

Regression Benchmarking in Middleware Development *

Lubomír Bulej
Charles University in Prague
Faculty of Mathematics and Physics
Malostranské náměstí 25
Prague, Czech Republic
lubomir.bulej@mff.cuni.cz

Petr Tůma
Charles University in Prague
Faculty of Mathematics and Physics
Malostranské náměstí 25
Prague, Czech Republic
petr.tuma@mff.cuni.cz

Academy of Sciences of the Czech Republic
Institute of Computer Science
Pod Vodárenskou věží 2
Prague, Czech Republic
bulej@cs.cas.cz

ABSTRACT

The paper advocates the concept of regression benchmarking as a part of middleware development and quality assurance process and highlights some of the outstanding issues related to implementing, running and evaluating regression benchmarks. The issues are analyzed in depth and the course of our work towards resolving them is presented.

1. 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 [10], RMI [17], or EJB [16].

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.

Regression testing as a part of the development and release process is slowly becoming a standard in many open source projects. The complexity of the software, distributed development model and the demand for certain level of quality assurance has led many open source projects to adopt

some form of regression testing.

Many of the open source middleware projects, such as TAO [6], OpenORB [1], or CAROL [11] to name a few, employ some form of regression testing in their development process. The most advanced in this regard appears to be ACE [4] and TAO with the Distributed Scoreboard [5], which collects results of every day's building and testing on various platforms from many places around the world.

Regression testing has many potential uses and can be applied to source code, libraries and whole applications. Source code testing aims to ensure syntactically correct code base and portability to various platforms, while run time testing guarantees correct functionality of various subsystems or functional units.

Performance is an aspect, which is not often subject to regression testing. Being mostly orthogonal to correct functionality, it often plays a minor role. Nevertheless, in many cases performance can be a major concern, especially in communication and application middleware projects. So far though, performance evaluations have been limited to occasional measurements and comparisons, while automated, thorough, and repetitive benchmarking has been neglected in most of the projects. This kind of benchmarking would allow for assessing performance impacts of various changes during development, tracking performance of individual releases, and could play an important role in the quality assurance process.

2. REGRESSION BENCHMARKING

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 implementation errors. 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-

*This work was partially sponsored by the Grant Agency of the Czech Republic grant number 201/03/0911.

source domains. As for the open-source middleware implementations, the only exception from the fact 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. The collection, however, does not support automation, which is needed to perform repeated extensive testing and performance evaluation.

Commenting on the development practices of commercial closed-source middleware vendors is more difficult, as their practices are not public 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 regression benchmarking has not yet found a regular use even in the commercial sphere.

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. As a consequence, there is very little incentive to invest resources in it. Closer look at the initial costs reveals what we believe are the major issues causing the discouraging perception:

1. *Creation of benchmarks*

Creating regression benchmarks requires developer time, which is a resource that 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 required to create, update and extend the regression benchmark suite with new feature-specific benchmarks is reasonably small, the developers and/or project leaders are reluctant to invest valuable resources into work, that does not provide immediate profit.

2. *Execution of benchmarks*

Even when there is enough resources to create a regression benchmarking suite, our experience shows that it is non-trivial to conduct the actual measurements and conduct them repeatedly, which is essential for regression benchmarking. The problem is caused by the fact that the benchmark suite can contain hundreds of individual benchmarks and take days or weeks of running time to complete. 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 on the verge of possibility. Also, for the regression benchmarks to be conducted on daily basis, the execution time of the entire suite becomes an issue and requires a control mechanism.

3. *Evaluation of benchmarks*

The data collected during automated execution of the regression benchmarking suite are mostly raw numbers, in amount, which prevents direct human analysis. The data must be first processed by a machine and the size and detail reduced to a manageable level. An important requirement is automatic evaluation of the data with respect to previous runs of the suite, which

should be able to detect anomalies and 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, and which is centered around the following concepts:

2.1 Benchmarking Framework

The solution to the issue (1) above requires the suite to be easily manageable. Part of the solution to issue (2) requires automated feature-specific benchmark execution and collection of the results. To address these issues, we have focused on the following features when creating our benchmarking framework.

Easy creation of new benchmarks

To be of any use to a middleware developer, the benchmarking framework itself must not constitute an obstacle. To benchmark a specific middleware functionality on an already supported platform, the developer only needs to write the code to test the functionality and reuse the generic facilities provided by 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, it is desirable 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 versions of the middleware implementations and vendor-specific tools. Extending the suite to support a new broker is a matter of minutes, adding support for 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 individual benchmarks and supports running them either locally, or remotely on two different nodes on the network. The Cygwin [12] environment is used to support remote execution of benchmarks on machines running the Windows NT class of operating systems.

