2010 Seventh International Conference on Information Technology

Stress Testing an AI Based Web Service: A Case Study

Anand Chakravarty
The Microsoft Corporation
Redmond, WA 98052

Abstract

The stress testing of AI-based systems differs from the approach taken for more traditional Web services, both in terms of the design of test cases and the metrics used to measure quality. The expected variability in responses of an AI-based system to the same request adds a level of complexity to stress testing, when compared to more standard systems where the system response is deterministic and any deviations may easily be characterized as product defects. Generating test cases for AI-based systems requires balancing breadth of test cases with depth of response quality: most AI-systems may not return a perfect answer. An example of a machine learning translation system is considered, and the approach used for stress testing it is presented, alongside comparisons with a more traditional approach. The challenges of shipping such a system to support a growing set of features and language pairs necessitate a mature prioritization of test cases. This approach has been successful in shipping a Web service that currently serves millions of users per day.

Keywords: Internet applications, measurement techniques, software engineering, software performance testing, software testing.

1. Introduction

The stress testing of AI-based systems differs from the approach taken for more traditional Web services, in terms of the design of test cases and the metrics used to measure quality. Existing research on the performance and stress testing of software programs has tended to focus on measuring system performance under increasing loads [1], generating workloads that attempt to match real world behavior [2], and applying traditional object-oriented techniques to Web applications [3]. A review of prior research literature on stress testing software systems did not yield any significant and comprehensive approaches to stress testing AI-based Web services. Measuring the quality of such systems requires a departure from the traditional software testing approaches owing to the distinctive nature of AI-based system behavior. Validations of such systems have to account for changes over time in the response received for the same input, as the AI algorithm learns from experience and provides continually improving results. In addition, the behavior of the system under stress should result both in a good user experience and in a system that recovers from failures without adversely impacting the rules of the AI algorithm. This paper considers a machine learning system as an example of stress testing an AI-based Web service. The approach and practices outlined here have helped in shipping a Web service that has successfully served a growing number of users. Over 2 years, our AI-based translation system has grown from supporting 8 language pairs to more than 400, with a steadily increasing translation quality. Over the same time, the number of page-views for the different products hosted upon this service has grown dramatically as well.

The Machine Translation team in Microsoft Research ships products built upon an AI-based translation system that translates textual user input between human languages. This translation system is the result of decades long research in the field of natural language processing. Translation systems evolve for multiple language pairs and once the linguistic quality of a language pair meets a certain level (measured by an automated metric, BLEU [4]), we are ready to take that language pair to our production systems. In order to host the system in a data center, Web services have been developed which provide a standard interface upon which other front-end applications may be built.

At a high level then, there are three areas of testing for the entire system: linguistic quality, Web services and front-end clients. There are interesting technological challenges in each area, in addition to the verification of the interaction of these systems end-to-end, which is what the users eventually experience. Our paper focuses on the Web services component,

978-0-7695-3984-3/10 $26.00 © 2010 IEEE
DOI 10.1109/ITNG.2010.149

with an emphasis on testing the system under stress conditions that approach, and go beyond, the expected loads from live traffic.

The MSR-MT translation engine consists of systems that use vast amounts of training data to eventually create files that store language rules, such as semantics, casing, phrase mappings, etc. The resulting files are of the order of multiple GBs in size on average. Using these files at runtime, while also providing meaningful translations to users with reasonable response times, requires designing a system that is fast, balanced, robust, and resilient.

2. Architecture of the AI-Based Web Service

In order to meet design goals, the team decided on a distributed architecture designed to minimize CPU and memory pressures. This means distributing the language model files and other binaries across different machines while considering the expected traffic for a particular language pair, and the size of model files required to support translations for that language pair. It is also important for the architecture to have built-in redundancy and a certain awareness of the different types of errors that might be encountered. Some failures to translate may arise out of problematic input. In such cases, the system should be able to quarantine such sentences and remain available for other translation requests. A high level layout of the final Web services architecture is shown in Figure 1.

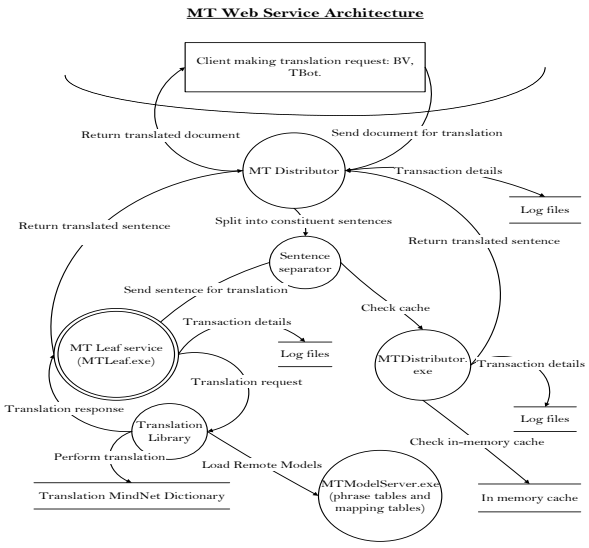

Figure 1. Web service architecture.

