarks measuring selected aspects of the CORBA ORB performance. The Open CORBA Benchmarking suite differs from the collection of benchmarks found in TAO in that it collects more detailed data and that it can be easily used for different ORB implementations, because it is not tied to a particular ORB at the source code level.

Regression Benchmarking

Benchmarks can be very helpful during middleware development. Our experience from a series of CORBA performance evaluation and comparison projects shows that systematic benchmarking of middleware primitives can reveal performance bottlenecks and bad design decisions as well as errors in the implementation. In other words, detailed, extensive and repetitive benchmarking can be used for finding regressions in middleware implementations, hence the term regression benchmarking.

Even though it is beyond doubt that regression benchmarking is a valuable tool in the course of middleware development, our practical experience indicates it is rarely used, both in the commercial and the non-commercial/open-source domains. As for the open-source middleware implementations, the only exception to the rule appears to be the already mentioned TAO, which, besides having a suite of regression tests to validate correct functionality, also has a number of benchmarks. These, however, are not automated enough to perform extensive testing and performance evaluation.

It is more difficult to comment on the development practices of commercial closed-source middleware vendors, as their practices are closed and our knowledge is thus limited to our experience with their products. Nevertheless, in the past years we have revealed several performance problems in leading commercial ORB implementations, which ranged from minor flaws to major scalability issues. These problems would probably have been found had the software been subjected to regression benchmarking. From this, we conclude that even in the commercial sphere, regression benchmarking has not yet found a regular use. The fact that most available benchmark results appear to have been edited and formatted by hand also indicates that the benchmarks are not conducted very often, which only supports our argument.

In our analysis of why the use of regression benchmarks is not more widespread, a major factor appears to be the fact that the initial cost of setting up regression benchmarking for a software project is perceived to be higher than the potential benefits. Therefore, there is little or no incentive to invest resources in it. Closer look at the initial costs reveals what we believe are the major issues causing the perception of unreasonably high costs:

1. Creation of benchmarks

Creating regression benchmarks requires developer resources, which may not be readily available. An important factor is the amount of work required to benchmark a specific functionality on a given platform. Unless the effort/cost necessary to create and, which is more important, update and extend the regression benchmark suite is reasonably small, the developers and especially the managers are reluctant to invest effort into the work, which does not provide immediate benefits.

2. Execution of benchmarks

Even when an effort has been invested into creation of a suite for regression benchmarking, our experience shows that it's not trivial to conduct the benchmarks, and conduct them repeatedly,

which is the purpose of regression benchmarks. There can be hundreds of tests, the whole suite can take days or weeks to complete and the amount of collected data can easily reach the order of gigabytes. Without a mechanism for automated execution and collection of benchmark data, the task is almost impossible. Also, for the benchmarks to be conducted on daily basis, the execution time of the entire suite is an issue and requires some control mechanisms.

3. *Evaluation of benchmarks*

The data collected during automated execution of the regression benchmarking suite are mostly raw numbers in amounts preventing direct analysis by humans. The data must be processed and the size and detail reduced to a useful level. It should be possible to evaluate the data automatically to a certain degree, at least to identify anomalies and to alert the developers to pay extra attention to the results. As in the previous case, without automated processing facilities, the benchmark results are merely gigabytes of useless data.

Based on our experience with industrial partners, we have created a regression benchmark suite, which attempts to address the above issues using the following concepts:

Benchmarking Framework

The solution to problem (1) involves comfortable writing of new benchmarks, while a partial solution to problem (2) should handle automated benchmark execution and collection of the results. To address these issues, we have created a benchmarking framework with the following properties.

Very simple creation of new benchmarks

To be of any use to the middleware developer, the benchmarking framework must not get in the way. If there is a need to write benchmark for specific functionality, it is enough to write the code to test the functionality and reuse the generic facilities of the framework, such as support for data acquisition, configuration, multi-threading, variation of test parameters, etc.

