Performance Analysis of Aspect Oriented Programming for Cloud Service Monitoring

Tejas N. Rao¹,*, Akash Kumar, S. V. Shanmuga Sunder and K. Chandrasekaran

Department of Computer Science and Engineering, National Institute of Technology, Surathkal, Karnataka, India.
e-mail: ¹ntejasn@gmail.com

Abstract. Monitoring is an important task in managing a Cloud Computing environment. But its implementation may result in added software complexity and altering of the offered cloud services. In this paper we demonstrate a monitoring approach called AOP4CSM (Aspect Oriented Programming for Cloud Service Monitoring) which monitors the quality-of-service parameters and provides fault tolerance to the Software-as-a-Service Layer. It is based on aspect oriented programming (AOP) and it modifies neither the server-side implementation nor the client-side implementation of the service. We demonstrate the efficiency of AOP by comparing the results obtained from three services — one without AOP, one with AOP and one with the monitoring built into the source code.

Keywords: aspect-oriented programming, cloud computing, fault tolerance, monitoring, quality-of-service.

1. Introduction

Cloud computing refers to the delivery of computing and storage capacity as a service to a community of end-recipients. These services are delivered in the form of: Software (Software as a Service: SaaS), Platform (Platform as a Service: PaaS) and Infrastructure (Infrastructure as a Service: IaaS). In such a complex environment, service failures are quite likely and consequently, quality of service (QoS) degradations may frequently occur at all layers. The focus of our work is the SaaS layer and we monitor Cloud services at this layer to detect QoS degradations and repair failures using Aspect Oriented Programming.

Aspect-oriented programming (AOP) is a programming paradigm which increases modularity by allowing the separation of cross-cutting concerns. It includes programming methods and tools that support the modularization of concerns at the level of the source code.

Aspect-oriented programming entails breaking down program logic into distinct parts (called concerns). Nearly all programming paradigms support some level of grouping and encapsulation of
concerns into separate, independent entities by providing abstractions (e.g., procedures, modules, classes, methods) that can be used for implementing, abstracting and composing these concerns. But some concerns defy these forms of implementation and are called crosscutting concerns because they “cut across” multiple abstractions in a program. Existing monitoring approaches for Web services observe a variety of QoS parameters, but they require access to the source code of the services being monitored and are typically operated by the provider. In this paper, a non-invasive monitoring approach for Cloud services is demonstrated. It does not require access to the source code of a service, and can be installed by the client. It neither modifies the server implementation, nor the client code.

The paper is organized as follows. Section 2 discusses related work. The methodology of AOP4CSM is presented in Section 3. It also describes the integration of AOP4CSM into the fault tolerance framework and shows the experimental results. Section 4 concludes the paper and outlines areas for future research.

2. Related Work

Shao et al. [2] propose a runtime monitoring model for Cloud computing. Their model, called RMCM (Runtime Model for Cloud Monitoring), collects all Cloud layer performance parameters. In the SaaS layer, RMCM monitors applications with respect to their design models and required constraints. For this issue, it converts the constraints to a corresponding instrumented code and deploys the resulting code at the appropriate location of the monitored applications. This makes RMCM an invasive approach, since it modifies the source code of the applications. Therefore, RMCM differs from our approach, since AOP4CSM is a non-invasive approach. Moreover, RMCM [5] uses interceptors (as filters in Apache Tomcat and handlers in Axis) for service monitoring. AOP4CSM is different, since it is not based on a special environment such as a Java-based application and can be implemented in many contexts.

Boniface et al. [4] propose a monitoring module that collects QoS parameters of SaaS. They use a monitoring application component (AC), which collects QoS parameters at both application and technical level. This AC has to be described and registered in the application repository in order to be used. Description and registration of AC makes this approach complicated and hard to install. However, the installation of AOP4CSM is easier and does not need any tasks of description/registration. Moreover, their approach has not been evaluated yet. In contrast, AOP4CSM has been evaluated and exemplified in the context of fault tolerance.

Ben-Halima et al. [1] propose a middleware (QOSH) for web service monitoring. QOSH monitors QoS parameters of web services. It is based on SOAP header interception. Since this approach enriches SOAP messages with QoS information, QOSH modifies the client and the server implementation to allow QoS parameter evaluation. AOP4CSM differs from QOSH, since it neither modifies the server nor the client implementation.

Jalilian et al. [5] and [6] use aspect oriented programming for developing fault tolerant software. They implement a few fault tolerant mechanisms and make use of the fact that AOP separates crosscutting concerns such as fault tolerance. Some of the fault tolerant mechanisms they have considered are software error detection and correction, acceptance test, time redundancy execution and control flow checking.

Mdhaffar et al. [3] propose a method for monitoring cloud services. It relies on aspect-oriented and object-oriented programming techniques and does not require access to the source code of the
Performance Analysis of Aspect Oriented Programming for Cloud Service Monitoring

service implementation. It is a non-invasive approach since it does not modify the source code of the application.

