

# METHOD INPUT PARAMETERS AND PERFORMANCE OF EJB APPLICATIONS

ALEXANDRE OUFIMTSEV AND LIAM MURPHY

---

**Abstract.** We investigated the impact of method input parameters on component performance, which is usually neglected during an application design stage. We evaluated a set of Commercial Off-The-Shelf (COTS) components which use Enterprise Java Beans (EJB) technology. These components were deployed on WebSphere Application Server and tested using a custom-built remote client. The client keeps track of execution times, while the server has JProbe Profiler embedded in the EJB container to monitor the interactions. For test purposes we used only stateless session beans, allowing us to concentrate on the possible dependency of server performance on input parameter variations. Test results show that significant performance impacts can be caused by a relatively small number of input parameter dependencies.

**Keywords:** EJB container, COTS, method input parameters, performance, profiling

---

## 1. Introduction

Software components are one of the most promising ideas extant for the efficient design of quality software systems. The software industry has realized that robust, reliable and scalable technology is needed to support their enterprise-scale, e-business systems. Middleware and component technologies have emerged as a base infrastructure for running advanced e-business systems. The predominant ones in use today are J2EE from Sun Microsystems, .Net and COM+ from Microsoft and CORBA from Object Management Group.

One of the major problems in building large-scale enterprise systems is anticipating the performance of the eventual solution before it has been built. This problem is especially germane to modern Internet-based e-business applications, where failure to provide high performance and scalability can lead to application and business failure. By considering these requirements sufficiently early, both software engineers and information systems engineers can and should use appropriate techniques to deliver performance, just as computer hardware engineers and telecommunications engineers have always done.

Using UML and Model Driven Architecture (MDA) approaches, system designers start modelling their systems from the top layer down paying special attention to the most intensively used parts of the system, or so-called "critical paths". A component within a critical path is typically called thousands, millions, and even billions of times and its performance is absolutely critical to the overall system performance evaluation. However, the common belief, compounded by influential studies of EJB performance [ECZ02] is that the execution time of method code is within one or two percent of the overall execution time, including communication, transaction support, pooling, naming, security, etc.

To make the matter worse, most of the researchers in the area of EJB performance tend to use simple example applications provided by Sun with the J2EE SDK [Mic04b], which do not demonstrate any correlation between input parameters variation and application performance [GDE04, MM02].

This paper challenges this common belief by testing the COTS EJB components with various input data and tries to correlate data variation to performance of components.

## 2. EJB Components Reasoning

For the purpose of the study it was important to obtain real, industry-grade components, not to use homegrown or sample ones. ComponentSource.com [Com04] was identified as an industry leader in component offerings on the Internet. ComponentSource offers more than 10,000 components. It offered 31 EJB components by 15 publishers on the day of testing, in the following areas:

- Addressing, Postcode and ZipCode (1)
- Code Creation/Editing (3)
- Credit Card Authorisation (1)
- Data Storage (1)
- Database Management (1)
- eCommerce (4)
- Email (1)
- Financial (6)
- Internet Communication (1)
- Maths and Stats (8)
- Security and Administration (1)
- Source Code Generators (1)
- Web Site (1)
- XML (1)

17 components out of 31 were available for download either with limited functionality or as a demo. 11 components out of 17 were used in this study. The functionality of other components was either too restrictive or just not suitable for the purpose of the experiment.

## 3. Experimental Environment

### 3.1. Software Environment

We use a WebSphere Application Developer 5.0.1 [IBM04] environment for the tests. The monitoring and measurement of execution times within the EJB Container were done with JProbe Profiler v 5.0. All experiments were run on IBM's native WebSphere JVM v 1.3.1 for Windows x86 with the following options:

- -Xms256m: sets the initial Java heap size to 256 MB to prevent performance loss in increasing heap size at warm-up period;

- `-Xbootclasspath`, `-Xrunjprobeagent`: parameters for including JProbe profiler suit into the testing environment. This option was turned off when JProbe was not required.