The Web service is made of three types of servers:

- A distributor machine: Receives chunks of data for translation, splits them into sentences and sends them on for actual translation. Returns response to front-end clients.
- A leaf machine: Performs translations one sentence at a time. Loads target language models, does factoid validation, uses remote mapping tables for true-casing and phrases.
- A model server machine: Specifically designed to only host memory-mapped files.

The AI-based components reside on the leaf and model server machines. In terms of the resources used on each machine type: the distributors are CPU intensive, the model servers are memory intensive, while the leaf machines both use a lot of memory and a fair amount of CPU resources.

3. Testing the Web Service

Testing the functionality and performance of this distributed system was done at both the individual component level and at the end-to-end integrated level. For functionality testing, it is important to be able to measure correctness of translations while accounting both for rolling versions and for differences in translations that appear while the system learns and improves. This property of the system precludes using the approach outlined in [5], wherein we emphasize uncovering "transient [and] convoluted failures that often plague distributed systems". For our purposes, we used a well defined set of input sentences for each language pair, and started with a baseline of expected results. As the system changes and improves over time, we programmatically apply linguistic quality measurements of the translation response against a baseline. We used measurements for calculating the Word-Edit-Distance, and the F-Score, which is a linguistics measurement that is a composite of precision and recall. As long as these values were within a predefined limit, the tests reported a success.

Stress testing this system was aimed towards measuring the following key behavioral characteristics:

1. Predictability of response: A translation request under stress should give a response that is close to or exactly the same as when the same request is sent when the system is under no other load. This is different from predictability under low or no loads, for which the approach defined in [6] might suffice.
2. Capacity to handle expected traffic: At loads that approach predicted usage, the system should be stable, continue to return predictable results, and not suffer performance degradation beyond a well-defined limit.

3. Mean Time Between Failures (MTBF): The definition given in [7] was used for our purposes. Under real world traffic, any long term degradation, such as those caused by memory leaks or fragmentation, should be at a pace that is slow enough that the system remains performant under steady load over prolonged durations.

The design of such a stress testing system involves defining test cases and then distributing requests for these test cases in a pattern that matches current traffic or an approximation of expected traffic. For the MSR-MT Web service, we created test case requests for each of the supported language pairs, and then created load-tests that apportioned these requests over time such that the request distribution matched the goal. For example, if the English to Spanish language pair is expected to account for 10% of the total traffic generated by all language pairs, the test system should be able to send English to Spanish test requests that, over a reasonable period of time, account for 10% of total requests, and so on for each language pair.

It is also important to pick the right set of requests to generate stress testing traffic. This varies according to the nature of processing performed by the system [8]. There are multiple approaches to accurately simulating this: using genetic algorithms to generate traffic that maximizes throughput [9] or understanding patterns of real-time communication between machines [10], etc. Considering the complexity added to our Web service by the AI-components, the approach we took emphasized end-to-end performance using realistic user data.

4. Effect of Search Depth on Translation Quality and System Performance

There is an additional factor to consider for AI-based systems. These systems perform extensive searches to come up with an acceptable answer. And the more data they search and analyze, the better is the ultimate answer they come up with. The depth of searching for better answers is usually configurable. Stress testing of such AI-systems must be able to measure the impact on overall performance of increasing the depth of these searches. The stress tests should provide data that answer the question: what is the maximum work the AI algorithm can do and still respond with an acceptable solution in a reasonable response time?

The MSR-MT translation system fits the expected pattern of being slower when it is given longer sentences to translate. At the same time, it could also become slower for shorter sentences if it is configured to search longer for a better quality of translation. How much should we turn this quality knob? Our stress tests were run to deliver a quantitative answer to this question. Sample results are shown in Table 1.

Table 1. Stress test results at varying AI-quality settings

CPU usage	Lower Quality Setting	Higher Quality Setting	%-age increase
Leaf	10%	30%	300
Model-Server	0.5%	6%	1200

Table 1 shows CPU usage on the MT model server and leaf machines for systems that have 2 different levels of the quality knob. These numbers are with the system being under similar levels of load. As mentioned before, the model servers are memory intensive. While the CPU usage is barely 1%, these are on machines that are using 90% or more of available machine memory. The extra work done by the model servers and the leaf machines with the 2nd setting resulted in a measurably improved translation quality. The stress tests were able to confirm that with a particular distribution of language pair models across available machines, the response time for the user and the rate of errors within the system remain within a predefined upper limit.

