28TH ANNUAL
PACIFIC NW
SOFTWARE
QUALITY
CONFERENCE

OCTOBER 18TH – 19TH, 2010

ACHIEVING QUALITY IN A COMPLEX ENVIRONMENT

Conference Paper Excerpt from the

CONFERENCE PROCEEDINGS

Issues in Verifying Reliability and Integrity of Data Intensive Web Services

Anand Chakravarty
The Microsoft Corporation
Redmond, WA 98052

Abstract

Validating the behavior of web-services in general involves the verification of the constituent parts in isolation and in integration. There are many variables to consider within each component, and these have to be verified as the system consumes growing resources under actual usage once they are shipped. These challenges multiply when such systems also have to deal with large volumes of data. In addition to the impact large data make on system performance, there are also implications on code-coverage metrics, identifying bottlenecks, predicting failure points, system versioning, data maintenance, etc. There are further challenges involved in creating a reliable set of quality metrics and measuring them within the constraints of shipping under the agile development model. A web-service that depends on huge and growing volumes of data is considered here, with an emphasis on: measuring the overall system performance under expected traffic patterns, impact of abnormal data input and system behavior under different failure conditions. The lessons learnt here have helped ship a web-service that currently serves millions of users per day, with improved data quality and following the agile development and shipping model.

Author Biography

Anand Chakravarty has been a Software Engineer in Testing at Microsoft for close to 10 years, most of it working on web-sites and web-services based products. Currently, he is on an incubation team under Microsoft Research that ships products based on Machine Translation technologies. On that team, he is primarily responsible for the stability of web-services supporting Microsoft's diverse applications that translate user input between different human languages.. This small team has been successful in shipping features and services that have seen a manifold increase in user traffic and significant improvements in linguistic quality over the past few years.

1. Introduction

Verifying the quality of the more commonly found web-services involves verifying the components of the web-service(s) individually, and then verifying the interaction of these components end-to-end. Verification of the individual components may take the form of traditional unit-tests and functional level tests. These types of tests verify the components in a more static form. While these are useful, especially if run during the development phase [1], the utility of many bugs found with such tests is significantly surpassed by verifications performed when these

components are actively running, and even more by tests run when these components are interacting with their downstream dependencies and their upstream consumers.

To such systems, the addition of components that are data-intensive introduces a multitude of diverse areas which require more rigorous verifications. These range, chronologically, from building and deploying the data to upgrading and re-versioning them. At runtime, there are the obvious performance implications of processing huge volumes of data. In addition, it is critical to ensure that the services remain reliable in terms of the data they process, resilient to data related errors and available with a high level of certainty. In addition to the testing challenge of generating adequate test data to reliably verify the system [2], the verifications performed for such a system should measure the impact of data being unavailable for any period of time, the effects of redundancy mechanisms that help keep the data synchronized across failover instances, up-to-date and online, and help in designing maintenance jobs, their impact on the execution of primary functions of the web-service and their scheduling and update notifications.

Existing research on the topic of testing web-services has involved extending the approach of testing more traditional systems to the sphere of web-services [3], applying test-driven development to web-services [1], generating pair-wise test cases for web-services [4], etc. Bochicchio *et al* [5] have presented a modeling exercise for information intensive systems.

