Assessing the Performance Impact of Service Monitoring

Garth Heward, Ingo Müller, Jun Han, Jean-Guy Schneider
Swinburne University of Technology
Melbourne, Australia
{gheward,jhan,imueller,jschneider}@swin.edu.au

Steven Versteeg
CA Labs
Melbourne, Australia
Steve.Versteeg@ca.com

***Abstract*—Service monitoring is an essential part of service-oriented software systems and is required for meeting regulatory requirements, verifying compliance to service-level agreements, optimising system performance, and minimising the cost of hosting Web services. However, service monitoring comes with a cost, including a performance impact on the monitored services and systems. Therefore, it is important to deploy the right level of monitoring at the appropriate time and location in order to achieve the objectives of monitoring whilst minimising its impact on services and systems. Although there have been many efforts to create Web services monitoring techniques and frameworks, there has been limited work in quantifying the impact of Web service monitoring. In this paper, we report on experiments assessing the performance impact of service monitoring under typical system monitoring settings. The performance impact of monitoring method, monitor location, monitor processing capability, and monitoring mode are taken into consideration. Based on the experimental results, we advise on the most appropriate ways to deploy service monitoring.**

***Keywords*-Web service monitoring; performance impact; service interceptors;**

I. Introduction

Service-Oriented Architecture (SOA) provides for distributed software with loose coupling, user-level composition, and a high level of business support through aspects such as service-level agreements and various management standards [1]. Web services represent the dominant means for implementating Service-Oriented Architectures. Since Web services are usually under the control of third parties, their properties, such as response time, are important when considering the overall quality of an application that uses them and are often used to verify service-level agreements and optimise service compositions or Web service infrastructures [1].

Certain service properties (*e.g.*, response time) can only be determined at run-time via monitoring [2]. Monitoring is generally performed by a *monitoring agent*, a component in a Web services system that monitors some aspect of that system. For example, a monitoring agent may be a Web service proxy-based interceptor that logs all messages that it sends and receives. Monitoring agents may also perform actions based on some initial measurement, such as blocking a message due to a security violation.

A service provider may need to monitor and manage a number of properties of a Web services system, such as performance (response time, resources consumed, throughput etc.), security (security model, trust in partners, certificate quality, key quality etc.), reliability, and availability [3], [4]. Even in simple scenarios with only one or two of these quality properties being monitored, the cost of monitoring the entire system all the time may be greater than the benefit provided by doing so [2]. For example, using a proxy-based interceptor to intercept and log messages in order to measure response time of a Web service may increase the response time of that Web service to an unacceptable level under a service-level agreement. In this case, the service provider may choose to monitor the Web service using a different method, or not monitor the Web service at all.

Unfortunately, there are cases where service monitoring is mandated by law, contract, or business policy. The specific method of monitoring may also be mandated, leaving no choice to the service provider. Even in these situations, knowing the cost of monitoring is still valuable in order to assign resources to the system or bill any costs of monitoring to external parties.

However, there are situations where the designer of a Web services monitoring system is able to choose how they meet regulations for monitoring. In these cases, monitoring can be appropriately optimised. Optimisation of a monitoring system may involve selecting exactly what aspects are monitored, at what resolution, at what times, and for what qualities. To achieve such an optimisation, the cost of Web services monitoring must be determined.

Existing research into service monitoring techniques and frameworks has described the impact of monitoring or management using a variety of approaches (cf. [5]–[9] just to name a few). We have summarized these techniques and frameworks in Table II, including their overheads or impacts as measured by the original authors. Our survey has uncovered no studies that determine the impact of Web services monitoring at a fundamental level. For example, we have discovered no existing studies that answer the question "is the performance impact of a SOAP proxy greater than the performance impact of software level eavesdropping?". As such, it is difficult to perform a direct comparison between techniques without implementation and testing. Therefore, we have conducted a range of experiments to measure and quantify the performance impact of monitoring a Web

1530-0803/10 $25.00 © 2010 IEEE
DOI 10.1109/ASWEC.2010.28

services system based on a new classification of monitoring techniques. This paper reports on these experiments and the results obtained. For this study, we have focused our attention on *response time*. This property was chosen as it is a common measure of performance, is a requirement in many service level agreements, and is easily measurable and quantifiable.

One of our key findings is that it is possible to monitor Web services with little to no measurable impact on response time. However, the method of monitoring used to achieve this does not allow for blocking or 'firewalling' actions by the Web service monitor. When these actions are taken into account in a web service monitor, we have measured impacts on response time of a Web service of up to 40%.

The rest of this paper is organised as follows: Section II introduces the testing methodology used for measuring the performance impact of monitoring Web service. In Section III, we present the main results or our experiments. Section IV provides a discussion and interpretation of these results and their respective impact, followed by a review of related work in Section V. Finally, Section VI presents our conclusions and discusses future work.

II. Methodology

In this section, we present the experiments designed to measure the performance impact of monitoring web services. After introducing a classification of monitoring techniques, we discuss the experimental variables, system setup, and platform details.

A. Classification

The classification given in Figure 1 is a generic monitoring classification that holds for all OSI/technology stack levels. For example, a generic proxy type monitor may be realised using a hardware proxy such as a network router or a routing server. Proxies include all methods of monitoring that involve implicit or explicit redirection of traffic between the source and the intended recipient. This includes wrappers for virtual machines or runspaces, since traffic must pass through them, even at a software level, and they are able to modify or redirect that traffic.