5. Distributing translation models for realistic user traffic

Another important aspect of testing and shipping the MT system was to determine the optimal distribution of the multiple translation model files. Let us consider just the model servers for this purpose. As mentioned earlier, these machines host memory mapped files, each of which is 0.5GB or more in size. If we assume that the machines hosting these files have 8GB of RAM, we could theoretically fit 16 language pair models on each machine. In practice, performance of a system starts to degrade drastically if it uses more than 90% of available RAM. Stress tests run against the MT system helped us identify how to distribute the language pairs across these machines so that none of them takes more than 90% of available RAM. We start by distributing the models based on their static disk file size, so that at most 80% of available RAM is used on each model-server. Then, in order to get meaningful numbers, the tests should use input that, over time, exercises almost all areas of the memory mapped files. This means using sentences with more variety in their content; length is of secondary consideration for this

purpose. For our stress tests, we used about 250,000 unique sentences per language pair to help activate more code paths inside the translation engine and its models. As these test sentences are being translated, the virtual bytes usage on the model server increases as the tests progress, as shown in Figure 2.

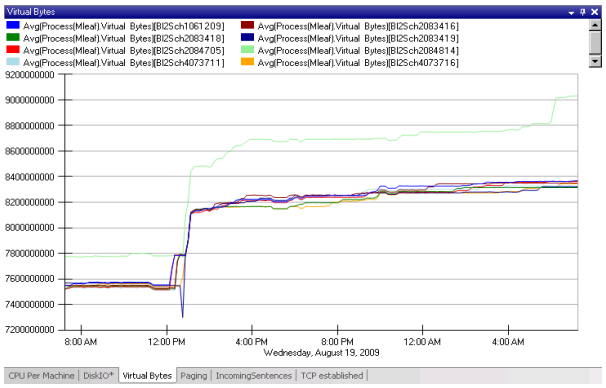

Figure 2. Virtual bytes usage.

Notice that virtual bytes usage increased from 7.5GB initially to almost 9GB on some machines. This indicates a problem, best captured by an increase in hard page faults, almost 1000/sec as shown in Figure 3.

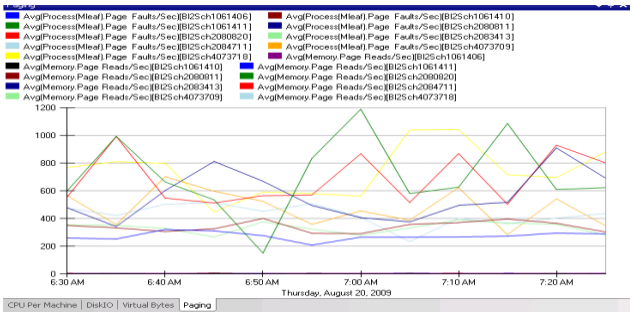

Figure 3. Page faults.

The optimal rate of hard page faults, as observed from the behavior of stable systems in production, is less than 100/sec. A page fault rate above that causes serious thrashing; performance then degrades until the system becomes unresponsive. The stress tests thus showed that the current distribution of language pairs was not optimal and we have to reallocate them across available model server machines. Multiple iterations of the stress tests at realistic traffic eventually led to a distribution where the virtual bytes usage on the model servers was less than 90% of available RAM and page fault rate was also at acceptable levels.

6. Identifying bottlenecks

The last area in this case study is an example of using stress tests to identify bottlenecks in a distributed AI-based system. Before shipping the Web service to a data center, we have to identify the quantities of each particular type of machine to ship. How many machines should run the distributor, how many should be leafs and how many model servers? A big contributor to this decision is user traffic levels and patterns. Another contributing factor is the effect of the number of machines of each type on the performance of the overall system. Our stress tests revealed that the system did well as long as the ratio of leafs to model servers was 7:1. The performance of the model-servers with this configuration is shown in Figure 4; CPU usage is well below 1% and quite acceptable

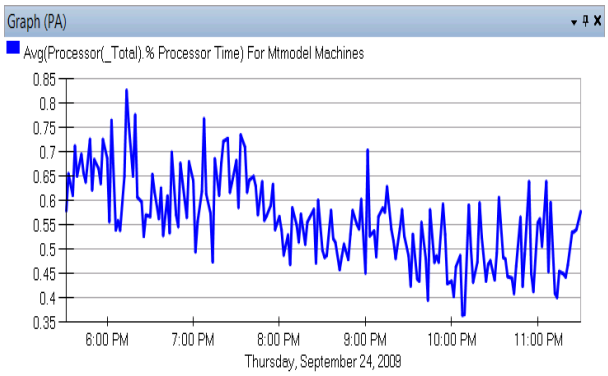

Figure 4. CPU usage.