Portability and extensibility of the benchmark suite

If a particular middleware supports multiple platforms, the developer will want to be able to compare the performance of the middleware on those platforms. Our framework is reasonably portable to commonly used software and hardware platforms, including several operating systems, compilers, different version of the middleware implementation and vendor-specific tools. Adding a new broker is a matter of minutes, adding a completely new platform takes longer because of the platform dependent code which has to be supplied to the framework.

Automated compilation and unattended execution

Manual compilation, configuration and execution of suite benchmarks, as is the case of some application oriented benchmarks for EJB servers is simply not an option. Our suite contains hundreds of autonomous tests and supports running them either locally, or remotely on two different nodes on the network. The Cygwin environment is used to support remote execution of benchmarks on machines running Windows NT class of operating systems.

Sample Collection Mechanism

Since we want the regression benchmarks to return very detailed information, we cannot settle for collecting only averages, minima, and maxima of the observed values. Very often, the distribution of the samples, its median and inter-quartile range are much more telling than just average values, especially when we are interested in real time behaviour or in comparing performance of a particular middleware implementation on various platforms. In addition to average, minimal, and maximal values, our framework therefore collects individual samples for a selected thread of execution as well.

To avoid interference caused by just-in-time compilation, disk cache flushes, stabilization of adaptive resource allocation algorithms and other phenomena accompanying application start up, the framework throws away data from certain period at the beginning of the measurement.

Automated adjustment of benchmark execution time

In our framework, the execution time of a single test is determined by the length of the warm up period, the time necessary to collect resource usage data both on the client and on the server, and the time required to collect the individual samples from the selected thread and the averaged data. The lengths of these periods are fixed and

contribute significantly to the total execution time of a single test, along with the time required to collect a fixed number of individual samples.

The lengths are selected conservatively so that we can assume we collect more than enough samples to obtain accurate data for possible further processing. Some of the values (such as warm up period) are probably very pessimistic, resulting in longer execution times than necessary, but we prefer accurate data (and longer execution) to inaccurate data. Currently, the amount of data gathered from execution of the complete regression benchmark tends to rank in order of gigabytes and the running time is measured in days or weeks.

Such running times are not so disastrous for extensive performance evaluation of particular middleware implementation, but cannot be used for day-to-day measurements, which are of interest to the developers. To reduce the running time of the benchmark, it is necessary to be able to control the amount of data gathered without compromising the credibility of the data.

We expect to be able to limit the execution time by specifying certain parameter representing the required accuracy of the collected data. As a result, the framework should only collect the right amount of data to satisfy the accuracy requirement. This would produce significant savings in the overall execution time. As a bonus, we would need to collect and process less data. Along with determining the minimal number of samples that need to be collected, we would like the framework to be also able to determine the length of the warm up period.

Assuming that we can find the appropriate statistical methods to estimate the length of the warm up period and the minimal number of samples required for specified accuracy of the data, there is one more issue: outliers.

The observed data will most likely contain outliers caused by unpredictable and hardly avoidable events, such as process rescheduling and other phenomena caused by the operating system interference. These outliers should not be hidden from the developer so that he or she can see the real behaviour of the middleware on a particular platform. On the other hand, these outliers will be potentially harmful for statistical methods that make use of variance σ^2 or sample variance s^2 , both of which are very sensitive to outliers in the population sample. We therefore need a mechanism to detect the outliers and exclude them from the parameter estimation process.

Possibly useful statistical methods

The following section presents an overview of the statistical methods we consider suitable for controlling the minimal amount of data along with a short explanation of how we plan to use it. Fortunately for us, sequential analysis is a quite popular scientific discipline with contributors from the simulation area. This is subject to future work though, since the analogies need to be confirmed.

The values observed by the benchmark can be considered random, but, except for few specific cases, we cannot make any assumptions about the distribution of those values. The results from the simulation community mostly apply to stationary processes, or, in other words, processes whose mean, variance and autocorrelation structure do not change over time.

- *Outlier detection and removal*

We plan to use outlier detection methods to identify and hide outliers from the sequential estimation methods that operate with sample variance s^2 .