To avoid unnecessary memory consumption and possible measurement intervention, only one component was deployed on the server at a time. The application server was restarted for each bean test.

The test machine runs Windows 2000 Server.

### 3.2. Hardware Platform

The EJB server runs on P-IV 1.6 Ghz with 1GB SDRAM and 40GB IDE HDD. Our tests do not make use of HDD, so its performance is not an issue. To minimise unpredictable behaviour of Virtual Memory Manager (VMM) with swap storms, swap size was set to zero to make sure all code runs in RAM. For the duration of the tests the CPU was always the bottleneck.

### 3.3. Measurement Methodology

The measurements are performed with a remote client that calls EJBs directly. Each bean is tested separately. The client runs on the same machine to exclude the possibility of network delays impact on measurements. Java Reflection mechanism was used to obtain to construct an appropriate invocation parameter sets for each bean. Beans were tested repeatedly with different input data and the results were logged after the initial "warm-up" phase of 10 subsequent calls.

No attempt was made to understand the inside logic of components tested for purity of experiments. Components treated solely as "blackboxes" with known interfaces. Subsequently, no parameter tweaking was conducted to adjust the particular test to a certain bean to increase the amount of correct responses from it. The only exception to this rule was the warm-up phase duration, which is chosen by observing the method response times. The "cut-off" point was chosen when a response time reaches a steady state and by observation of the length of time necessary to obtain reproducible results. For each set of parameters, each bean is called 10 times. Fewer calls would cause undesirable loss of result precision, since the clock measurements provided by the Java environment is not very accurate. More calls would result in longer testing sessions, especially for beans with a lot of input parameters. For instance, tests for beans with methods that have 8 or more parameters are known to last for 12 hours or longer. The disadvantage of that method is that every test result is the average of 10 calls to a bean.

All the result data are analysed post-mortem to minimise the influence on the system during the experiments. Each input parameter is changed 4 times during the testing (except for Boolean, which has just two states). Java primitives, such as doubles, longs, ints are assigned a value of  $10^n$ . Arrays are constructed using  $n$ -elements in each dimension, populated with random numbers within a range of [0...1].

A separate set of experiments was performed using the JProbe profiler to monitor the stability of the environment and EJB container logic. In particular, the average time to initialise a bean was determined, as well as the average overhead for each bean invocation

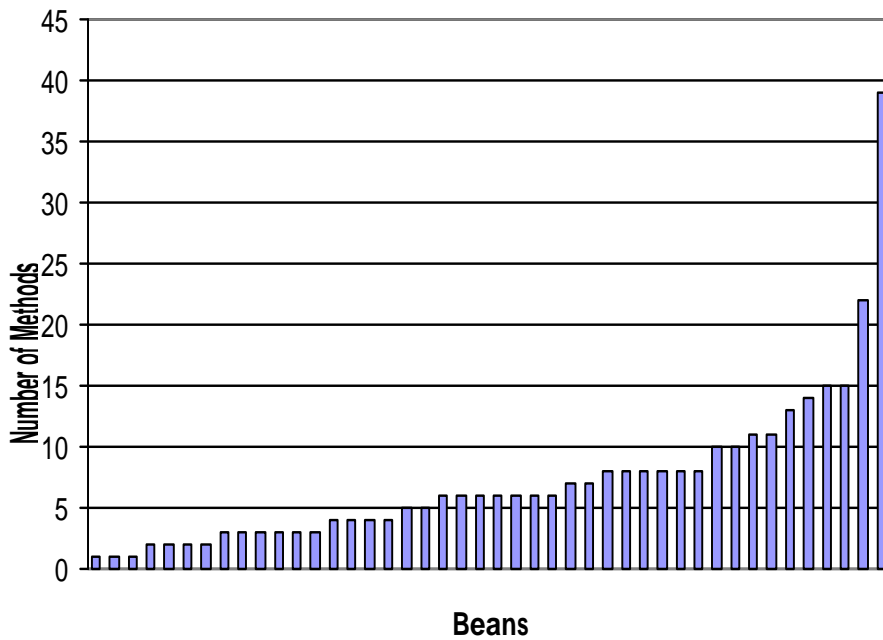


Figure 1: Number of Methods in Beans (sorted)

