Optimizing the Configuration of Web Service Monitors

Garth Heward[1], Jun Han[1], Ingo Müller[1],
Jean-Guy Schneider[1], and Steve Versteeg[2]

[1] Swinburne University of Technology, Hawthorn, Victoria, Australia
[2] CA Labs, Melbourne, Victoria, Australia
{gheward,jhan,imueller,jschneider}@swin.edu.au,
Steve.Versteeg@ca.com

Abstract. Service monitoring is required for meeting regulatory requirements and verifying compliance to Service Level Agreements (SLAs). As such, monitoring is an essential part of web service-based systems. However, service monitoring comes with a cost, including an impact on the quality of monitored services and systems. To deliver the best value to a service provider, it is important to balance meeting monitoring requirements and reducing monitoring impacts. We introduce a novel approach to configuring the web service monitors deployed in a system so that they provide an adequate level of monitoring but with minimized quality impacts, delivering the best value proposition in terms of monitoring benefits and costs. We use a prototype system to demonstrate that by optimizing a web service monitoring system, we can reduce the impact of a set of deployed web service monitors by up to two thirds.

Keywords: Web Services, Monitoring, Monitoring Optimization.

1 Introduction

Web service providers give quality guarantees for their services, so that consumers know the expected Quality of Service (QoS) of services prior to using them. To demonstrate that they have met these guarantees, providers monitor their services. Whilst achieving this goal, service monitoring creates a problem because the monitoring itself reduces the quality of the web services. This impact has been demonstrated to be as much as 40% (for response time) for a single monitor[1]. As such, there is a conflict between the goal of performing monitoring, and the goal of maintaining acceptable quality levels.

To meet these conflicting goals, a web service provider should carefully select what monitoring is performed in the system in order to meet their monitoring obligations without violating any quality guarantees or requirements. Achieving a balance between meeting monitoring requirements and achieving required quality levels by manual configuration is difficult, time-consuming, and unverified for optimality. We propose an automated approach, which allows a service provider to discover an optimal monitoring configuration with less human effort.

P.P. Maglio et al. (Eds.): ICSOC 2010, LNCS 6470, pp. 587–595, 2010.
© Springer-Verlag Berlin Heidelberg 2010

Methods exist for optimizing the performance of web service systems through selective execution or load balancing, e.g. [2,3]. Whilst highlighting the need to optimize the performance of web service systems, none of these techniques relate to the management of deployed service monitors. Only Baresi and Guinea [4] discuss the configuration of web service monitors in order to achieve a trade-off between performance and monitoring. However, they only go as far as manually assigning a resolution of monitoring to each individual service in the system.

We introduce a novel approach that takes as inputs the monitoring requirements (benefits) and the monitoring impacts on system performance (costs), and determines an optimal monitoring configuration that delivers the best value by trading-off these benefits and costs. It considers the quality aspects and monitoring levels for each monitor and service. The optimal monitoring configuration consists of which monitors should be enabled, for what measures of quality, and at what sampling rates, in order to meet monitoring requirements whilst not reducing delivered qualities of service to unacceptable levels.

We demonstrate the benefits (the performance and utility increases) of applying our approach to the deployed monitors of a web service system. We have shown that in the best case, the web services under an optimal monitoring configuration had a response time impact just one third of the impact that a standard monitoring configuration had on the same services.

In Section 2, we introduce our approach to automated monitoring optimization. In Section 3, we discuss the experiments conducted to demonstrate the possible performance benefits of our approach.

2 Monitoring Optimization

As discussed above, to optimize the configuration of deployed web service monitors, their impact on system performance must be minimised, whilst meeting monitoring requirements. As such, the optimization problem is to find a monitoring configuration that gives the maximum value (utility) in terms of performance impact (costs) and monitoring coverage (benefits). Figure 1 shows a framework illustrating our approach to monitoring optimization, annotated with the ordering of activities of the optimization process. In step **1**, the set of deployed monitors is identified based on IT records, and then the Enumerator generates all the possible monitoring configurations for this set of monitors. In step **2**, the performance impacts of each monitor are identified from IT Records or benchmarking, and the Impact Analyser derives the total performance impact of each possible monitoring configuration. In step **3**, the monitoring requirements and associated penalties for not meeting them are identified from analysing SLAs, policies and laws, and the Requirements Analyser transforms requirements into a set of utility functions that define the benefit gained from monitoring a quality whilst a specific level of performance is being achieved. In step **4**, the Optimizer uses the set of utility functions from the Requirements Analyser to score all monitoring configurations with impacts from the Impact Analyser, and the monitoring configuration that yields the highest utility is selected and applied to the monitoring system. Each framework component is described below in more detail.