- A simple method that can be used to detect outliers is the application of the Chebychev inequality and sample mean absolute deviation as an estimator of population standard deviation. Given the nature of outliers in our case, this method may be too pessimistic even for our purposes and throw away reasonable data.
- A more complex procedure to identify the outliers is based on an analysis of the distance to an example's nearest neighbours. [Knorr et al., 2000] defines distance based outlier as follows: *An object O in a dataset T is a $DB(p,D)$ -outlier if at least fraction p of the objects in T lies greater than distance D from O .* We need to find the right values of the p and D parameters, or better find a way to determine them automatically for the method to give us the desired results.

- *Quantile and histogram estimation*

We may want to use the quantile and histogram estimation instead of the sequential mean estimation to determine the minimal number of samples. This method should be considerably less susceptible to enormous deviations in the presence of outliers.

- A simple method for obtaining confidence interval for a population quantile is to use the standard non-parametric estimation based on the order statistics. This may be the right method for our purposes.
- [Chen et al., 2001] discusses an implementation of a sequential procedure to construct proportional half-width confidence intervals for simulation estimator of the steady-state quantiles and a histogram of stochastic processes.

- [Chen, 2002] describes an implementation of a two-stage procedure to construct confidence intervals for a simulation estimator of the steady-state quantiles of stochastic processes. The algorithm dynamically increases the sample size so that the quantile estimates satisfy the proportional precision at the first phase and the relative or absolute precision at the second phase.
- *Warm up period length estimation*
This is a non-trivial task to solve automatically, we will probably have to use a parametrized but still empirical method. There is a number of methods for estimation of the length of the warm up period in the simulation area, ranging from graphical methods, through heuristic approaches and statistic methods to various hybrid methods. As it appears now, however, the problem is better defined in the simulation area in that it usually estimates the length of warm up period of a stochastic process, which can be formally described, whereas in our case, we would like to estimate the time necessary to filter out phenomena accompanying an execution of a program in an operating system, i.e. priming of processor caches, disk cache flushes, just in time compilation.
- *Minimal number of samples estimation*
The basic idea is to specify the relative precision of the mean and determine the number of samples needed to ensure that the mean will be within the confidence interval with probability $(1 - \alpha)$. The straightforward two-stage procedure suggested by [Stein,1945] cannot be used, because the observed values do not have the normal distribution.
Since the sample mean does not have the normal distribution, we may as well group the observed values, compute the sample means for individual groups and thus obtain a sequence of sample means, which is in fact a sequence of random variables with the normal distribution. This assumption is also used in a class of sequential methods called batch means procedures. These procedures use the grand mean of batch means as the estimator of the mean value of the population distribution. Confidence interval is then computed using the sample variance of the batch means and quantiles of Student's t-distribution.
 - [Nakayama, 1994] describes a modification of Stein's two-stage procedure to work with the method of batch means. This is the primary method of interest for determining the minimal number of samples, but does not work very well in presence of outliers.
 - [Hlavka, 2000] describes a three-stage procedure, which improves the asymptotic behaviour of Stein's two-stage procedure by adding an additional sampling stage. Modification of this procedure to work with the method of batch means would be interesting.
 - [Steiger et al., 2000] summarizes the results of an extensive experimental performance evaluation of selected batch means procedures ABATCH, LBATCH and ASAP. These methods are iterative and their operation may depend on the results of additional tests performed on the sequence of batch means, such as von Neumann test for independence or Shapiro-Wilk test for multivariate normality. This is the case of the ASAP method [Steiger et al., 1999], which can adaptively change the batch size and the number of batches in response to the results of the above mentioned tests. While the application of such methods seems attractive, we would like to achieve our goals using simpler (and less computationally intensive) methods so as not to disturb the measurement by additional load. We plan to evaluate the iterative methods in case of failure of the simpler methods.