3. Methodology of AOP4CSM

In this section we present the implementation of the cloud service monitoring approach called AOP4CSM. We have implemented AOP4CSM within our software called Sudoku Solver. The software is used to generate solutions to the popular puzzle game ‘Sudoku’. The software is hosted on the cloud and can be accessed using any web browser supporting JavaScript. On this software we use AOP4CSM to measure time related QoS parameters such as execution time and response time of the cloud service.

3.1 Conceptual model

AOP4CSM is based on Aspect-Oriented Programming (AOP) to intercept method calls at particular points of execution that correspond to relevant events to be monitored. Figure 1 illustrates this – the four red points ($t_1$, $t_2$, $t_3$ and $t_4$) correspond to the four particular points in execution that we would like to intercept.

(i) $t_1$ – corresponds to the instant when the client invokes the request
(ii) $t_2$ – corresponds to the instant when the server receives the request
(iii) $t_3$ – corresponds to the instant when the server sends the response
(iv) $t_4$ – corresponds to the instant when the client receives the response

Using these four values we calculate the various QoS parameters as explained in section 3.3.2. In addition, the aspect code computes the number of successfully processed requests as well as the total number of requests.

![Figure 1. A conceptual model showing how AOP can be used in cloud service monitoring.](image-url)
3.2 Working

AOP4CSM is based on four specific join points. Each one corresponds to an instant \((t_1, t_2, t_3 \text{ and } t_4)\) where AOP4CSM intercepts method calls. AOP4CSM executes the corresponding advice for each join point which consists of (1) saving the timestamp and (2) calculating the number of invocations at the client. When all the instants have been processed, AOP4CSM computes the difference between \(t_4\) and \(t_1\) to deduce the response time. It also computes the difference between \(t_3\) and \(t_2\) to deduce the execution time. The difference between the response time and execution time represents the communication time.

AOP4CSM also checks for input errors such as invalid characters or repetitions, etc. at the instant \(t_2\) and executes the corresponding advice which reports the errors and prevents the further execution of the program.

3.3 Experiment

To demonstrate the fault tolerance and overhead of AOP4CSM we implement three variants of the Sudoku Solver:

(i) \textbf{Without AOP}: In this Sudoku Solver, there are no fault tolerant mechanisms and it consists of only the basic service.

(ii) \textbf{Built-in}: In this Sudoku Solver, the main source code of the service has been modified to include fault tolerance.

(iii) \textbf{With AOP}: In this Sudoku Solver, we implement AOP4CSM. We do not modify the original source code, instead, we implement the fault tolerance mechanism using AOP.

3.3.1 Platform

Following is a list of the various platforms we made use of to implement AOP4CSM:

(i) \textbf{Heroku}: Heroku is a cloud platform as a service (PaaS) supporting several programming languages. Heroku, one of the first cloud platforms, has been in development since June 2007 when it supported the Ruby programming language, but since has added support for Java, Node.js, Scala, Clojure and Python and PHP. The base operating system is Debian or the Debian-based Ubuntu in the newest stack.

(ii) \textbf{AspectJ}: AspectJ is an aspect-oriented extension created at PARC for the Java programming language. It is available in Eclipse Foundation open-source projects, both as stand-alone and integrated into Eclipse. AspectJ has become the widely-used de-facto standard for AOP by emphasizing simplicity and usability for end users. It uses Java-like syntax and has included IDE integrations for displaying crosscutting structure since its initial public release in 2001.

(iii) \textbf{Maven}: Maven is a build automation tool typically used for Java projects. It can also be used to build and manage projects written in Ruby, Scala, and other languages. Maven serves a similar purpose to the Apache Ant tool, but it is based on different concepts and works in a profoundly different manner. Maven is hosted by the Apache Software Foundation, where it was formerly part of the Jakarta Project. Maven uses an XML file to describe the software project being built, its dependencies on other external modules and components, the build order, directories, and required plug-ins.
It comes with pre-defined targets for performing certain well-defined tasks such as compilation of code and its packaging.

Maven dynamically downloads Java libraries and Maven plug-ins from one or more repositories such as the Maven 2 Central Repository.

3.3.2 Parameters

We use AOP4CSM to measure time related QoS parameters which are briefly explained below:

(i) Execution time ($T_{exec}$): measures the time needed to execute a request on the server.

(ii) Response time ($T_{resp}$): defines the time needed to serve a request. It starts when the client sends a request and finishes when the client receives the corresponding response.

(iii) Communication time ($T_{com}$): is the time needed to transfer the request from the client to the server plus the time needed to transfer the response from the server to the client. Thus, the communication time is the response time minus the time necessary for executing the request and the time necessary for processing the messages involved:

$$T_{com} = T_{resp} - (T_{exec} + T_{processing})$$

The processing time consists of only message handling and is negligible in our case.