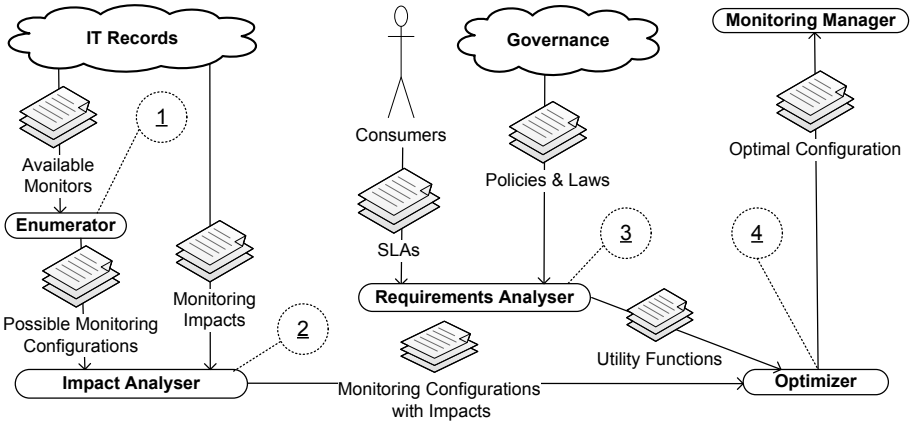

Fig. 1. Monitoring Optimization Framework

Enumerator. The Enumerator takes as input the set of available monitors, and outputs the set of all possible monitoring configurations.

A monitor can monitor one or more quality types of one or more services, simultaneously or separately. We refer to the set of all quality types of a system as Q, and the set of all services as S. A *monitoring capability*, $mcap = (m, s, q) \mapsto SR$ represents the ability of a monitor $m \in M$ to monitor a quality $q \in Q$ of a service $s \in S$ at a sampling rate $sr \in SR$. A *monitoring setting*, $ms = ((m, s, q) \mapsto sr, sr \in mcap(m, s, q)$ represents a monitoring capability set at a particular sampling rate sr. MS represents the set of all possible monitoring settings. A set of monitoring settings obtained by assigning every monitoring capability to a sampling rate is called a monitoring configuration, $mc \in MC$ and $mc = \{(m, s, q) \mapsto sr \in mcap(m, s, q) | \forall (m, s, q) \in MCAP\}$, where $MCAP$ is the set of all the deployed monitoring capabilities in the system.

For example, if we have monitor m_1 capable of monitoring quality q_1 of service s_1 at sampling rates of 0, 0.5, or 1.0, and monitor m_2, capable of monitoring quality q_1 and q_2 of service s_1 at sampling rates of 0 or 1.0, the three monitoring capabilities are $\{(m_1, s_1, q_1) \mapsto (0, 0.5, 1.0), (m_2, s_1, q_1) \mapsto (0, 1.0), (m_2, s_1, q_2) \mapsto (0, 1.0)\}$. One of the monitoring configurations is $mc = \{(m_1, s_1, q_1) \mapsto 0.5, (m_2, s_1, q_1) \mapsto 0, (m_2, s_1, q_2) \mapsto 1.0\}\}$. The Enumerator generates all possible monitoring configurations MC by stepping through all the allowable combinations of monitors, services, qualities and sampling rates. For the above example, the output from the Enumerator is presented in Table 1, where a row corresponds to a monitoring capability, a column corresponds to a monitoring configuration, and a cell is the specific sampling rate of the corresponding monitoring capability in the relevant monitoring configuration. The monitoring configuration mentioned above is column mc_8 in Table 1.