Probes include all methods of monitoring that generate their own messages or other traffic intended for the target system. This may include simulated consumer messages and invocations of test procedures. This also includes software that requests data from remote logs.

Eavesdroppers include all methods of monitoring that passively intercept communications between the consumer and provider. This may include methods such as network level interception using logs on routers, software level interception using modifications of the operating system or network interface drivers, or modification of existing SOAP proxies. This also includes systems that receive messages from a heartbeat or broadcast.

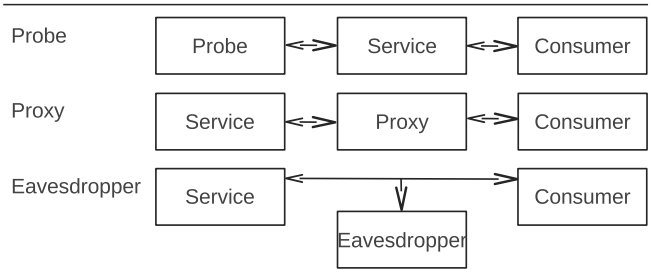

Figure 1: Classification of monitoring techniques.

B. Experiment Variables

It is expected that the cost of monitoring will vary with the method and level (amount) of monitoring. Our experiments are specifically designed to measure the *relative* cost of different monitoring methods and levels under our monitoring classification scheme (cf. Figure 1). The monitoring methods selected were monitoring using a software-based communications eavesdropper (Wireshark) [10], which represents the eavesdropper type monitor in Figure 1; and monitoring using a Web service based intercepting proxy, which represents the proxy type monitor in Figure 1. These methods of monitoring were selected because they are commonly used by others [5], [8] and are easily repeatable.

The impact of probe-based monitoring is not included in this paper, as the impact of probe-based monitoring is not expected to be consistent across systems, due to the many possible types of probes that may have impacts ranging from negligible to extreme. For example, a probe may request the current CPU usage, or it may stress test a server.

Other methods of monitoring are available, such as instrumentation of Web service or client code, and instrumentation of virtual machines or runtimes. The performance of instrumentation-based methods is not measured here, since these methods require access to source code or virtual machines, respectively, which may not always be available. These instrumentation based methods are also restricted in that instrumentation-based monitors can only be placed where a virtual machine or application exists, whereas network level proxies may be placed anywhere along communication paths. Some of the results of proxy-based monitoring testing that are presented here may however be applied to instrumentation-based monitoring. For example, a major cause of delay for a blocking proxy is the action that is taken by the proxy, rather than the overhead of hosting the proxy itself. In this case, an instrumentation-based proxy will yield a similar performance impact to other types of proxies such as those used in our tests.

Three variants of the proxy-based monitor were created. First, a basic, "empty" proxy was used as a baseline for the cost of this method of monitoring. Second, a variant wherein the proxy has a constant delay was created in order to establish the cost of a delay in the proxy. Third, a variant wherein the proxy performed processing that was equivalent

to the processing performed by the Web service was created in order to establish the cost of a proxy-based monitor that is performing some heavy processing. This variant has 'blocking' and 'non-blocking' forms. The blocking version receives a message, performs the required processing, and then forwards the message. Conversely, the non-blocking version receives a message, forwards the message, and then performs the required processing.

In general, to achieve the experiment objectives, we consider the following variables for the monitoring system:

(1) The use of a Web service proxy-based interceptor (hereafter referred to as proxy);
   (a) The type of action(s) that the Web service proxy takes;
   (b) The location of the Web service proxy-based monitoring agent;
   (c) Whether the proxy is blocking or non-blocking; and

(2) The use of an eavesdropping, software-based web service monitoring system.

The first monitoring method (1) was used to determine the performance impact of using a proxy, compared to directly invoking a Web service. Once this impact was established, any processing that was performed at the proxy could be compared with this baseline response time. Variable (1.a) was used to determine the performance impact of different actions that a proxy takes, such as logging, or performing some CPU/RAM intensive task, which may be likened to XML serialisation/de-serialisation or encryption/decryption. Variable (1.b) was used to determine the impact of the proxy's location. The location was either on the server hosting a Web service, or on a dedicated node. Variable (1.c) was used to determine the difference between a blocking and non-blocking proxy. Each of these methods was measured using 10, 20, 30 and 40 simultaneous clients in order to determine the impacts of the level as well as method of monitoring. The second monitoring method (2) was used to determine the impact of monitoring a Web services system without directly modifying that system. In this case, monitoring was via software instrumentation of all TCP/IP communications.

We expect that any monitoring taking place at the consumer side will not have an impact on the provider's system (unless of course the consumer and provider are co-located). Therefore, the system setups have been designed to measure performance impact of monitoring on a Web services system, when monitored from the service provider's side only.