by a container. That is done by analysing the JProbe snapshots and statistically aggregating the results received. In particular, this set of tests tries to determine the authenticity of the common belief that an EJB method execution time is on average less than 1 – 2% of a total bean execution time.

## 4. Experimental Results

### 4.1. Test Set Analysis

Since the purpose of this study is to determine the influence of method input values on component performance, only stateless session beans were evaluated to minimise the influence of other aspects of EJB technology. The EJB components tested in this study represent a variety of statistical and financial commercial-off-the-shelf (COTS) components available at component marketplaces.

11 components were tested, which have 4 beans on average. There are 44 beans in these 11 components, which provide, in turn, 312 methods. The number of methods within beans also varies greatly, from 1 method per bean to 39 methods, and 7 methods per bean on average (See Figure 1 for more information)

From the 312 methods tested in these beans, 196 did not throw any exception during tests. 115 methods (or 37% of the total) throw an exception because of incorrect call

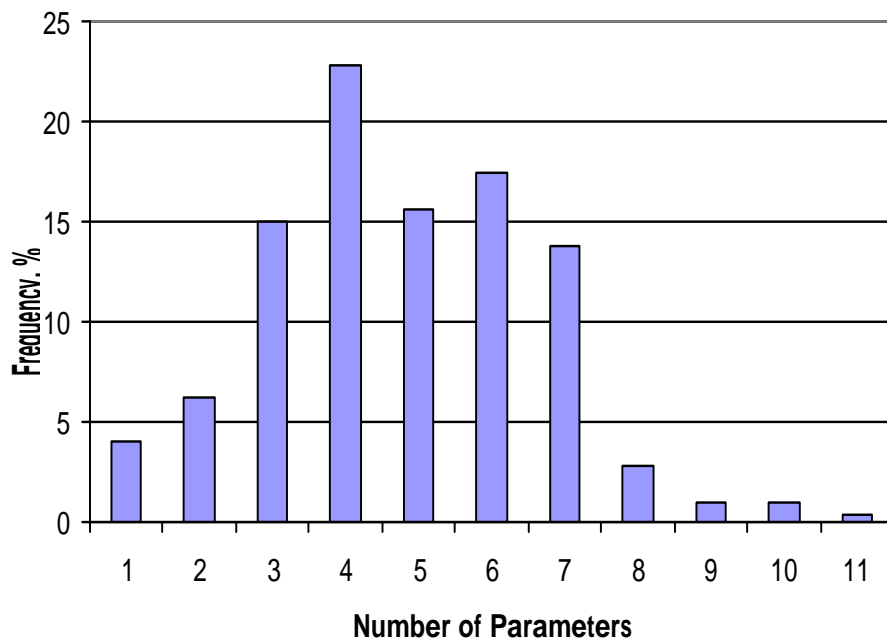


Figure 2: Distribution of Input Parameters in Beans

order (some methods should be called before others), values out of range, or illegal access. Components which throw exceptions because of trial or demo limitations are not considered during result interpretation.

The majority of methods have just three arguments (See Figure 2). However, some methods require up to 10 input parameters.