Table 1. Monitoring Enumeration

	mc_1	mc_2	mc_3	mc_4	mc_5	mc_6	mc_7	mc_8	mc_9	mc_{10}	mc_{11}	mc_{12}
$m_1s_1q_1$	0	0.5	1.0	0	0.5	1.0	0	0.5	1.0	0	0.5	1.0
$m_2s_1q_1$	0	0	0	1.0	1.0	1.0	0	0	0	1.0	1.0	1.0
$m_2s_1q_2$	0	0	0	0	0	0	1.0	1.0	1.0	1.0	1.0	1.0

Impact Analyser. The Impact Analyser takes as input all possible monitoring configurations MC, all monitor setting impacts MI and monitor overheads MO, and outputs the set of monitoring configurations with impacts MCI.

il denotes the *impact level* on a quality q of a service s, $(s, q) \mapsto il$. Impact level $il \in IL$ is an impact as a percentage on the original quality. For example, the impact of $((s_1, q_1) \mapsto 0.5)$ means the level of q_1 of s_1 is halved.

Monitoring has both overhead and monitor setting impacts. The overhead mo of a monitor m on a service s occurs whenever the monitor is in use, regardless of the activities of the monitor. Let SQ be all the service-quality pairs of concern in the system, the overheads a monitor has on every relevant quality q of every service s are $mo = \{(s, q) \mapsto il : IL | \forall (s, q) \in SQ\}$. For example, an interceptor m_1 that redirects messages of services s_1 and s_2 for analysis may reduce response time of all messages passing through it by 10%, whether or not those messages are actually analysed. Let $MIS = SQ \rightarrow IL$, the set of all monitor overheads is the collection of all the monitors' overheads, $MO = \{m \mapsto \mathcal{P}(MIS) | \forall m \in M\}$.

The monitoring setting impacts msi represent the impacts that a monitoring setting ms has on each quality q of each service s in the system. That is, $msi = \{(s, q) \mapsto il : IL | \forall (s, q) \in SQ\}$. For example, IT records may show that when monitor m_1 is configured to measure q_1 of s_1 at sampling rate 0.5, it reduces q_1 of services s_1 and s_2 by 10%. The impacts of this monitor setting are $\{(s_1, q_1) \mapsto 0.1, (s_2, q_1) \mapsto 0.1\}$. The set of impacts of all possible monitor settings in MS is: $MI = \{ms \mapsto msi : \mathcal{P}(MIS) | \forall ms \in MS\}$.

The impact of a monitoring configuration mc on quality q of service s is made up the following components: (1) the sum of the impacts of all monitoring settings on q of s: $I_{s,q}(mc) = \sum_{ms \in mc} MI(ms)(s, q)$; and (2) the sum of all the monitors' overheads on q of service s: $O_{s,q}(mc) = \sum_{m \in M} MO(m)(s, q) \times inUse(m)$, where $inUse(m) = \{^{1 iff \exists mc_i \in mc, mc_i.m = m, mc_i.sr > 0}_{0 iff \nexists mc_i \in mc, mc_i.m = m, mc_i.sr > 0}\}$. Therefore, mc's impacts on q of s are: $IM(mc)(s, q) = I_{s,q}(mc) + O_{s,q}(mc)$, and mc's impacts on all (s, q) pairs are: $IM(mc) = \{(s, q) \mapsto IM(mc)(s, q) | \forall (s, q) \in SQ\}$. Finally, the set of all monitoring configurations with impacts (on service-quality pairs) is: $MCI = \{mc \mapsto IM(mc) | \forall mc \in MC\}$.

Requirements Analyser. The Requirements Analyser takes requirements from SLAs and other sources such as corporate governance as input, and outputs a set of utility functions representing the values of achieving those requirements.

We assume that requirements can be translated into statements that describe a fine for not meeting or not monitoring a QoS. For example, an SLA may require a service s_1 to have response time of 15 seconds and sampling rate of 50% to demonstrate a QoS has been met, with a penalty of $100 for exceeding this.

We say that the benefit is \$100 for monitoring service s_1 for response time whilst that response time is less than 15 seconds with a sampling rate of at least 0.5.