In order to measure the impact of the variables listed above, a Web service monitoring system was created. This system had the following configurations:

(1) Location of Proxy
   (a) Nowhere - direct invocation (Figure 2, Type 1)
   (b) Local (Figure 2, Type 2)
   (c) On a dedicated node (Figure 2, Type 3)

(2) Actions of Proxy
   (a) None (Empty)
   (b) Static delay
   (c) Relative load
      (i) Blocking
      (ii) Non-Blocking

(3) Use of eavesdropping

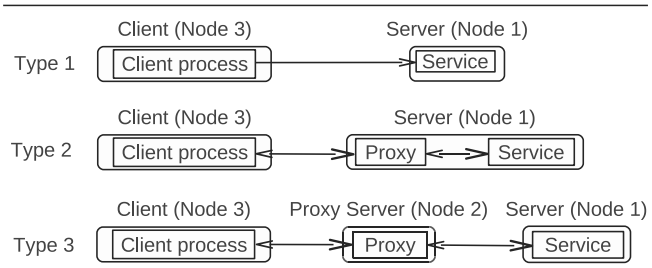

Figure 2: Web service monitoring scenarios.

Configuration (1) was used to test variable (1.b), configuration (2) was used to test variables (1.a) and (1.c), and configuration (3) was used to test software-based eavesdropping (variable (2)), respectively.

C. System Setup

The system setup involved the use of a single Web service, a Web service client load generator, and a Web service proxy-based intercepting monitor. Both blocking and non-blocking proxies were developed.

The web service has a positive natural number as input parameter, and returns π to that number of decimal places, as a string. For example, executing `pi_service(3)` returns "`3.142`." The computation of π is based on the Gauss-Legendre algorithm. Through this computation, this service stresses the RAM and CPU of the hosting server, and allows for varying degrees of load based on the input parameter. Apart from this flexibility, this method of load-generation was chosen because it is easily repeatable, and can provide consistent results. This method of load generation generates a relatively high level of server-side processing per client request, which is useful since the server's processing capacity can be stressed without stressing other elements in the test bed, such as the client system or network. Since Web services are designed as interfaces to existing systems, stressing these aspects of the system are realistic (*e.g.*, stressing of disk I/O would more likely represent the behaviour of an application that a web service is the interface for). The Web service was created in Java/J2EE and hosted on an Apache/Tomcat server.

The Web service proxy-based interceptor is a simple redirection service. The proxy was modified to perform

various tasks such as sleeping (artificial static delay), and computing π locally (artificial relative load). The static delay allows for the measurement of change in response time due to a known and static delay. This would answer the question "what is the response time cost of a 5 second delay in a Web services proxy-based monitor?". The artificial relative load allows a load to be put on the monitor that is equivalent to the load on the server. This allows for a simple baseline measurement of the impact of monitoring with a level of processing equivalent to that of the Web service itself. The Web service proxy was created in Java/J2EE and also hosted on an Apache/Tomcat server.

The Web service client load generator invokes the `pi_service` method on the Web service and measures the total time taken (response time) from when the service is invoked by the client until the result is returned to the client. Apart from the number of places for which to compute π, the client application takes as input `num_threads`, the number of threads (simultaneous clients) to execute. Modifying the number of simultaneous clients allows for the simulation of varying loads on the servers hosting the Web service and proxies, respectively. The client simultaneously executes the Web service `num_threads` times and returns the average of the response times for all threads.

For eavesdropping, Wireshark was used to monitor the IP traffic for each node's network interface. Response times were measured with Wireshark running on one, two, or three of the nodes in order to determine if there was any impact.

Figure 2 shows the test setups that were used to exercise the required test scenarios. Rectangles with rounded corners represent node boundaries. Rectangles represent services or processes, whereas arrows represent communication links.

Type 1 in Figure 2 shows the baseline system, with direct invocation of a Web service (no proxy). Type 2 shows a proxy co-located with the Web service, and Type 3 shows the proxy on a separate server to the Web service.

Each test result presented is the average of at least 15 test runs (more tests were run to increase confidence, where required) at 10, 20, 30, and 40 threads and 10,000 digits of π. Each average has vertical error bars showing the standard error at 95% confidence for that value. These error bars demonstrate the separation of the average response times.

D. Platform Details

Table I presents the platform details of the test setup. Each node involved in the test was running Microsoft Windows XP SP3. The tests were run on a dedicated LAN with no other traffic. All addressing was via IP (no DNS lookups). Apache/Tomcat was used for hosting the Web services. The network was never under more than 5% load during tests. This eliminates the chance that the network was a 'bottleneck' in the system. None of the systems involved in the test required the use of swap space during any test.

Table I: Node Specifications

	Node 1 (server)	**Node 2 (proxy server)**	**Node 3 (client)**
CPU	Intel E8500 @ 3.16GHz	Intel T7700 @ 2.4GHz	T5600 @ 1.8GHz
RAM	4GB	3GB	1GB

The systems selected for this benchmark cannot be representative of every possible combination of client, server, and proxy that is used. However, the comparative results from these tests will scale to any systems of similar architecture. That is, we expect that running the same tests on a group of high performance servers would yield faster response times, but the difference between response times for different monitoring methods would remain.

III. Results

Using the experimental method discussed in the previous section, we have conducted a wide range of performance tests with the various monitoring system configurations. The following sections present the results of the experiment under the most representative scenarios.