#### 4.2. Container Time vs Method Time

A separate set of experiments was conducted to determine the grounds for the popular belief that the actual method execution time on average is less than 1 – 2% of the overall method invocation time. JProbe Profiler was used to obtain the container time and method times. It was determined that it takes on average 100 ms for a container to initialise the bean. This included the container's security checks, naming service, communication, etc. Containers also add an average overhead of 1.2 ms for each subsequent method invocation of various services. The communication delay between client and EJB container takes another 1.3 ms. Method execution time was derived by subtracting those values from overall method execution times, which included both container and method parts (see Figure 3). The average method execution time is 0.6 ms, with standard deviation of 2.9. Though this type of calculation might be somewhat inaccurate due to the small number

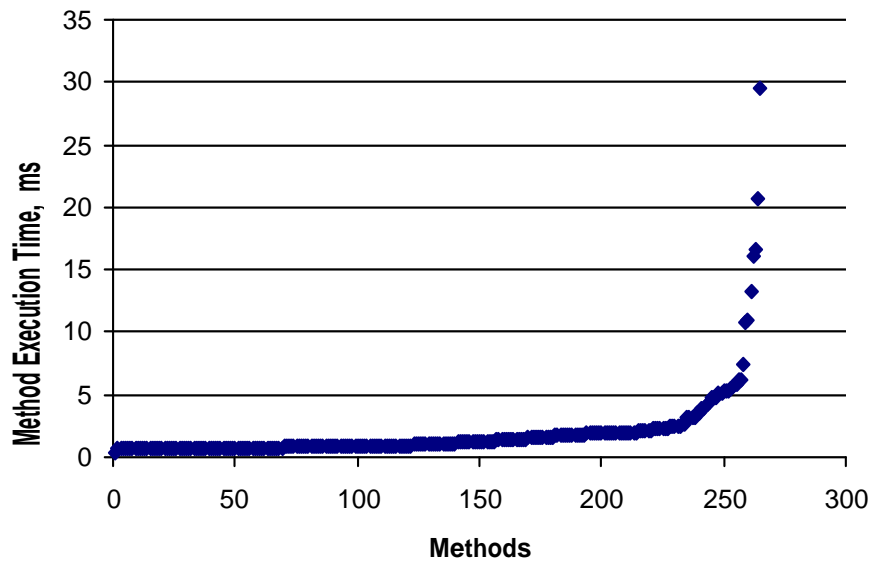


Figure 3: Average Method Execution Time

of methods tested, the result suggests that on average method execution time is more significant than just 1-2% of overall method invocation time.

### 4.3. Component Performance Variation

First, the influence of input parameters on component performance was calculated. For this purpose we used range of variation, arithmetic mean, and standard deviation. Range of variation table suggests that for some methods performance change is quite significant (See Figure 4) - up to 250 ms. Though it might not seem as a large value, these response time "jumps" should be considered by a system designer, especially when a component is used within the main usage scenarios. If thousands of users call a component and it returns an answer within 240 ms instead of originally envisaged 10 ms, that could become a performance problem.

Next, we determined how significant are these numbers and how intensive the variation is (See Figure 5).

It can be noted that the vast majority of methods execute within a 3 ms period, which makes the variation of around 200 ms very significant. Then the intensity of variation is determined by calculating standard deviation (See Figure 6).

Again, for the vast majority of methods the standard deviation never goes beyond 5 ms regardless of the input data passed to it. Still, a number of methods with a significant