A requirement that the quality q of service s must be at least ql and monitored with a sampling rate at least sr can be represented as $r = (s, q, ql, sr)$, and the requirement r with a $fine$ for non-compliance can be represented as $rp = ((s, q, ql, sr) \mapsto fine)$. The quality level ql is a fraction of the required quality level over the *ideal quality level* that is provided by the service. For example, if the best possible response time of a service is 10 seconds, and the required response time is 20 seconds, then the quality level ql is 0.5. The $fine$ is the penalty that must be paid if the sampling rate sr and the quality level ql are not met. The set of all requirements with fines is $RP = \{(s, q, ql, sr) \mapsto fine : \mathbb{R} | \forall (s, q, ql, sr) \in R\}$, where R is the set of all requirements.

Consider the requirements that the quality q of service s should be monitored all the time and achieve a quality level of 0.75, with a fine of \$60 if the quality level goes below 0.75, and a fine of \$90 if the quality level goes below 0.5. This set of requirements with fines can be represented as $rps = \{(s_1, q_1, .50, 1) \mapsto \$90, (s_1, q_1, .75, 1) \mapsto \$60, (s_1, q_1, 1, 1) \mapsto \$0\}$.

We transform each monitoring requirement describing fines for non-compliance into *utility functions* describing benefit (utility) for compliance. Since our utilities are cumulative, for $rp = ((s, q, ql, sr) \mapsto fine)$, we have the corresponding utility $util(s, q, ql, sr) = (curmax - RP(s, q, ql, sr)) - \sum_{lr \in LR(s,q,sq,sr)} (curmax - util(lr))$, where $curmax$ is the benefit of achieving the quality level ql and sampling rate sr, and LR is the set of requirements with a quality level $lr.ql < ql$ or sampling rate $lr.sr < sr$, $LR(s, q, ql, sr) = \{(s, q, ql_1, sr_1) | \forall (s, q, ql_1, sr_1) \in R((ql_1 < q_1) \wedge (sr_1 \leq sr)) \vee ((ql_1 \leq q_1) \wedge (sr_1 < sr))\}$. Therefore, the utility function covering all the monitoring requirements is $U = \{(s, q, ql, sr) \mapsto util(s, q, ql, sr) | (s, q, ql, sr) \in R\}$.

For the above set of example requirements rps, we have the following corresponding utilities: $us = \{(s_1, q_1, .50, 1) \mapsto \$0, (s_1, q_1, .75, 1) \mapsto \$30, (s_1, q_1, 1, 1) \mapsto \$60\}$. The last element $((s_1, q_1, 1, 1) \mapsto \$60)$ of the utility set us is read as "achieving quality level greater than .75 of quality q_1 for service s_1 with a sampling rate of 1 is worth an additional \$60 (relative to the next lower quality level of 0.75), since fines of that amount will not have to be paid". The absolute utility for the $ql = 1$ level is \$90, including the utility at the lower quality level (\$30). Designers may also add utilities that are not directly derived from requirements.

Optimizer. The Optimizer takes as input the set of utilities U, the set of monitoring configurations with impacts MCI, and the *ideal service quality levels* for all qualities of all services $IQL = \{(s, q) \mapsto iql : QL | \forall s \in S, \forall q \in Q\}$, where iql is the pre-determined best achievable quality level for quality q of service s. The Optimizer outputs a monitoring configuration that gives the best utility in terms of meeting requirements and reducing monitoring impacts.

For a monitoring configuration mc with impacts, the Optimizer first obtains the Final Quality Levels FQL of all (s, q) pairs by subtracting all relevant impacts from their ideal quality levels IQL, i.e., $FQL(mc) = \{(s, q) \mapsto fql : QL | iql = IQL(s, q) - MCI(mc)(s, q)\}$.

The utility of the (s, q) pair under monitoring configuration mc, $u(mc)(s, q)$, is the sum of those utilities of (s, q), (1) whose required quality level ql has been met by the pair's final quality level $FQL(mc)(s, q)$ and (2) whose sampling rate sr is met by at least one monitor setting in mc.

Let $R_{(s,q)}$ be the set of all requirements concerning (s, q), i.e.,$R_{(s,q)} = \{(s_1, q_1, ql, sr) | (s_1, q_1, ql, sr) \in R \wedge s_1 = s \wedge q_1 = q\}$. Then,$u(mc)(s, q) = \sum_{(s,q,ql,sr)\in R_{(s,q)}} (util(m, s, ql, sr) \times a \times b$, where

$a = \{^{1 iff\ FQL(s,q) \geq ql}_{0 iff\ FQL(s,q) < ql}\}, b = \{^{1 iff\ \exists m \in M, ms(m,s,q) \geq sr}_{0 iff\ \nexists m \in M, ms(m,s,q) \geq sr}\}$.