A. Location of Proxy

Figure 3 shows the results from testing the system using no proxy, an empty proxy on the same server as the Web service, and an empty proxy on a separate node (Node 2). These results show the minimum impact from using a Web service proxy for monitoring, as well as the comparative impact of the proxy based on its location.

The results in Figure 3 show that moving an (empty) proxy from being co-located with the service to another node increases the response time by approximately 8%. These results also show that direct invocation of the Web service was 8% slower than invoking the Web service through a

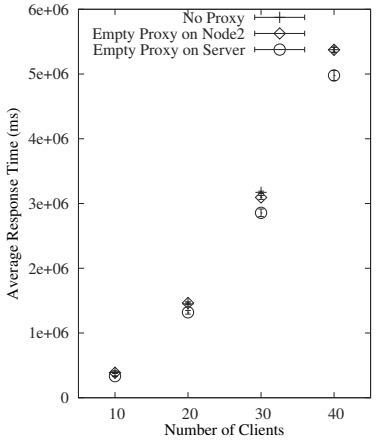

Figure 3: Effect of proxy-based interceptor on response time.

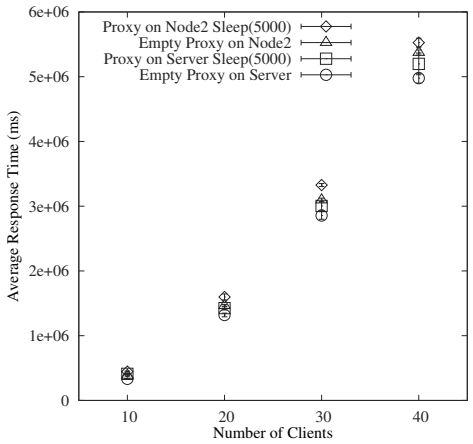

Figure 4: 5000ms blocking delay at proxy-based interceptor.

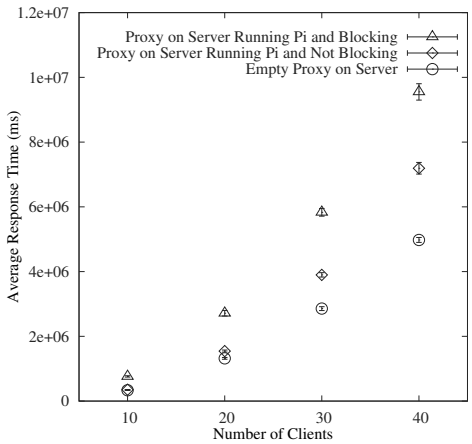

Figure 5: CPU/RAM intensive processing at proxy on server.

proxy on the server. This counter intuitive result may be explained by the fact that having an extra proxy on the server doubles the buffers available on the serving side, reducing the number of rejected connections. Similar results have been reported in [11].

B. Static delay at proxy

Figure 4 shows the results from testing with empty proxies, and with proxies with an inbuilt, blocking 5-second delay, on both the server and a dedicated node, respectively. This figure allows for a comparison of the impacts of a blocking proxy on a server and a dedicated node, as well as a comparison of the impacts of blocking proxies and empty proxies.

Proxies on the server and Node 2 with a blocking 5-second delay averaged about 14 seconds slower than an empty proxy. This shows that the impact of a delay is not a simple arithmetic function. For example, the response time of an empty proxy minus the response time of a proxy which sleeps for 5 seconds before returning the result is 14 seconds, not 5 seconds.

C. Processing on proxy located at server

Figure 5 shows the results from testing a proxy that performs a processing-intensive task on the server. The results are for blocking and non-blocking proxies, respectively. This figure allows for a comparison of the impacts of blocking and non-blocking proxies located on the server.

As expected, the blocking proxy caused approximately 40% greater response times than the non-blocking proxy. This difference in the impact on response time increased as load, due to the number of simultaneous clients, increased. The results also show that the impact of a non-blocking proxy on response time in this scenario was approximately 30%. Once again, this impact increased as the load on the service increased. The relative impact between the blocking

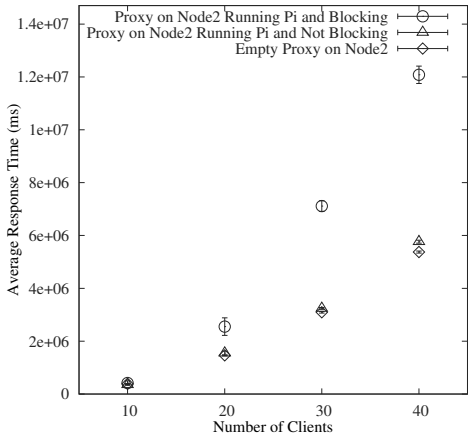

Figure 6: Blocking and non-blocking CPU/RAM intensive processing at proxy on dedicated node.

and non-blocking proxy has reduced and almost stabilised at around 20 threads. This is to be expected, since once the local resources are saturated due to high load, the effect of non-blocking processing at the proxy is diminished.

D. Processing on proxy located at Node 2

Figure 6 shows the results from testing a proxy that is performing a processing-intensive task on a second, dedicated node (Node 2). Similar to the previous section, the results are for configurations of the proxy as blocking and non-blocking, respectively.

The blocking proxy caused approximately 40% greater response times compared to the non-blocking proxy. Once again, this difference in the impact on response time increased as the load (number of simultaneous threads) increased. The results in Figure 6 also show that the impact of a non-blocking proxy on response time in this scenario

was approximately 5%.

