Improving SOA Applications Response Time with Service Overload Detection*

Valeria Cardellini
University of Roma "Tor Vergata"
cardellini@ing.uniroma2.it

Stefano Iannucci
PhD Student at University of Roma "Tor Vergata"
iannucci@ing.uniroma2.it

ABSTRACT

SOA applications are usually built by assembling third-party software services. Even if each single service could offer a contract stating its non-functional properties, enforcing a monitoring policy is needed to detect services overload and consequent contract violations. In this paper we present a monitoring system capable of detecting service state changes. Our experimental results, obtained with a prototype, shows an improvement of 26% of the application response time.

1. INTRODUCTION

Service Oriented Architecture (SOA) is now a popular paradigm for designing and developing distributed applications. Such a paradigm radically shifts the way network applications are designed, since it carries to the extreme the logic of reusable software components by letting software architects easily integrate any software component into their application. Precisely, the ease of integration led to envision a service marketplace [5], where invocations to software components are sold and bought. However, when developing an application by exclusively using third-party components, a software architect must be carefully select which service to use because a wrong choice of a single external component may lead the entire application to fail. For this reason, each service in the marketplace should be labeled with a vector of quality parameters, which represents a proper contract between the service provider and the service user and defines a Service Level Agreement (SLA). In this paper, we are interested in Service Level Objectives (SLOs), i.e., the non-functional parameters belonging to a SLA, and specifically in the response time metric.

A service marketplace may contain several identical services from a functional perspective, but they may differ from a non-functional point of view. SLOs are therefore a good starting point for a software architect to select services that meet both its non-functional requirements and its budget, that is, the SLA the SOA application is willing to offer to its users. However, a number of factors may influence the perceived performance of an external service: network overload, service overload, power outages may cause a service not to fulfill its SLA. In this case, the SOA application must be able to quickly select another service to fulfill its quality requirements, otherwise it would risk to violate its own SLA. In other words, the SOA application must be able to monitor third-party services to detect whether SLOs are met and, in case of contract violations, it must be able to adapt itself to fulfill its own SLO.

Monitoring and adaptation in the SOA context have been largely investigated in the past, e.g., [4, 1]. In this paper, we present the experimental results we obtained by introducing an online adaptive Cusum detector [2] (a state change detection mechanism) into MOSES, a runtime adaptation framework for a SOA-based system [1]. We show that the monitoring system, coupled with the Cusum detector, allows to improve the SOA application response time by 26%.

2. SYSTEM ARCHITECTURE

MOSES is architected in a number of software components, classified according to the MAPE reference model for autonomic systems. In this paper, we focus only on the following MOSES components: (i) Adaptation Manager, (ii) Optimization Engine, and (iii) QoS Engine. Both the Execute and the Monitor MAPE phases are carried out in MOSES by using a BPEL Engine which, instead of directly calling the needed services, invokes the *Adaptation Manager* internal component. The latter is in charge of invoking the third-party services according to the optimal adaptation strategy computed by the *Optimization Engine* and to collect response times and reliability measures for each service invocation. The Optimization Engine belongs to the MAPE Plan macro-component and is triggered whenever the *QoS Engine* detects a change in the SLO of some third-party service. The latter belongs to the MAPE Analyze macro-component and implements the online adaptive cumulative sum (Cusum) algorithm for state change detection.

The online adaptive Cusum detector we implemented [2] is made-up by an Exponential Weighted Moving Average (EWMA) filter [3], that tracks the slow varying mean, and by a two-sided Cusum test with varying thresholds for detecting relevant state changes. We consider the tracking EMWA filter $\mu_i = \alpha y_i + (1-\alpha)\mu_{i-1}$, where μ_i represents the *i-th* average response time computed in the current epoch and y_i represents the *i-th* collected response time sample in the time series. We suppose that when a service provider defines its SLO, it also considers a certain margin of error to be safe in case of little overloads, network problems, and so on. Therefore, we can safely suppose that whenever a service provider violates its SLO, it is certainly in trouble and thus we must be able to detect such situation as quick as possible. So, we set $\alpha = 0.5$, since such a value gives the same importance both to the previously computed average response time and the new perceived response time.

*Poster presented at the 21st International ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC '12).