This paper considers a case-study of a distributed set of web-services that consume huge volumes of data at runtime. These web-services together host a machine-learning system which drives Machine Translation (MT) products shipped by Microsoft Research's incubation team. These products help users translate text between different human languages, and support multiple user scenarios. The web-services use a machine-learning system that has arisen out of decades-long research in the field of Machine Translation. The entire system processes huge volumes of data, both for training purposes and during actual execution and processing of translation requests. In this paper, I present the results of verifications of such web-services in terms of their behavior at runtime when receiving requests that reasonably simulate actual traffic. While this simulation is more reliably accomplished for systems that are already live [6], there are challenges when such data are not available. In addition to estimating numbers of expected users, the test design should also account for the varying nature of user input. It is important that the load-tests are a good simulation of real user traffic, to help accurately identify memory usage patterns and stress the components realistically. While there are models that exist to create such simulations, such as the form-oriented and stochastic-form models [7], we have found that a distributed set of test requests, combined with individual test case input that is significantly larger than the average real input, leads to meaningful test scenarios that help identify key code-defects. The advantage of such an approach is that it reliably creates sufficient stress on the system to simulate conditions that are both higher than actual expected live traffic and closer to the system's overall failure point. Test results under such conditions thus help to increase confidence in the reliability of the system when it is shipped. In analyzing test results with this data, special emphasis is given to measuring system reliability, impact of performance enhancement algorithms, resilience of the system to abnormal input and recovery from failure of individual components for varying periods of time.

2. High-level architecture

The system under consideration is a distributed system of web-services consuming large volumes of data to translate text between human languages. The architecture of the system was designed to meet well-defined goals of performance, reliability and linguistic quality. Components that are data-intensive should be hosted separately and there should be separate components depending on the resources that are being consumed. Below is a high-level overview of the final design:

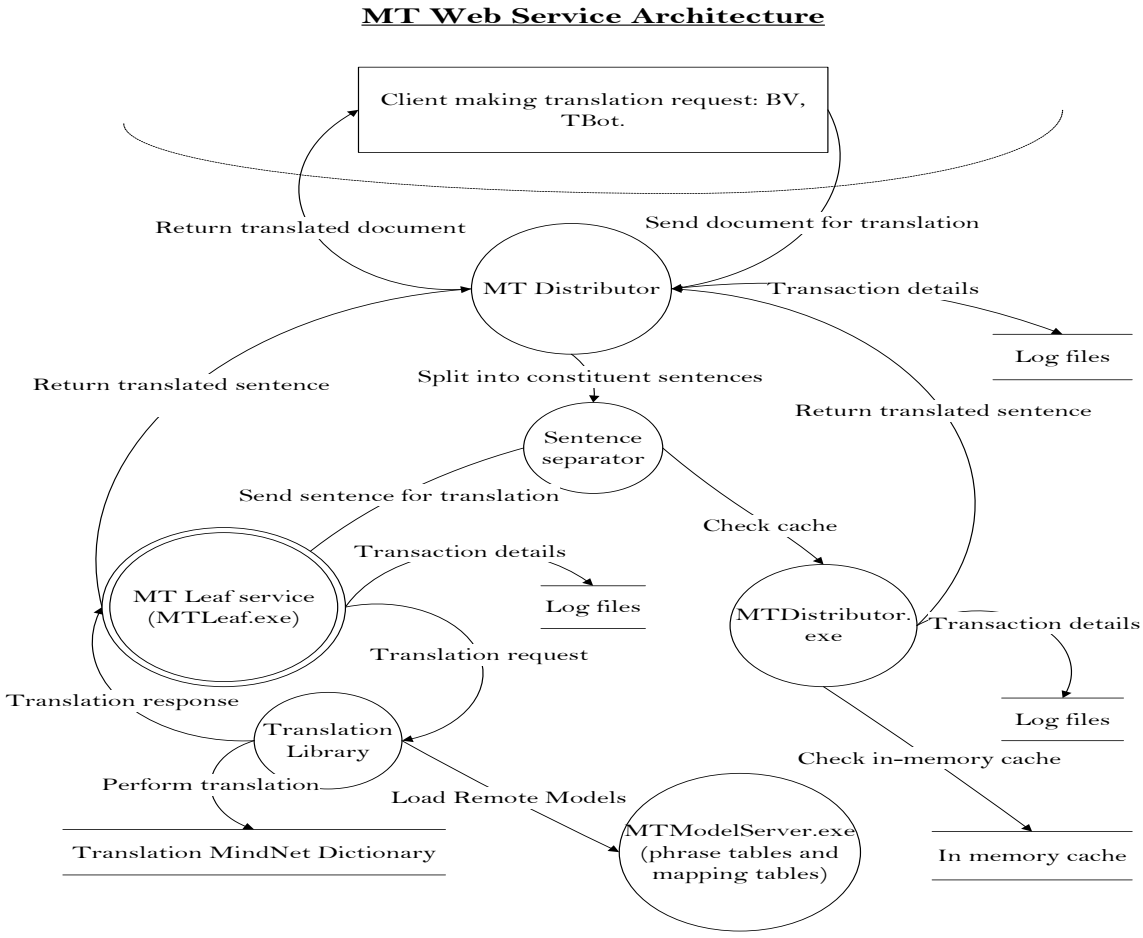


Figure 1. Web service architecture.

The data intensive components in this system are the set of web-services hosted by the leaf and the model-server machine types. These machines host translation models for each language, and are of the order of multiple gigabytes in size. In the next sections, we consider different test scenarios for these components, starting with the effects of loading multiple instances of a data component.

3. Measuring impact of different memory optimization algorithms

For applications that use large amounts of data at runtime, it is sometimes useful to load frequently used data into a memory-mapped file. These get the advantages of quick access combined with the utilization of operating system memory handling techniques. Also, when such systems are run on multi-processor systems, it is beneficial to load up multiple instances of the same data into memory and attempt to maximize the potential speed enhancements that accrue. There is however a threshold beyond which multiple instances yield little value and in fact begin to adversely impact system performance. To identify this for the Microsoft Translator-MT system, we ran the leaf processes with different settings. In one configuration, the leaf processes would load 2 instances of a required file into memory, then 3 and 4. Load-tests were run for each of these configurations. For the configuration where 2 instances were loaded into memory, the average throughput was about 5,000 words per second. When the same load-test was then run for a configuration where 4 instances of the model files were mapped into memory, the throughput went up, as expected, to 8,000 words per second.

However, the increase in throughput was offset by an increase in the number of errors that the system returned at runtime. This is a consequence of increased memory usage when more instances are mapped to memory, which results in higher page faults and consequently more errors. Below is a comparison of test results for one of the language pairs that was stressed:

Table 1. Comparison of failure rates

	With 2 instances of memory mapped file	With 4 instances of memory mapped file
Throughput, in words per second	5,000	8,000
Percentage of requests failing	1.6%	13.5%

Running load-tests with different configurations, and measuring the performance and success rate of the test cases, showed that depending on available resources and the size of data files, memory mapping soon begins to drastically increase the rate of errors beyond a certain point. In this specific case, which dealt with a service using managed .NET components, part of the overhead was in garbage collection.

The load-tests found that when 4 instances of each translation model is loaded into memory, the time spent in garbage collection by the leaf processes was between 10 to 40%, depending on the size of models. Leaf process that had larger models to load spent close to 40% of time in garbage collection, while leaf processes that had smaller translation models spent 10% of their time in garbage collection. When applications spend more than 10% of time in garbage collection, there are fewer resources available for doing real application specific work. The next section gives more details of the impact of garbage collection on system behavior.

4. Garbage collection for services

Garbage collection is a memory management method used in certain application runtime environments such as the Java VM and the .NET runtime environment [8]. While there are many benefits of this method, it is vital to measure its impact on overall system performance so as to use resources in an optimal manner. Applications that are data intensive are especially vulnerable to memory pressures that might be worsened by garbage collection (GC) operations.