Figure 6 also shows the impact of a blocking proxy on a second, dedicated node rising sharply away from the impact of a non-blocking proxy. In this case, the difference is exaggerated at 40 clients due to the CPUs on the proxy host becoming saturated, however the trend is detectable from 10 to 30 clients already.

E. Eavesdropping

Figure 7 shows the results from testing the effect of an eavesdropping, software-level monitoring on the system, configured with and without a dedicated proxy server. The results are for each scenario run with 10 threads.

As can be seen in Figure 7, eavesdropping on the system has no measurable negative effect on response time. As such, these results show that it is *possible* to eavesdrop on a Web service without negatively impacting the response time of that Web service.

IV. Discussion

By analysing the results from monitoring using eavesdropping, it can be seen that it is possible to eavesdrop on a Web service without significantly affecting the response time of that Web service. Conversely, by analysing the results for monitoring using a proxy, it can be seen that monitoring using a proxy Web service which performs some computation, can have a measurable negative impact on response times.

However, this eavesdropping method of monitoring is limited. Firstly, the eavesdropper may not understand captured traffic if that traffic is encrypted. Secondly, users must have access to the systems hosting the proxies and Web service, respectively, in order to take measurements. For example, the user must be able to install and administer the application to be tested. Otherwise, our results show that proxy-based intercepting monitors should often only be used when they are required to perform a blocking function.

The results for testing the effect of a CPU and RAM intensive proxy show that the impact from a proxy performing work (in this case computing π) may be reduced by over 20% by moving the proxy from the server hosting the Web service to another node.

Typical usage for a proxy would be for logging messages or duplicating/redirecting messages based on content. The proxies in our scenario had a heavier processing load than would be expected of proxies performing simple logging/trafficking actions, since they were computing π. The proxies were designed this way so that their processing was roughly equivalent to that performed by the Web service. This equivalent processing load is realistic in many scenarios, where the Web service is simply performing some database operation or executing a remote process.

Apart from the cost of a proxy-based monitor compared to an eavesdropping monitor, the cost of a blocking monitor

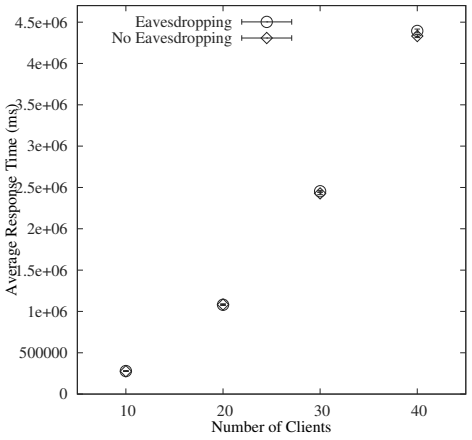

Figure 7: Eavesdropping vs. No Eavesdropping

varied significantly, compared to the cost of a non-blocking monitor. The results show that performing a blocking function at a proxy-based monitor was approximately 40% slower than performing a non-blocking function. This is a significant result, and demonstrates that a small design choice in a Web service monitor can have a large impact on performance. This impact of 40% on response time occurred for a proxy on both the server hosting the Web service and a dedicated proxy server, respectively. This demonstrates that the impact cannot be reduced by simply relocating the proxy to a dedicated server.

Processing load is an important factor for designers of Web services systems to consider when designing systems for meeting service-level agreements. Designers must trade off between the response times provided and the number of servers required.

For each of the sets of results provided in Section III, the response time is given as an average over all numbers of simultaneous clients. For example, the response time for 10 simultaneous clients is directly comparable to the response time for 20 simultaneous clients, since each of these times is an average over all client invocations. Knowing this, it can be clearly seen that as the number of simultaneous clients goes up, so does the average response time. This effect occurred in all test scenarios that were executed.

The results presented were based on tests that stressed the processors of each node in the system. It is important to note that other benchmarks may stress other aspects of a system, such as the network, which may yield different results.

V. Related Work

Table II presents a summary of a survey of distributed software monitoring techniques. For each technique, we have classified the types of monitors used into our monitoring taxonomy (Figure 1). We have identified those systems that used proxy-based monitoring when eavesdropper-based

monitoring would have sufficed ("bad proxy"). We have also identified those papers that mention the performance impact of their monitoring solution ("overhead"). Only two of the 37 papers provided a quantified impact; of "up to 40%" due to probing, and "1–7%" due to eavesdropping. Three papers stated the impact as "negligible" due to use of eavesdropping or existing proxies. These results show that, in general, designers of distributed monitoring systems have placed little importance on performance impacts. We have provided more detailed analysis of some of the papers presented in Table II. These papers present work explicitly designed for monitoring of Web services.

Panahi *et al.* [7] presented a middleware for management and monitoring of Web services. This is the only work that presents a framework for monitoring that includes a detailed measurement of the performance overhead. The authors have identified the need for the management system to be efficient enough to manage and monitor large numbers of services without affecting the execution of the underlying business process. The system has been prototyped as a Mule ESB extension. This prototype has been used to compare the performance impact of monitoring. The performance impact determined was between 1–7% for local agents (a type of proxy), and negligible for the service bus (since it is a passive listener).