When we have more than 7 leafs hitting a model server, the traffic generated on 1 model server by the increasing numbers of leafs causes the model server performance to degrade significantly. For example if the ratio is 10:1, the CPU usage increases 10-15 times, as shown in Figure 5. This excessive CPU usage has an effect on the performance of the machines dependant on the model server: the upstream leafs and distributors, and eventually impact user experience, causing slow response times and translation timeouts.

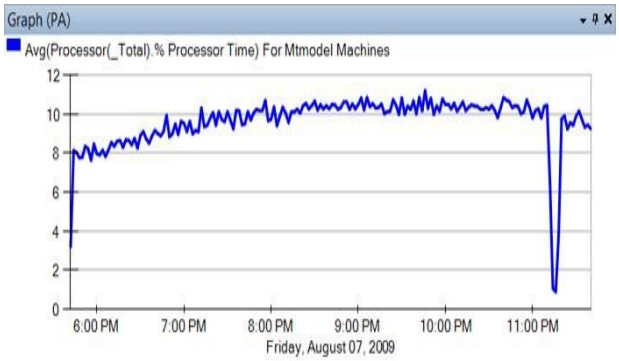

Figure 5. CPU usage increase.

6. Conclusions

The approach taken to stress testing an AI-based Web service outlined above has helped the MSR Machine Translation team provide a stable service as a platform to develop multiple products to a large and rapidly growing base of users. An emphasis on measuring end-to-end performance has made it easier to measure actual user experience. And a discriminating focus on the behavior of individual components has made it easier to identify bottlenecks and make the overall system almost as good as the sum of the constituent parts. The above examples also show that for an AI-based system, it is important to accurately differentiate changes in system response caused by load and changes caused by improvements in the AI algorithm as the system learns over long durations. Implementing and executing tests with these priorities and knowledge has led to a stress testing system that has remained useful and scalable as the Web service has evolved to support a growing number of features and language pairs. This has helped our team successfully ship a product that has seen a huge growth in user base over 2 years, adding 50 times the original number of language pairs, while continuously improving the overall quality of the translations, user-experience and product performance.

7. References

[1] Elaine J. Weyuker, and Filippos I. Vokolos, "Experience with Performance Testing of Software Systems: Issues, an Approach and Case Study", *IEEE Transactions on Software Engineering*, vol. 26, no. 12, December 2000, pp. 1147-1156.

[2] Diwakar Krishnamurthy, Jerome A. Rolia, and Shikharesh Majumdar, "A Synthetic Workload Generation Technique for Stress Testing Session-Based Systems", *IEEE Transactions on Software Engineering*, vol. 32, no. 11, November 2006.

[3] B.M. Subraya, and S.V. Subrahmanya, "Object Driven Performance Testing of Web Applications", *Proceedings of the First Asia-Pacific Conference on Quality Software*, Oct. 2000, pp. 17-26.

[4] Papineni, K., Roukos, S., Ward, T., and Zhu, W. J. (2002). "BLEU: a method for automatic evaluation of machine translation" in *ACL-2002: 40th Annual meeting of the Association for Computational Linguistics,* pp. 311–318.

[5] A. Watkins, D. Berndt, K. Aebischer, J. Fisher, and L. Johnson, "Breeding Software Test Cases for Complex Systems", *Proceedings of the 37th Annual Hawaii International Conference on System Science*, vol. 9, January 2004.

[6] Ilinca Ciupa, Alexander Pretschner, Andreas Leitner, Manuel Oriol, and Bertrand Meyer, "On the Predictability of Random Tests for Object-Oriented Software", *2008 International Conference on Software Testing, Verification, and Validation*, 2008. pp. 72-81.

[7] Mitchell J. Mondro, "Approximation of Mean Time Between Failure when a System has Periodic Maintenance", *IEEE Transactions on Reliability*, vol. 51, no. 2, June 2002.

[8] Jian Zhang, and S.C. Cheung, "Automated Test Case Generation for the Stress Testing of Multimedia Systems", *Software Practice & Experience*, vol. 32, no. 15, December 2002.

[9] Vahid Garousi, Lionel C. Briand, and Yvan Labiche, "Traffic-Aware Stress Testing of Distributed Systems Based on UML Models", *Proceedings of the 28th International Conference on Software Engineering*, 2006, pp. 391-400.

[10] Vern Paxson and Sally Floyd, "Wide Area Traffic: The Failure of Poisson Modeling", *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, June 1995, pp. 226-244.