(iv) Availability: measures the accessibility of a service. It is calculated using the following equation:

$$\text{Availability} = \frac{\text{Successful Requests}}{\text{All Requests}}$$

where Successful Requests represents the number of requests successfully processed and All Requests is the total number of requests sent by the client.

3.4 Results

The computational overhead of AOP4CSM can be obtained by measuring the difference between the execution time of the Sudoku Solver that does not use AOP4CSM and the execution time of the Sudoku Solver that uses AOP4CSM. To obtain the overhead we have run the Sudoku solver on three sets of problems – easy, medium and hard. Each of these sets had 50 different questions. Additionally the problem sets also had a few invalid questions – such as invalid characters, conflicting input, etc.

In the case of the Sudoku Solver without any fault tolerant mechanism, the execution time is between 246 and 328 microseconds for easy problems. The corresponding percentage of requests completed successfully is 88%. The execution time is between $421\mu s$ and $626\mu s$ for problems of medium difficulty. The corresponding percentage of requests completed successfully is 76%. Finally, the execution time is between $703\mu s$ and $858\mu s$ for problems of hard difficulty and the corresponding percentage of requests completed successfully is 60%.

In the case of the Sudoku Solver with the fault tolerance built-in, the execution time is between 252 and 354 microseconds for easy problems. The corresponding percentage of requests completed successfully is 92%. The execution time is between $467\mu s$ and $645\mu s$ for problems of medium difficulty. The corresponding percentage of requests completed successfully is 80%. Finally, the
execution time is between 726 $\mu$s and 878 $\mu$s for problems of hard difficulty and the corresponding percentage of requests completed successfully is 82%.

In the case of the Sudoku Solver with the fault tolerance implemented using AOP4CSM, the execution time is between 257 and 361 microseconds for easy problems. The corresponding percentage of requests completed successfully is 100%. The execution time is between 473 $\mu$s and 649 $\mu$s for problems of medium difficulty. The corresponding percentage of requests completed successfully is 98%. Finally, the execution time is between 741 $\mu$s and 882 $\mu$s for problems of hard difficulty and the corresponding percentage of requests completed successfully is 94% (see Figures 2, 3, 4, and 5).

The following graphs show a comparison of the execution times and success rates of the three Sudoku Solvers:

![Figure 2. Comparison of execution times for easy problems.](image1)

![Figure 3. Comparison of execution times for medium problems.](image2)

![Figure 4. Comparison of execution times for hard problems.](image3)
The discontinuation in the Figure 4 indicates one among the unsolvable problems for the first Sudoku solver.

3.5 Inference

From the results, we can infer the following:

(i) **Efficiency of AOP4CSM**: The results clearly indicate that the difference in the execution times of the Sudoku Solvers with and without AOP is nearly the same. This implies that a cloud service implemented using AOP4CSM is almost as efficient as the original service (without AOP4CSM). Hence, AOP4CSM keeps the service nearly as efficient as it was without AOP4CSM.

(ii) **Low Overhead**: The computational overhead of AOP4CSM, in terms of response time, is so small that it is almost negligible. The difference between the response times of the Sudoku Solvers with and without AOP is just a few microseconds. Hence, AOP4CSM has a very low overhead.

(iii) **Fault Tolerance**: The number of successful requests in the case of the Sudoku Solver with AOP4CSM is much higher than the number of successful requests in the one without any fault tolerance mechanisms. Hence, AOP4CSM can be used to implement fault tolerant software very efficiently.

4. Conclusion

In this paper we have demonstrated an innovative approach to Cloud service monitoring. The proposed approach, called AOP4CSM, is based on Aspect-Oriented Programming. It monitors quality of service parameters of the SaaS layer without modifying the service implementation. The advantages of AOP4CSM are many.

Firstly, AOP makes it possible to separate functional and non-functional concerns’ implementation. Therefore implementing AOP4CSM could reduce software complexity and simplify development.

Secondly, it is a non-invasive monitoring approach and its installation does not need any access to the source code of the service and the client. It has been implemented on our Sudoku Solver to evaluate the approach.
Thirdly, the computational overhead of using AOP4CSM is very low. Experimental results have shown that the computational overhead of using AOP4CSM is very low and that AOP4CSM improves recovery results when applied to fault tolerance.

However, there are a few disadvantages to this approach. Firstly, the environments for software development using AOP have not matured enough yet and they don’t provide complete control over the main source code. Hence, implementing some low level features such as variable monitoring becomes harder. Secondly, object-oriented technology has come a long way from research to industrial applications. It is now used not only in commercial applications but also in critical systems such as satellite software development. AOP on the other hand is still in the research stage and there is a long way to go before it can be widely used in development such as fault tolerant software.

Finally, AOP4CSM is a promising approach and has potentially powerful capabilities in implementing fault tolerance concerns in a transparent manner. There are several areas for future work. For example, AOP4CSM can be used to cooperate with other monitoring tools operating at the infrastructure and platform layers of the Cloud architecture in order to send monitored data to an analysis module that detects failures.

References