Both da Cruz *et al.* and Raimondi *et al.* have presented methods for monitoring using SOAP intermediaries [5], [8]. They have prototyped their respective proposed frameworks and claim insignificant or negligible overheads for performance. However, details of these measurements were not described. For example, the impact of monitoring using existing proxies will be lower than the impact of monitoring by creating new proxies. The impact will also vary depending on load and proxy actions. Raimondi *et al.* [8] presented a method for the verification of service level agreements at run-time using *timed automata*. Monitoring is based on the interception and analysis of SOAP messages by SOAP intermediaries. The performance of the verification algorithm has been considered, and the authors have prototyped the system and reported no "significant overheads" [8]. This is consistent with our results for performing monitoring using non-blocking proxies. da Cruz *et al.* [5] presented a method for measuring Web service usage, based on intercepting SOAP messages and analyzing these messages by SOAP intermediaries. The system performance has been considered, and a prototype has been tested. The performance overhead of the prototype was reported to be "negligible", compared to the network latency of target systems [5].

Methods or frameworks capable of monitoring Web services have been presented that discuss the performance or overhead of monitoring [2], [12]–[14]. Although each of these discusses the performance impact of their respective monitoring and management systems, none provide information on the performance overhead of their proposed solution.

Baresi and Guinea [2] discussed the performance impact of monitoring, and the solution presented allows for the management of the performance impact of monitoring at run-time. However, it does not quantify the performance overhead of the monitoring framework presented.

Ranganathan and Dan [13] and Ludwig *et al.* [12] presented solutions aimed at guaranteeing the quality of service provided for a Web service by allocation of resources to Web services, and systematically accepting Web service requests based on the ability to meet service-level agreements, with consideration for current and expected future load on the Web services. Once again, neither work provides a measurement of the overhead of their proposed monitoring system.

Keller and Ludwig [14] presented a framework for monitoring SLAs in web services-based systems with dynamic business agreements. The authors considered the impact of monitoring on the performance of the monitored Web services, but did not provide a measurement of the overhead of their proposed system.

Ezenwoye and Sadjadi [15] presented an approach for increasing the fault tolerance of BPEL processes, which relies on the monitoring of web service interactions. Similarly, Benbernou *et al.* [16] presented a method for monitoring a web service environment at run-time in order to detect privacy violations, where monitoring is based on analysis of event logs. However, neither of them discussed the performance overhead of their proposed frameworks.

In summary, there are various solutions for monitoring Web services systems. Most of these solutions are targeted towards minimizing the performance impact of monitoring and or managing a Web services system. Only two of the 37 papers reviewed for this work provided performance measures. There is no comparison between techniques in terms of performance and no in-depth discussion of the performance impact of monitoring in general. In contrast, the empirical study we have reported in this paper provides a basis for the understanding of the performance impact of different types of Web service monitors.

VI. Conclusions and Future Work

We have identified no work that quantifies the impact of various types of Web services monitoring systems on the quality of service provided by those Web services. In order to fill this gap, we have conducted a series of experiments in order to quantify and assess the impact of monitoring on Web services in typical Web services scenarios. These tests provided three key results: firstly, it is possible to eavesdrop on a Web service with no measurable negative impact on response time. Secondly, a blocking monitor may significantly increase the response time of a Web service. Thirdly, the location of the monitor has an impact on the response time of the Web service being monitored.

The tests also demonstrated that a Web service monitoring system can accumulate significant costs in the form of a

Table II: Survey of Monitoring Techniques.

Source	Technique	Bad Proxy?	Overhead	Description
[16]	Probe	No	n/a	Monitoring event logs for evidence of privacy violations
[13]	Probe	No	n/a	QoS Guarantee via resource allocation and monitoring
[17]	Probe	n/a	n/a	WS Robustness testing
[18], [19]	Probe	n/a	n/a	Automated test case generation and distributed testing of WS
[20]	Probe	n/a	n/a	Classification of monitors for runtime testing
[21]	Probe	n/a	n/a	Generation of test cases for web service QoS
[22]	Probe	n/a	n/a	Distributed performance testing
[23]	Probe	n/a	n/a	Automatic conformance testing of WS
[24]	Probe	n/a	n/a	Automatic testing of WS compositions
[15]	Proxy	No	n/a	QoS of BPEL processes via interaction monitoring and dynamic service replacement
[25]	Proxy	No	n/a	Adaptation of WS for QoS of WS and Composition at runtime using AOP
[8]	Proxy	No	negligible	Verifying QoS of WS using timed automata
[26]	Proxy	No	n/a	Management layer for WS based on AOP
[27]	Proxy	No	n/a	Monitoring of run-time interactions of WS
[28]	Proxy	No	n/a	Creation and monitoring of policies for WS transactions
[2]	Proxy	No	n/a	Dynamic monitoring of BPEL processes using SOAP interception
[29], [30]	Proxy	No	n/a	Association and monitoring of assertions on business processes
[31]	Proxy	Yes	n/a	Distributed WS SLA monitoring via SOAP Proxies
[5]	Proxy	Yes	negligible	SOAP-Proxy based monitoring using logs
[6]	Eavesdrop	n/a	negligible	Verification of WS interaction patterns
[32]	Eavesdrop	n/a	n/a	Customer-side QoS monitoring
[33], [34]	Eavesdrop	n/a	n/a	Monitoring of WS (BPEL) compositions
[35]	Eavesdrop	n/a	n/a	Generation and use of monitors for monitoring requirements in WS systems
[36]	Eavesdrop	n/a	n/a	Monitoring WS BPEL processes for requirements compliance
[37]	Probe, Eavesdrop	n/a	n/a	P2P service recovery framework
[9]	Probe, Eavesdrop	n/a	Up to 40%	Method for monitoring distributed systems using queries
[38]	Probe, Proxy	Yes	n/a	Use of AOP for measuring WS QoS
[39]	Probe, Proxy	No	n/a	QoS-Aware WS selection and monitoring
[7]	Eavesdrop, Proxy	No	1-7%	WS Monitoring Middleware
[40]	Eavesdrop,Proxy	Yes	n/a	Method for formal modelling and verification of WS behaviours using DEC
[14]	All	No	n/a	Monitoring for dynamic business agreements
[12]	All	No	n/a	QoS Guarantee via resource allocation and monitoring
[41]	All	Yes	n/a	P2P based grid monitoring for QoS
[42]	Generic Framework	n/a	n/a	General system for monitoring electronic services
[43]	Generic Framework	n/a	n/a	automatic verification and analysis of asynchronous interaction patterns
[44]	Generic Framework	n/a	n/a	Method for predicting and monitoring QoS of a workflow
[45]	Generic Framework	n/a	n/a	Adaptive system for optimal web service composition and execution