The total utility for the monitoring configuration mc will be the sum of the utilities for all (s, q) pairs under the configuration, $u(mc)$. Let $SQ_{mc} = \{(s, q) | \forall (m, s, q, l) \in mc\}$. Then, $u(mc) = \sum_{(s,q)\in SQ_{mc}} u(mc)(s, q)$. The set of all monitoring configurations with utility is, $MCU = \{mc \mapsto u(mc) | mc \in MC\}$. A monitoring configuration that gives the highest total utility will be an optimal monitoring configuration, $optimal_{mc} \in \{mc_1 | \forall mc_2 \in MC, MCU(mc_1) \geq MCU(mc_2)\}$, which may be applied to the monitors in the system either manually or automatically (if available via methods such as SNMP or WS-Management).

Optimization Complexity. The number of monitoring configurations is the product of the number of monitoring levels (ML_{mcap}) for all monitoring capabilities in the system ($mcap \in MCAP$). The size of this search space is bounded by $\mathcal{O}(avg(|ML_{mcap}|)^{|MCAP|})$. Since we step through the search space once for each of Impact Analysis and Optimization, this is also the time complexity for our optimization technique. The problem is a combinatorial optimization problem with both overhead (fixed) and instance (variable) costs, a case of an integer fixed charge network flow (FCNFP) problem, demonstrated to be NP-hard [5]. We have provided a brute force technique to demonstrate the possible performance and utility benefits of applying optimization. This also provides a baseline of a 'perfect' optimization against which to compare future heuristic algorithms.

3 Performance Evaluation

A prototype[1] of a travel agency with two web services, five monitors, four quality types, six SLA requirements, four Corporate Governance requirements, and 2.5×10^5 possible monitoring configurations has been implemented to verify our optimization technique and measure possible performance benefits of monitoring optimization. There always exists one or more optimal monitoring configurations (yielding the highest utility) in terms of monitoring coverage and performance. The purpose of these experiments is to discover these optimal monitoring configurations, and compare the utility and performance of each optimal configuration to the utility and performance of each corresponding maximum (un-optimized) monitoring configuration, in which all monitors run at a 100% sampling rate.

[1] Full details at http://www.ict.swin.edu.au/personal/gheward/

The performance measurements are based on a series of simulated service executions, which use response time measured through real service invocations to determine what the performance will be for a given monitoring configuration.

3.1 Results

We report average response times and utilities to demonstrate that system performance has increased whilst monitoring requirements have been met. Utility increased in each experiment, as only the minimum valuable monitoring level was met, allowing for a performance increase which minimised the chance of breaching requirements for response time. Figure 2 shows response time versus load level with unmonitored, maximum, and optimal monitoring configurations. The load levels on the horizontal axis represent the average number of active client requests, and the vertical axis is the average response time over both services.

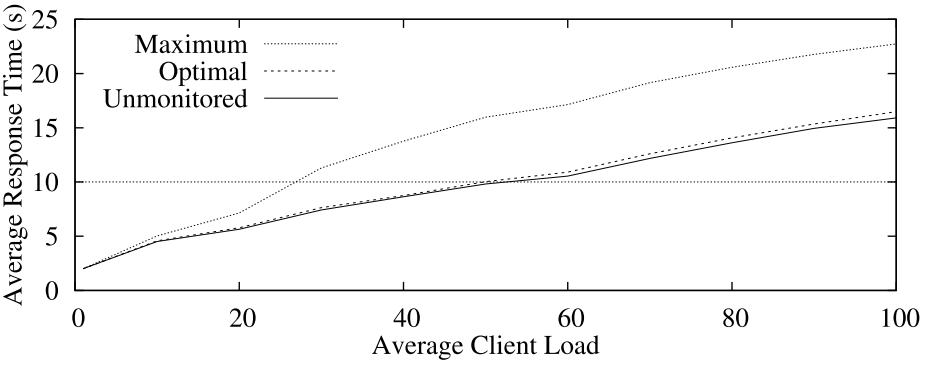

Fig. 2. Performance Evaluation

Optimization reduced the average response time by approximately 30% from maximum monitoring, and reduced the average response time impact by 80%.