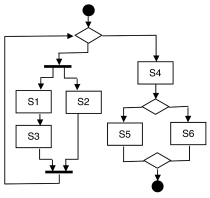

Figure 1: Application managed by MOSES

The Cusum algorithm uses two accumulators, g^+ and g^-, to detect changes on the leading edge and on the trailing edge, respectively. They start both from 0 and then they vary as follows: $g_i^+ = max\{0, g_{i-1}^+ + y_i - (\mu_i + K^+)\}$ and $g_i^- = max\{0, g_{i-1}^- + (\mu_i - K^-) - y_i\}$, where K^+ (K^-) is the smallest shift we want to detect on the leading (trailing) edge. In our experiments, we set it equal to 25% of the response time stated in the SLOs of the monitored services. A change is detected whenever g_i^+ or g_i^- are greater than a threshold H^*, which is computed with a numerical inversion of the Siegmund approximation [2]. In case of change detection, a new epoch starts and the Optimization Engine looks for a new service selection strategy according to the new epoch predicted average response time: $\mu_i = \begin{cases} \mu_{i-1} + K + g_i^+/N^+ & \text{if } g_i^+ > H^* \\ \mu_{i-1} - K - g_i^-/N^- & \text{if } g_i^- > H^* \end{cases}$ where N^+ (N^-) is the number of samples collected from the last leading (trailing) edge change.

3. EXPERIMENTAL RESULTS

We have conducted an experimental analysis using the MOSES prototype to compare a SOA system that does not execute any monitoring activity to a SOA system that monitors the third-party services it uses and reacts to state changes. We consider the SOA system composed of 6 stateless tasks and defined by the workflow in Figure 1, and assume that 4 services (with their respective SLAs) have been identified for each task. We also suppose that, for any given task t_i, each implementing service offers the same SLA in terms of response time and cost. Being $r_{i,j}$ the response time of the *j-th* implementation of t_i, we set $\forall i$ $r_{i,1} = r_{i,2} = r_{i,3} = r_{i,4} = r_i$, so that the optimal service selection turns out to be a balanced round robin. Specifically, we set $r_1 = 2$, $r_2 = 1$, $r_3 = 1$, $r_4 = 0.5$, $r_5 = 2$, $r_6 = 1.8$, all in seconds. Each service is a simple stub, without internal logic, but its non-functional behavior conforms to the guaranteed levels expressed in the SLA. Specifically, its response time is obtained by modeling the service as a $M/D/m/PS$ queue, which is implemented inside a Web service deployed in a Tomcat container. and parameterized in such a way to have an average CPU usage between 65% and 70% when the request rate is equal to 10 req/sec. To issue requests to the SOA application managed by MOSES and to mimic the behavior of users that establish SLAs before accessing the service, we have developed a workload generator. It is based on an open system model, where users requests arrive at mean *user inter-arrival rate*. Each user is characterized by a *contract duration* and is subject to an admission control carried out by the Optimization Engine [1].

The testing environment consists of 3 physical servers, respectively hosting: (i) the Execute and Monitor macro-components, (ii) the services and (iii) the Analyze and Plan macro-components. A KVM virtual machine hosts the workload generator.

Figure 2 compares the SOA application response time when MOSES runs without and with Monitor and Analyze macro-components. In the first case, the average response time is 9.727 sec with a confidence interval of 37 msec and 1.11% of rejected users, while in the second case the response time decreases to 7.707 sec with a confidence interval of 26 msec and 4.13% of rejected users, thus obtaining a 26% improvement on the response time. The horizontal line represents the SLO of the running SOA application. This result clearly shows how monitoring coupled with a state change detector can help tracking the services load, thus obtaining a more effective admission control on the incoming contract requests, which in turn allows not to overload the services.

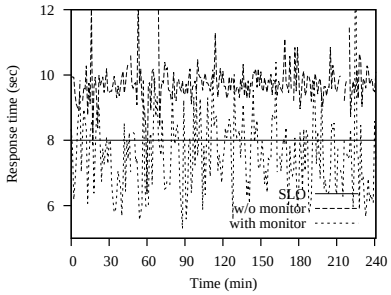

Figure 2: Response time over time

4. CONCLUSIONS

In this paper we showed how the introduction of a monitoring mechanism, together with an effective state change detector, can improve the performance of a SOA application. Our results improved the response time by 26%, but this value could be even greater because our test did not take into account the possibility for third-party services to be invoked by users external to our system, as in a real case. In future work, we plan to evaluate the system when the services are subject to an external noise.

5. REFERENCES

[1] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. Lo Presti, and R. Mirandola. MOSES: a framework for QoS driven runtime adaptation of service-oriented systems. *IEEE Trans. Softw. Eng.*, 2012. to appear.

[2] S. Casolari, S. Tosi, and F. Lo Presti. An adaptive model for online detection of relevant state changes in internet-based systems. *Perform. Eval.*, 69(5), 2012.

[3] D. C. Montgomery. *Introduction to Statistical Quality Control.* Wiley, 2008.

[4] O. Moser, F. Rosenberg, and S. Dustdar. Non-intrusive monitoring and service adaptation for ws-bpel. In *Proc. WWW '08*, pages 815–824, 2008.

[5] E. D. Nitto, C. Ghezzi, A. Metzger, M. P. Papazoglou, and K. Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Autom. Softw. Eng.*, 2008.