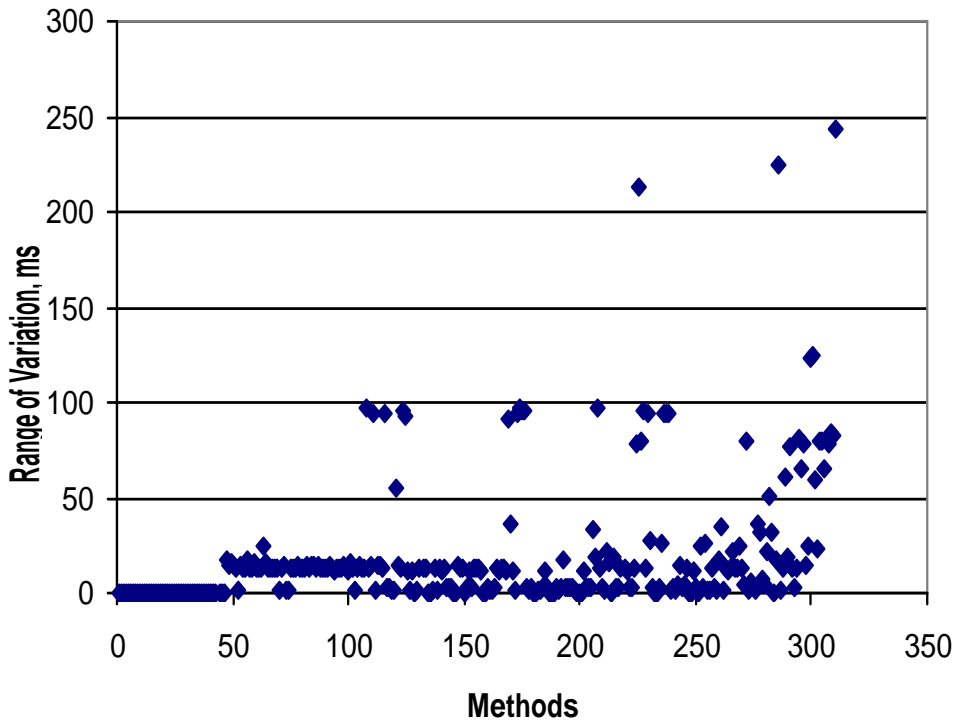


Figure 4: Range of Variation

range of variation is present in the graph. The coefficient of variation was examined to determine the results (See Figure 7). The presence of values above 1.0 suggests performance instability for a method.

Three most probable reasons for significant performance fluctuations:

- Component design - certain parameters cause "spikes" in the system's resource consumption by the component;
- Undesirable system activity - e.g. garbage collection, etc;
- Measurement errors.

The tests were repeated on another machine and similar results were obtained.

#### 4.4. Correlation between Input Data and Component Performance

We also calculated the correlation between each pair of input parameters and the resulting component performance (See Figure 8). The average correlation for components tested is 0.08 with standard deviation of 0.14, which suggests that the vast majority of components do not seem to have any correlation between their input data and performance. According to our tests, only 0.7% of methods have strong ( $corr \geq 0.9$ ) correlation between input data and performance. Another 0.6% have a significant ( $0.7 \leq corr < 0.9$ ) correlation.