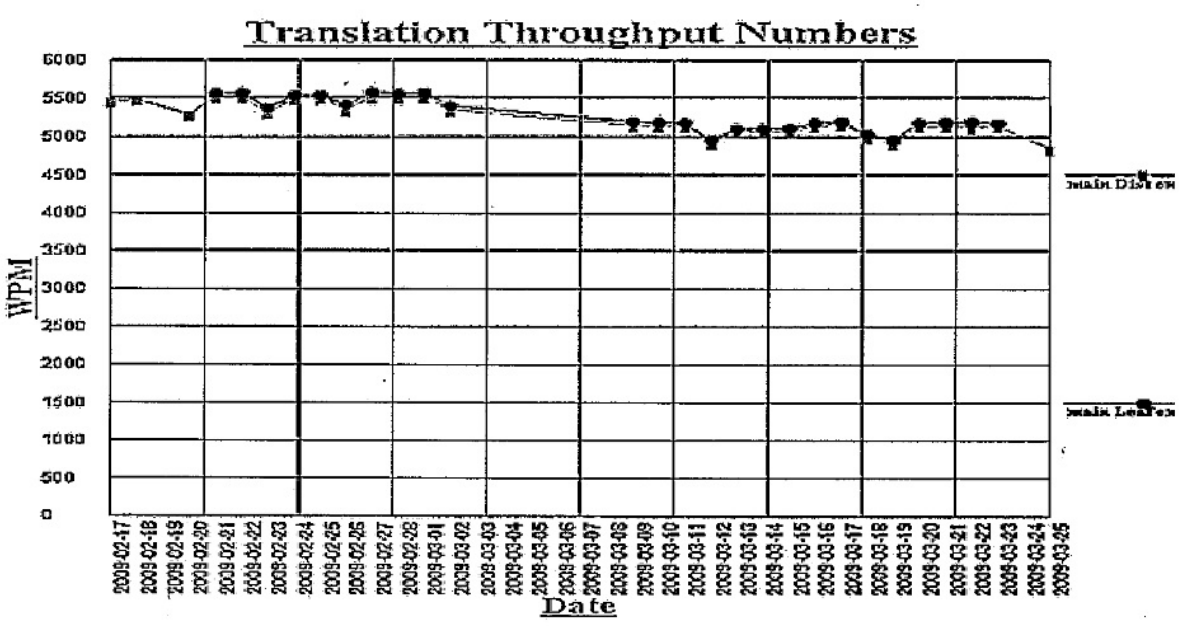
In the .NET runtime environment, there are different flavors of garbage collection, named as: workstation garbage-collection, server GC and concurrent GC. It is important to choose the right flavor to optimally use available resources while sustaining real usage scenarios. For the MT web-services, in order to figure out the right version of garbage-collection to use, load-tests were run with the constituent applications using different flavors of garbage-collection. The results for such a load-test run with the web-services using Workstation-GC showed that the web-services spent close to 20% of their time in garbage collection. The same system when run with Server-GC enabled saw less overhead, only about 4% on average.

The percentage time in garbage collection reached close to 20% during this round of tests. The same system when run with Server-GC enabled saw less overhead, only about 4% on average. There was also a difference in the pattern over time of garbage collection. The overhead was reduced when Server-GC was enabled, compared to the more commonly used Workstation-GC. Reducing this overhead is especially valuable for data-intensive systems which use large amounts of memory or disk-space. Resources thus freed up are diverted to more meaningful data-processing tasks and consequently yield benefits in overall system performance.