reduction in quality of service provided by a Web service depending on the implemented monitoring methods (type and location).

With these results as the basis, we plan to investigate strategies to optimise a Web service monitoring system at run-time. We will consider balancing the cost of monitoring as discovered in this study against a model for benefits of monitoring using an optimisation algorithm. A generic model will be created that will allow for the measurement of qualities of service other than response time.

ACKNOWLEDGEMENT

We would like to thank the anonymous reviewers for the valuable comments on the draft manuscript. This work is supported by the Australian Research Council under grant LP0775188 in collaboration with CA Labs.

REFERENCES

[1] M. Papazoglou and W. Heuvel, "Service oriented architectures: Approaches, technologies and research issues," *The*

Very Large DataBases Journal (VLDB'07), vol. 16, pp. 389–415, 2007.

[2] L. Baresi and S. Guinea, "Towards dynamic monitoring of ws-bpel processes," in *International Conference on Service Oriented Computing (ICSOC'05)*. Amsterdam, The Netherlands: Springer, December 2005, pp. 269–282.

[3] L. O'Brien, P. Merson, and L. Bass, "Quality attributes for service-oriented architectures," in *International Workshop on Systems Development in SOA Environments (SDSOA'07)*. Washington, USA: IEEE Computer Society, May 2007, pp. 3–10.

[4] J. O'Sullivan, D. Edmond, and A. ter Hofstede, "Formal description of non-functional service properties," Centre for Information Technology, Queensland University of Technology, Tech. Rep., 2005, http://tiny.cc/WsuZC (accessed 10 Sept 2009). [Online]. Available: http://tiny.cc/WsuZC

[5] S. da Cruz, M. Campos, P. Pires, and L. Campos, "Monitoring e-business web services usage through a log based architecture," in *International Conference on Web Services (ICWS'04)*. San Diego, USA: IEEE Computer Society, July 2004, pp. 61–69.

[6] Y. Gan, M. Chechik, S. Nejati, J. Bennett, B. O'Farrell, and J. Waterhouse, "Runtime monitoring of web service conversations," in *Conference of the Center for Advanced Studies on Collaborative Research (CASCON'07)*. New York, USA: ACM, April 2007, pp. 42–57. [Online]. Available: http://dx.doi.org/10.1145/1321211.1321217

[7] M. Panahi, K. Lin, Y. Zhang, S. Chang, J. Zhang, and L. Varela, "The llama middleware support for accountable service-oriented architecture," in *International Conference on Service Oriented Computing (ICSOC'08)*. Sydney, Australia: Springer, December 2008, pp. 180–194.

[8] F. Raimondi, J. Skene, W. Emmerich, and B. Wozna, "A methodology for online monitoring non-functional specification of web-services," in *Workshop on Property Verification for Software Components and Services (PROVECS'07) in the conference on Objects, Models, Components and Patterns (TOOLS'07)*. Zurich, Switzerland: Springer, June 2007, pp. 170–180.

[9] A. Singh, T. Roscoe, P. Maniatis, and P. Druschel, "Using queries for distributed monitoring and forensics," in *European Conference on Computer Systems (EUROSYS'06)*. Leuven, Belgium: ACM Press, April 2006, pp. 389 – 402.

[10] U. Lamping, R. Sharpe, and E. Warnicke, *Wireshark Manual*, NS Computer Software and Services P/L, 2008, http://tiny.cc/U4i01 (accessed 10 Sept 2009). [Online]. Available: http://tiny.cc/U4i01

[11] D. Bressler, "Zin and the art of web service management performance," Actional, Tech. Rep., 2004, http://tiny.cc/XPfMG (accessed 10 Sept 2009). [Online]. Available: http://tiny.cc/XPfMG

[12] H. Ludwig, A. Dan, and R. Kearney, "Cremona: An architecture and library for creation and monitoring of ws-agreements," in *International Conference on Service Oriented Computing (ICSOC'04)*. New York, USA: Springer, November 2004, pp. 65–74.