2.2 Sample Collection Mechanism

Since we want a regression benchmark to produce 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 the averages, especially if we are interested in real time behavior or in comparing performance of a particular middleware implementation on different platforms. Therefore, in addition to average, minimal, and maximal values, our framework also collects individual samples for a selected thread of execution.

In our framework, the execution of a single benchmark progresses through several stages and the amount of time required for a single benchmark run depends on the time

spent in the execution of the individual stages. The first stage is warm up, the second is collection of resource usage data both on the client and on the server, the third is collection of individual samples from a selected thread of execution, and the fourth is collection of averaged data.

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 the *warm up stage*, a fixed time interval at the beginning of the measurement.

Automated adjustment of benchmark execution time

With the exception of the third stage, all stages have a fixed execution time. The execution time of the third stage depends on the time it takes to collect a fixed number of individual samples of the duration of the measured action. As such, the execution time of the third stage is not known in advance, because it depends on the complexity of the measured action. This has a major impact on the total execution time of a single benchmark.

The execution times of individual stages and the number of samples to be collected in the third stage have been chosen rather conservatively, so that we collect more than enough samples to obtain accurate data for further processing. Some of the values (such as the duration of the warm up stage) are probably very pessimistic, resulting in longer execution times than necessary. Nevertheless, we prefer accurate data (and longer execution) to inaccurate data. Currently, the amount of data gathered from execution of the complete regression benchmark suite ranks in the order of gigabytes and its running time is measured in days or weeks.

Such running times are not so disastrous for a one-time extensive performance evaluation of particular middleware implementation, but are hardly acceptable for day-to-day measurements, which are of interest to the developers. To control the execution time of individual benchmarks, we need to determine at run time the appropriate duration of the fixed-length stages and the number of samples to be collected during the third stage. This must be done without compromising the credibility of the data.

We expect to control the execution time of the third stage by specifying the required accuracy of statistics computed from the collected data. By being able to collect only the number of samples necessary to satisfy the accuracy requirement, we can control the execution time of the most unpredictable (in terms of execution time) stage of the benchmark, which would in turn produce significant savings in the overall execution time of the whole suite. Besides determining the number of samples that need to be collected, we would like the framework to be also able to determine the appropriate length of the warm up stage.

Depending on the type of an individual benchmark, the observed values can be modeled by response variable with a single constant or variable factor and a random error. This fact led us to evaluate application of statistical methods for data processing and for estimation of benchmark timing parameters. Assuming that we can find the appropriate statistical methods to solve the above issues, we have to take care of one more issue, outliers.

The observed data will contain outliers caused by unpredictable events that are hard to avoid, such as process rescheduling and other interference caused by the operating

system. Such outliers should not be hidden from the developer so that it is possible to observe the real behavior of the middleware on a particular platform. However, the outliers will considerably skew sample variance s^2 , which is very sensitive to extreme values in the population sample. Therefore, we need a mechanism to detect the outliers and exclude them from the benchmark parameter estimation process controlled by statistical methods based on sample variance.

Potentially applicable statistical methods

The following section presents a brief overview of the statistical methods we consider suitable for solving some of the issues above. Most of the methods come from the area of sequential statistical analysis, which is quite popular scientific discipline with many contributions from the area of discrete-event simulations.

Whether we can really benefit from the research done in that area is still subject to future work though, mainly to confirm that certain approaches used in discrete-event simulations can be used for middleware benchmarking.

- *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 approach to detecting outliers is the application of the Chebychev inequality, using 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 valid data points.

A more complex, yet promising approach is to identify the outliers based on the analysis of the distance to an example's nearest neighbors. Our task would be to devise a method for determining the parameters for distance-based outliers as defined by Knorr [8] in order to be able to correctly classify the measured data.

- *Quantile and histogram estimation*

Outliers caused by the middleware behavior should not be removed by the outlier detection and removal process. If the remaining outliers still cause unacceptable deviations and considerably harm the performance of methods for sequential mean estimation, we may need to evaluate methods for quantile and histogram estimation, which should be less susceptible to such deviations.

A simple method for obtaining confidence interval for a population quantile is to use the standard non-parametric estimation based on the order statistics. While this methods may be sufficient for our purposes, we may also need to evaluate and experiment with other methods.

Another approach would be to evaluate application of sequential and two-stage procedures for constructing confidence intervals for simulation estimators of steady-state quantiles of stochastic processes, described by Chen in [3] and [2].

- *Warm up stage length estimation*

This is a non-trivial task to solve automatically, since 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, all of which is rather hard to formalize. We will probably have to use an empirical method with a mechanism for platform dependent parametrization.