5. Overhead of individual components

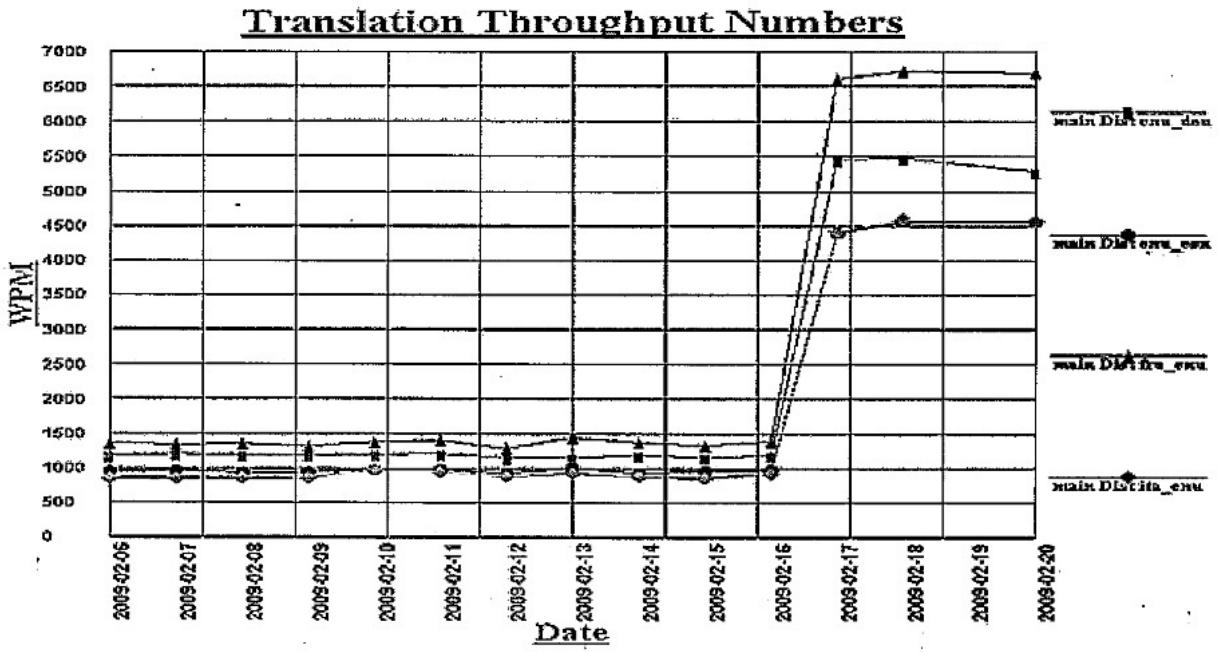
Identifying the bottlenecks in a distributed system is vital to ensuring efficient use of resources [9]. System bottlenecks are most commonly found in components performing I/O heavy tasks [10]. The architecture of the MT Translation web-service involves communications between multiple machines to completely process a translation request. This requires each component to be lean and extremely responsive to other components. Each component has its own communication ports, its own queues, different rates of execution, and different ways of failing. Our stress tests helped to measure these overheads, and as the system was made more efficient, they helped to prove that in all this interaction, the overhead in going across multiple machines was minimal.

For these tests, we start with a tightly defined set of input test cases, and hit the individual components separately. For example, we send translation requests to the distributor machines, and then the same set of translation requests directly to a leaf. The leaf is one step closer to the model-server, so it is expected that performance is faster when requests are sent to a leaf, as opposed to when they are sent to a distributor. As seen from Figure 6, the eventual implementation of these 2 components was good enough that there was almost no overhead involved in going through another machine type, other than network latencies. The darker line in Figure 6 is the throughput when test requests are going directly to a leaf machine, and the lighter line is the throughput when the same requests go through the distributor, which is an extra layer compared to the first scenario:



The tests that generated the above graph were run regularly, across rolling builds, thus enabling easy identification of changes that affect performance. Below is an instance where a particular code-change resulted in a 4-fold performance boost across multiple language pairs:

Figure 7. Quantifying impact of code-changes



Designing a test-case system that allows the automated monitoring of individual components and reports results that easily highlight the impact of system changes is essential to evaluating end-to-end performance of the distributed components of web-services.

6. Verifying Resilience to failure

It is imperative that web-services have a high rate of availability. While hardware redundancy is a good first step to achieving high availability, it is not price-optimal. Also, even with hardware redundancy, it is vital for all components of a web-service to understand the availability of the components they depend on, recover from the failure of these components by quickly identifying fallback paths upon failure, and report failures in an actionable manner when bottle-necked.

In addition to overall system resilience, there are also aspects of the behavior of individual components in their interaction with other parts of the web-service that affect resilience. For the MSR-MT system, with its distributed web-services, with different services having different resource constraints, tests were run with the individual components in different states of failure to identify such aspects of system behavior.