The horizontal line on Figure 2 shows the 10-second boundary, for which penalties apply in the test system. The optimized monitoring configuration stays under this 10-second boundary for approximately twice as long as the maximum monitoring configuration, i.e. the system with optimized monitoring dealt with twice as much load before a penalty would have been paid.

The utility provided by the unmonitored solution was 0, since no requirements could be verified. The Maximum and Optimal monitoring configurations both included complete monitoring coverage. For each of these two configurations, the utility at or below 10 seconds response time was 4.245, and the utility above 10 seconds response time was 3.75. We repeated all experiments with randomly generated utility functions and monitoring impacts ranging from mild (less than .01% per monitoring instance on average) to severe (up to 20% per monitoring instance), under various load levels. The response times for the optimized solutions were on average 40% lower than maximum monitoring, and the optimized

response time impacts were on average 70% lower than maximum monitoring. Utility values were on average 30% higher for the optimized solutions.

4 Related Work

Ranganathan and Dan present a Web Services management system to monitor and reallocate local system resources for services based on comparing their current QoS to their SLAs[6]. Whilst performing system-level administration, this method does not re-configure the web service monitoring system.

Baresi and Guinea present a method for dynamic monitoring of WS-BPEL processes, which uses high level monitoring rules [4]. These Monitoring Rules are used to control the monitoring of each WS-BPEL process. The rules are created with an equivalent of debug levels (one to five), which allows for optimization in terms of performance versus monitoring trade-offs at run-time. The monitoring level must be set for each service as a unit. Rather than assigning a monitoring resolution (debug level) to each individual service in the system, we assign a monitoring resolution to each quality of each service, directly reflecting requirements from SLAs. Furthermore, we enhance the optimization by selecting those monitors which will most efficiently monitor each service.

Overall, there have been numerous efforts for optimizing the QoS of web services and web service compositions, which consider the selection of services in order to optimize QoS. We have discovered no work that optimizes a web services monitoring system by trading off between monitoring costs and benefits, directly translated from SLAs and other requirements for monitoring.

5 Conclusions and Future Work

We have presented a framework and techniques for optimizing the configuration of web service monitors, in order to maximise utility for a web services provider.

A prototype instantiation of our proposed framework was used for a series of experiments, which demonstrated that both utility and performance can be improved by optimally configuring a web services monitoring system. Results indicate that the performance impact of web service monitoring can be significantly reduced, whilst meeting monitoring requirements.

We plan to extend the framework and implementation so that run-time properties of the system being monitored are fed back into the monitoring optimization framework, for re-optimization at run-time. We will also develop a heuristic approach to optimization in order to ensure scalability of the technique.

Acknowledgements. This work is supported by the Australian Research Council in collaboration with CA Labs.

References

1. Heward, G., Müller, I., Han, J., Schneider, J.G., Versteeg, S.: Assessing the performance impact of service monitoring. In: Australian Software Engineering Conference (ASWEC 2010), pp. 192–201 (2010)

2. Ludwig, H., Dan, A., Kearney, R.: Cremona: An architecture and library for creation and monitoring of ws-agreements. In: International Conference on Service Oriented Computing (ICSOC 2004), vol. 74, pp. 65–74 (2004)
3. Cardellini, V., Casalicchio, E., Grassi, V., Lo Presti, F.: Flow-based service selection for web service composition supporting multiple qos classes. In: International Conference on Web Services (ICWS 2007), pp. 743–750 (2007)
4. Baresi, L., Guinea, S.: Towards dynamic monitoring of ws-bpel processes. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 269–282. Springer, Heidelberg (2005)
5. Stevens, T., Vermeir, J., Leenheer, M.D., Develder, C., Turck, F.D., Dhoedt, B., Demeester, P.: Distributed service provisioning using stateful anycast communications. In: Annual IEEE Conference on Local Computer Networks (LCN 2007), pp. 165–174 (2007)
6. Ranganathan, K., Dan, A.: Proactive management of service instance pools for meeting service level agreements. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 296–309. Springer, Heidelberg (2005)