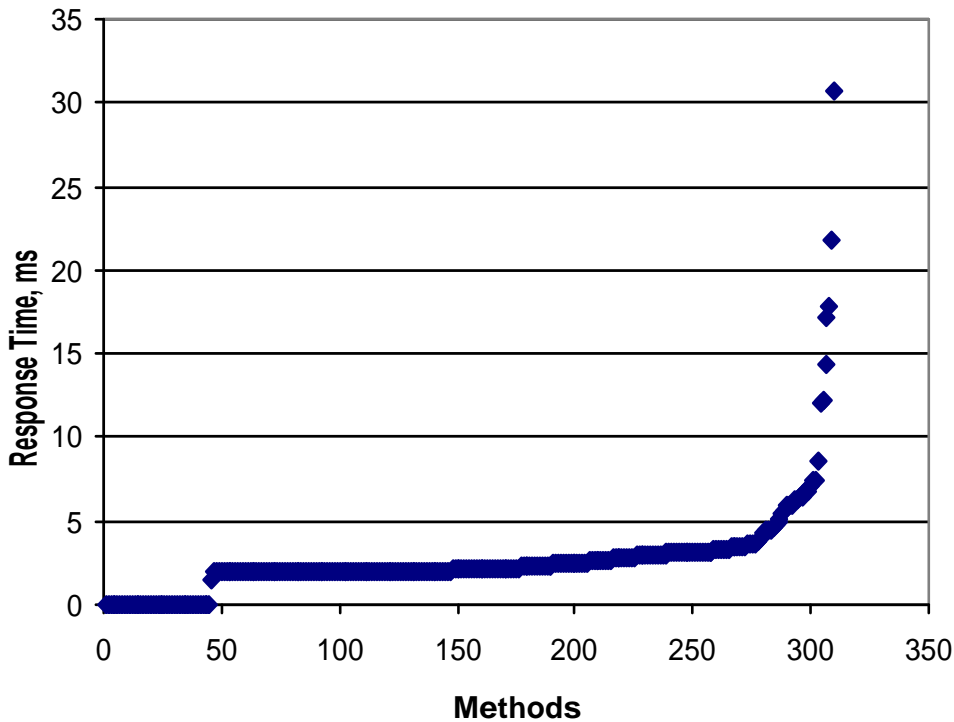


Figure 5: Mean Response Time

1.8% of methods also have weak correlation ( $0.5 \leq corr < 0.7$ ), while the rest of the methods (96.9%) do not have any correlation of their input data to performance.

Then the correlation was calculated separately for a subset of 10% of all methods, which had the largest range of variation, mean response time and standard deviation. Only 63% of these methods do not seem to have any correlation to component performance, while 31% of methods have strong correlation with one of its parameters, and 3% have more than one of its parameters strongly correlated to component performance. This suggests that the resource-consuming methods tend to be more sensitive to their input parameters.

## 5. Related Work

The performance of J2EE application server is a very hot topic in the research and e-business communities. The former is mainly concerned with the performance prediction of component-based applications [GDE04, MM02, BM04], also providing some guidelines for EJB server comparison [DSRG00]. The latter appears to over rely on standard and not always suitable TPC tests [tra04] for benchmarking. In an attempt to standardise the evaluation of EJB servers, Sun has released the ECperf [Mic04a] specification.

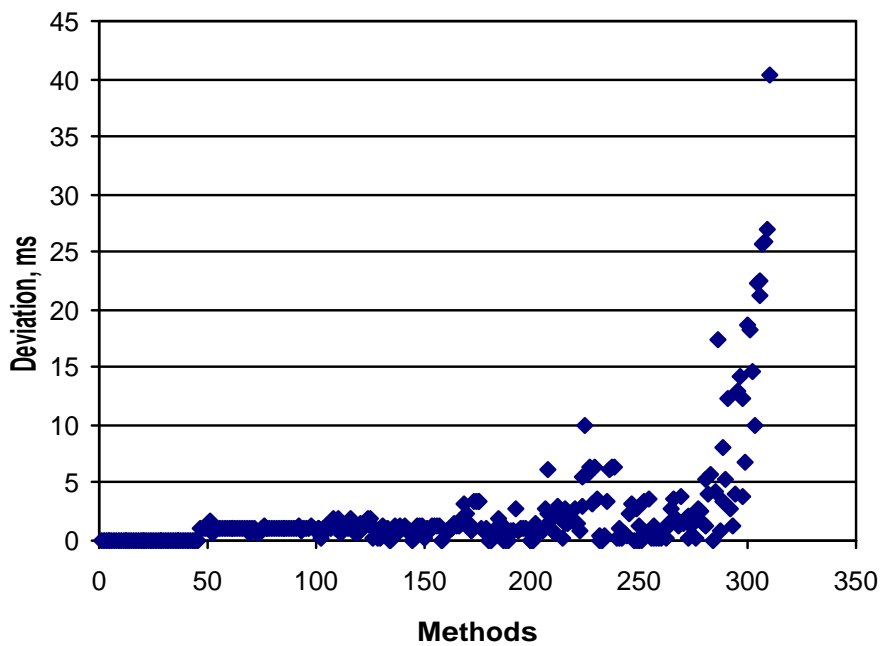


Figure 6: Standard Deviation

ECperf is a benchmark for evaluation of performance and scalability of a particular J2EE application server with a single application.