Assuming there are multiple machines supporting a particular component of a web-service, consumers of that component should have a clear idea of their availability. For example, in the MSR-MT system, if a model-server is assumed to be running on 4 different machines, and one of them experiences a failure, the leaf and distributor processes that depend on it should remain unaffected, apart from performance issues owing to reduced resources.

Verifying this scenario for the MSR-MT web-services resulted in the finding of bugs relating to process initialization, recovery and cleanup. It is important that these issues are validating by executing failure scenarios, to ensure the web-service remains responsive to user despite transient failures.

7. Conclusions

The challenges of testing web-services are compounded when there are data-intensive aspects of these services. In order to properly verify the impact of such data, it is important to define metrics that quantify the resource-usage of the overall system. Properly simulating actual user traffic is vital, as is verifying the system under conditions of failure. Validating design decisions by providing test results that show the performance of the system under different configurations, especially when these configurations impact resources such as processor or memory usage, helps to implement a system that optimally uses those resources. As shown in our investigations of garbage-collection, it is critical to verify the system considering the resource-management feature of the platform on which the web-services are built. The identification of system bottlenecks by focused testing of individual components results in a system that is combines the benefits of modular design of components with a high-performing, efficient overall system. Such an approach to testing has helped the MSR-Machine Translation team ship a system of web-services that have supported a rapidly growing set of users and features, with increasing performance and quality.

8. References

[1] Nuno Laranjeiro, Marco Vieira, "Extending Test-Driven Development for Robust Web Services," Dependability, International Conference on, pp. 122-127, 2009 *Second International Conference on Dependability, 2009*

[2] Ying Jiang, Ying-Na Li, Shan-Shan Hou, Lu Zhang, "Test-Data Generation for Web Services Based on Contract Mutation," ssiri, pp.281-286, 2009 *Third IEEE International Conference on Secure Software Integration and Reliability Improvement, 2009*

[3] Samer Hanna, Malcolm Munro, "An Approach for Specification-based Test Case Generation for Web Services," aiccsa, pp.16-23, 2007 *IEEE/ACS International Conference on Computer Systems and Applications*, 2007[4] Siripol Noikajana, Taratip Suwannasart, "An Improved Test Case Generation Method for Web Service Testing from WSDL-S and OCL with Pair-Wise Testing Technique," compsac, vol. 1, pp.115-123, 2009 *33rd Annual IEEE International Computer Software and Applications Conference, 2009*

[5] Mario Bochicchio, Antonella Longo, "Conceptual Modeling of Data Intensive and Information Intensive Web Applications," mmm, pp.292, *10th International Multimedia Modelling Conference, 2004*

[6] Ka-I Pun, Kin-Chan Pau, Yain-Whar Si, "Modeling Support for Simulating Traffic Intensive Web Applications," wi-iat, vol. 3, pp.512-516, 2008 IEEE/WIC/ACM *International Conference on Web Intelligence and Intelligent Agent Technology, 2008*

[7] Lutteroth, C.; Weber, G.; , "Modeling a Realistic Workload for Performance Testing," *Enterprise Distributed Object Computing Conference, 2008. EDOC '08. 12th International IEEE* , vol., no., pp.149-158, 15-19 Sept. 2008 doi: 10.1109/EDOC.2008.40

[8] Tomas Kalibera, Filip Pizlo, Antony L. Hosking, Jan Vitek, "Scheduling Hard Real-Time Garbage Collection," rtss, pp.81-92, 2009 *30th IEEE Real-Time Systems Symposium, 2009*

[9] Petriu, D.; Shousha, C.; Jalnapurkar, A.; , "Architecture-based performance analysis applied to a telecommunication system," *Software Engineering, IEEE Transactions on* , vol.26, no.11, pp. 1049- 1065, Nov 2000

[10] Kallahalla, M.; Varman, P.J.; , "PC-OPT: optimal offline prefetching and caching for parallel I/O systems," *Computers, IEEE Transactions on* , vol.51, no.11, pp. 1333- 1344, Nov 2002