Result Processing Mechanism

The amount and the character of data collected from the execution of the whole benchmark suite rules out any form of human processing of the initial results. To solve problem (3), automation is a must to reduce the amount of data.

Automated processing and evaluation of the results

Even if we had optimal estimates of the minimal number of required samples, we would still have to process rather significant amounts of data. Currently, our framework is able to automatically generate HTML reports with dozens of images, which are much more suitable for human inspection than the raw data. The automatic evaluation of results is left as a topic for future work, where we would expect to be able to detect anomalies in comparison with previous runs of the benchmark.

Conclusion

We have provided an overview of the state of the art in middleware benchmarking, which we have extended with the regression benchmarking approach. To do so, we have pointed out some of the major outstanding issues related to regression benchmarking and sketched the course in which we plan to address the issues in our future work.

To our best knowledge, we are not aware of related work comparable to our approach. Of course, there is a large number of various benchmarking projects for CORBA and EJB middleware, but none of them is currently comparable to our work either in the methodology used for conducting the measurements, or in the test coverage in case of CORBA benchmarking.

In the near future, we plan to perform experiments using statistical methods to control the minimal amount of samples depending on the requested accuracy of the data and methods for determining sufficient warm up period as well as methods for detecting outliers in the population sample. Optional prototype results will be available at our website at <http://nenya.ms.mff.cuni.cz>.

References

- Buble, A., Tůma, P.; On Benchmarking Object-Oriented Middleware, *Proceedings of the Week of Doctoral Students 2000 conference (WDS 2000)*, Faculty of Mathematics and Physics, Charles University, Prague, Jun 2000.
- Chen, E. J., Kelton, W. D.; Quantile And Histogram Estimation, *Proceedings of the 2001 Winter Simulation Conference*, 2001
- Chen, E. J.; Two-Phase Quantile Estimation, *Proceedings of the 2002 Winter Simulation Conference*, 2002
- Distributed Object Computing Group; TAO: The ACE ORB, <http://www.cs.wustl.edu/~schmidt/TAO.html>
- Hlavka, Z.; Robust Sequential Methods, *Ph.D. Thesis, Department of Probability and Statistics, Faculty of Mathematics and Physics, Charles University*, 2000
- Knorr, E. M., Ng, T. R., Tucakov, V.; Distance-Based Outliers: Algorithms and Applications, *VLDB Journal*, 8(3-4):237–253, 2000
- Nakayama, M. K.; Two-stage stopping procedures based on standardized time series, *Management Science* 40, 1189–1206, 1994
- Object Management Group; White Paper on Benchmarking, *OMG bench/99-12-01*, 1999.
- Object Management Group; The Common Object Request Broker: Core Specification, *version 3.0.2, OMG formal/02-12-02*, Dec 2002.
- ObjectWeb Consortium; RUBiS: Rice University Bidding System, <http://rubis.objectweb.org>
- Standard Performance Evaluation Corporation; SPECjbb2000: Java Business Benchmark, <http://www.spec.org/jbb2000>
- Sun Microsystems, Inc.; Enterprise JavaBeans Specification, *version 2.0, Final Release*, Aug 2001.
- Sun Microsystems, Inc.; ECperf Specification, *version 1.1, Final Release*, Apr 2002b.
- Sun Microsystems, Inc.; Java Remote Method Invocation Specification, *Java 2 SDK, version 1.4.1*, 2002a.
- Stein, C.; A two sample test for a linear hypothesis whose power is independent of the variance, *Annals of Mathematical Statistics*, 1945
- Steiger, N. M. 1999. Improved batching for confidence interval construction in steady state simulation, *Proceedings of the 1999 Winter Simulation Conference*, 1999
- Steiger, N. M., Wilson, J. R.; Experimental performance evaluation of batch means procedures for simulation output analysis, *Proceedings of the 2000 Winter Simulation Conference*, 2000
- Tůma, P., Buble, A.; Open CORBA Benchmarking, *Proceedings of the 2001 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2001)*, published by SCS, Orlando, USA, Jul 2001