The RuBIS framework [ECZ02] was the first to combine the results of several EJB implementation of the same e-commerce application, using different application implementation methods, container designs and communication layers. While enabling testing of various performance aspects of EJB systems, it overlooks the importance of the input data.

Hamlet [Ham01] looks into data passed between components, arguing different data could change the execution path and thus the response time considerably. Our work proves that performance can be different even without complex component interactions, by changing the data input values given to a component.

## 6. Conclusion and Future Work

Various industry-grade EJBs were examined, using various input parameters to watch their influence on components' performance. Stateless session beans were used to focus on the relation of input data and component performance, minimising the possible effects any other feature provided by EJB containers could have on the measurements.

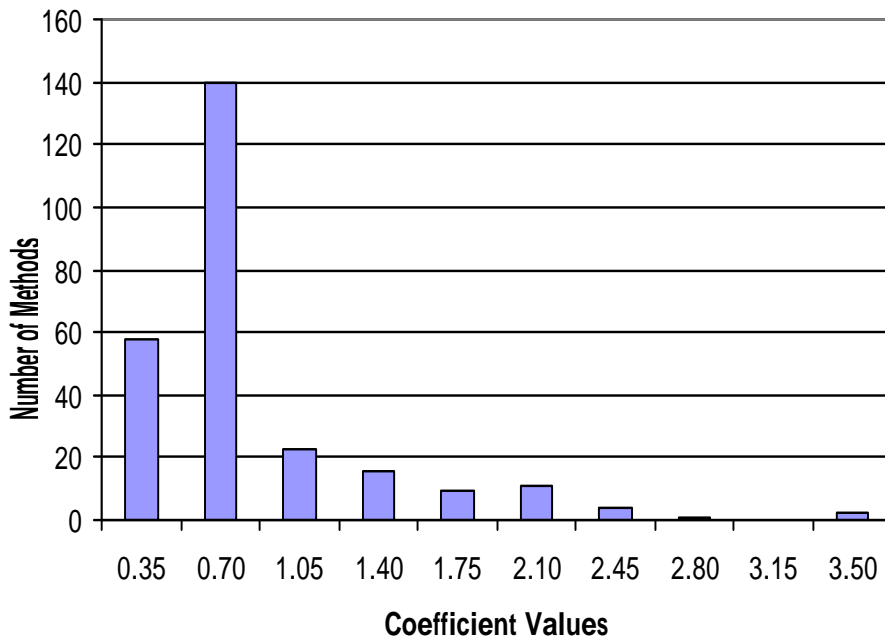


Figure 7: Coefficient of Variation

It was shown that the number of beans influenced by the input data, is small. However, an application is likely to get a significant performance hit if one or more input data sensitive components get in the "critical path" of the designed application. Also these beans tend to be more resource-consuming than beans insensitive to the input data.

Future work might include more intelligent approach to testing, including automatic detection of testing ranges, dynamic adaptation of test environment to various conditions, such as finding performance peaks varying component input parameters, increasing or decreasing the precision depending on the response times, automatic detection of proper call sequences to decrease number of invalid requests.

## 7. Acknowledgements

The support of the Informatics Research Initiative of Enterprise Ireland is gratefully acknowledged. The authors would also like to thank Mircea Trofin and Daniela Mania for their help with configuration of the test environment.