- *Required 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 $(1 - \alpha)$, α being the significance level.

Because the observed values do obey the normal distribution, we cannot use the two-stage procedure suggested by Stein [15]. Instead, we have to work with sequential methods termed *batch means procedures*, which assume that a sample mean of a batch of samples obeys the normal distribution. According to the central limit theorem, this assumption is valid.

There is a significant amount of work done in the area, and we plan to evaluate using modification of Stein's two-stage procedure for use with batch means described by Nakayama [9], which show promising results, but only when combined with effective outlier removal.

We may evaluate modifying three-stage procedure described by Hlavka [7] for use with batch means, but prior to that we will try to apply adaptive sequential procedures. Considering extensive performance evaluation of selected sequential methods conducted by Steiger [14], we may want to experiment with method called ASAP [13].

While the application of the adaptive methods seems attractive, we would prefer to achieve our goals using simpler (and less computationally intensive) methods so that we do not disturb the measurement by additional load caused by complex computations.

2.3 Result Processing Mechanism

The amount and the nature of the data collected from the execution of the whole benchmark suite practically rule out any form of human processing of the collected data. To address issue (3) mentioned above, automation is a must to reduce the amount of data to a level manageable by humans.

Automated processing and evaluation of the results

Even if we have optimal estimate of the number of samples that need to be collected in the variable-length stage of the benchmark, we still have to process significant amount 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 expect to be able to detect anomalies in comparison with previous runs of the benchmark.

3. CONCLUSION

We have presented the regression benchmarking approach as a part of middleware development and quality assurance process. 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.

In the near future, we plan to perform practical experiments using statistical methods to estimate the minimal number of samples required to satisfy a predefined accuracy of various statistics such as mean, median, or quantiles, methods for determining the appropriate length of the warm up stage as well as methods for detecting outliers in the measured population sample. Optional prototype results will be available at our website at <http://nenya.ms.mff.cuni.cz>.

4. REFERENCES

- [1] *OpenORB: The Community OpenORB Project*. <http://openorb.sourceforge.net>.
- [2] E. J. Chen. Two-Phase Quantile Estimation. In *Winter Simulation Conference*, 2002.
- [3] E. J. Chen and W. D. Kelton. Quantile and Histogram Estimation. In *Winter Simulation Conference*, 2001.
- [4] Distributed Object Computing Group. *ACE: The ADAPTIVE Communication Environment*. <http://www.cs.wustl.edu/~schmidt/ACE.html>.
- [5] Distributed Object Computing Group. *ACE+TAO Distributed Scoreboard*. <http://tao.doc.wustl.edu/scoreboard>.
- [6] Distributed Object Computing Group. *TAO: The ACE ORB*. <http://www.cs.wustl.edu/~schmidt/TAO.html>.
- [7] Z. Hlavka. *Robust Sequential Methods*. PhD thesis, Faculty of Mathematics and Physics, Charles University, Prague, 2000.
- [8] E. M. Knorr, R. T. Ng, and V. Tucakov. Distance-Based Outliers: Algorithms and Applications. *VLDB Journal: Very Large Data Bases*, 8(3-4):237–253, 2000.
- [9] M. K. Nakayama. Two-Stage Stopping Procedures Based on Standardized Time Series. *Management Science*, 40:1189–1206, 1994.
- [10] Object Management Group. *The Common Object Request Broker: Core Specification*, Dec 2002. OMG formal/02-12-02, version 3.0.2.
- [11] ObjectWeb Consortium. *CAROL: Common Architecture for RMI ObjectWeb Layer*. <http://carol.objectweb.org>.
- [12] Red Hat, Inc. *Cygwin: A Unix-like environment for Windows*. <http://www.cygwin.com>.
- [13] N. M. Steiger and J. R. Wilson. Improved Batching for Confidence Interval Construction in Steady-State Simulation. In *Winter Simulation Conference*, pages 442–451, 1999.
- [14] N. M. Steiger and J. R. Wilson. Experimental Performance Evaluation of Batch Means Procedures for Simulation Output Analysis. In *Winter Simulation Conference*, pages 627–636, 2000.
- [15] C. Stein. A Two-Sample Test for a Linear Hypothesis Whose Power is Independent of the Variance. *Annals of Mathematical Statistics*, 16(3):243–258, Sep 1945.
- [16] Sun Microsystems, Inc. *Enterprise JavaBeans Specification*, Aug 2001. Version 2.0, Final Release.
- [17] Sun Microsystems, Inc. *Java Remote Method Invocation Specification*, 2002. Revision 1.8, Java2 SDK, version 1.4.