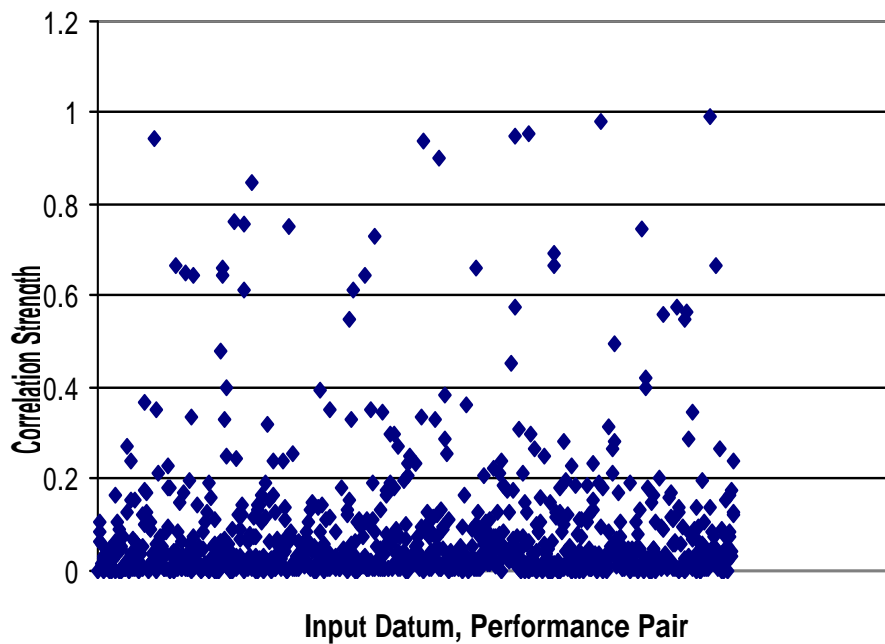


Figure 8: Correlation Between Input Values and Component Performance

## References

- [BM04] Antonia Bertolino and Raffaella Mirandola. Software performance engineering of component-based systems. In *Proceedings of the Fourth International Workshop on Software and Performance, WOSP 2004*, Redwood Shores, California, USA, January 14-16, 2004. ACM.
- [Com04] ComponentSource.com, 2004.
- [DSRG00] Charles University Distributed Systems Research Group. Ejb comparison project, 2000. <http://nenya.ms.mff.cuni.cz>.
- [ECZ02] Julie Marguerite Emmanuel Cecchet and Willy Zwaenepoel. Performance and scalability of ejb applications. In *Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 246–261, Seattle, Washington, USA, November 4-8, 2002.
- [GDE04] Andrea Polini Giovanni Denaro and Wolfgang Emmerich. Early performance testing of distributed software applications. In *Proceedings of the Fourth International Workshop on Software and Performance, WOSP 2004*, pages 94–103, Redwood Shores, California, USA, January 14-16, 2004. ACM.
- [Ham01] Dick Hamlet. Component synthesis theory: The problem of scale. In *Proceedings of 4th ICSE Workshop on Component-Based Software Engineering Component Certification and System Prediction*, Toronto, Canada, May 14-15 2001. IEEE Computer Society.
- [IBM04] IBM. Websphere application developer, 2004. <http://www-306.ibm.com/software/awdtools/studioappdev/>.
- [Mic04a] Sun Microsystems. Ecperv specification 1.1, 2004. <http://java.sun.com/j2ee/ecperf/>.
- [Mic04b] Sun Microsystems. Java 2 platform enterprise edition (j2ee), 2004. <http://java.sun.com/j2ee/>.

- [MM02] Adrian Mos and John Murpy. Performance management in component-oriented systems using a model driven architecture approach. In *Proceedings of the 6th International Enterprise Distributed Object Computing Conference*, pages 227–237, Lausanne, Switzerland, September 17-20 2002. IEEE Computer Society.
- [tra04] Transaction processing, 2004. <http://www.tpc.org/>.

1

**Authors addresses:****Alexandre Oufimtsev**

Performance Engineering Lab, Department of Computer Science  
University College Dublin, Belfield, Dublin 4, Ireland  
Alexandre.Oufimtsev@ucd.ie

**Liam Murphy**

Performance Engineering Lab, Department of Computer Science  
University College Dublin, Belfield, Dublin 4, Ireland  
Liam.Murphy@ucd.ie