[13] K. Ranganathan and A. Dan, "Proactive management of service instance pools for meeting service level agreements," in *International Conference on Service Oriented Computing (ICSOC'05)*. Orlando, USA: Springer, July 2005, pp. 296–309.

[14] A. Keller and H. Ludwig, "Defining and monitoring service-level agreements for dynamic e-business," in *USENIX Conference on System Administration (LISA'02)*. Philadelphia, USA: ACM Press, November 2002, pp. 189–204.

[15] O. Ezenwoye and S. Sadjadi, "Enabling robustness in existing bpel processes," in *International Conference on Enterprise Information Systems (ICEIS'06)*. Paphos, Cyprus: Springer, May 2006, pp. 95–102.

[16] S. Benbernou, H. Meziane, and M. Hacid, "Run-time monitoring for privacy-agreement compliance," in *International Conference on Service Oriented Computing (ICSOC'07)*. Vienna, Austria: Springer, September 2007, pp. 353–364.

[17] M. Vieira, N. Laranjeiro, and H. Madeira, "Benchmarking the robustness of web services," in *International Symposium on Pacific Rim Dependable Computing (PRDC'07)*. Melbourne, Australia: IEEE Computer Society, December 2007, pp. 322–329.

[18] W.-T. Tsai, Y. Chen, R. Paul, H. Huang, X. Zhou, and X. Wei, "Adaptive testing, oracle generation, and test case ranking for web services," in *International Computer Software and Applications Conference (COMPSAC'05)*. Edinburgh, Scotland: IEEE Computer Society, July 2005, pp. 101–106.

[19] W.-T. Tsai, X. Wei, Y. Chen, and R. Paul, "A robust testing framework for verifying web services by completeness and consistency analysis," in *International Workshop on Service Oriented Systems Engineering (SOSE'05)*. Beijing, China: IEEE Computer Society, October 2005, pp. 159–166.

[20] D. Brenner, C. Atkinson, O. Hummel, and D. Stoll, "Strategies for the run-time testing of third party web services," in *International Conference on Service-Oriented Computing and Applications (SOCA'07)*. Vienna, Austria: IEEE Computer Society, September 2007, pp. 114–121.

[21] M. Di Penta, G. Canfora, G. Esposito, V. Mazza, and M. Bruno, "Search-based testing of service level agreements," in *Conference on Genetic and Evolutionary Computation (GECCO'07)*. New York, USA: ACM, July 2007, pp. 1090–1097.

[22] C. Dumitrescu, I. Raicu, M. Ripeanu, and I. Foster, "Diperf: An automated distributed performance testing framework," in *International Workshop on Grid Computing (GRID'04)*. Pittsburgh, USA: IEEE Computer Society, November 2004, pp. 289–296.

[23] R. Heckel and L. Mariani, "Automatic conformance testing of web services," in *Conference on Fundamental Approaches to Software Engineering (FASE'05)*. Edinburgh, Scotland: Springer, April 2005, pp. 34–48.

[24] H. Huang, W.-T. Tsai, R. Paul, and Y. Chen, "Automated model checking and testing for composite web services," in *International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*. Los Alamitos, USA: IEEE Computer Society, May 2005, pp. 300–307.

[25] N. Narendra, K. Ponnalagu, J. Krishnamurthy, and R. Ramkumar, "Run-time adaptation of non-functional properties of composite web services using aspect-oriented programming," in *International Conference on Service Oriented Computing (ICSOC'07)*. Vienna, Austria: Springer, September 2007, pp. 546–557. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74974-5_51

[26] B. Verheecke, M. A. Cibran, and V. Jonckers, "Aop for dynamic configuration and management of web services," in *International Conference on Web Services (ICWS'03)*, L.-J. Zhang, Ed. Las Vegas, USA: CSREA Press, June 2003, pp. 25–41.

[27] Z. Li, Y. Jin, and J. Han, "A runtime monitoring and validation framework for web service interactions," in *Australian Software Engineering Conference (ASWEC'06)*. Los Alamitos, USA: IEEE Computer Society, April 2006, pp. 70–79.

[28] S. Tai, T. Mikalsen, E. Wohlstadter, N. Desai, and I. Rouvellou, "Transaction policies for service-oriented computing," *Data Knowledge Engineering*, vol. 51, pp. 59–79, 2004.

[29] A. Lazovik, M. Aiello, and M. Papazoglou, "Associating assertions with business processes and monitoring their execution," in *International Conference on Service Oriented Computing (ICSOC'04)*. New York, USA: Springer, November 2004, pp. 94–104.

[30] A. Lazovik, M. Aiello, and M. Papazoglou, "Planning and monitoring the execution of web service requests," *International Journal on Digital Libraries (JODL'06)*, vol. 1, pp. 335–350, 2006.

[31] A. Sahai, V. Machiraju, M. Sayal, L. J. Jin, and F. Casati, "Automated sla monitoring for web services," in *International Workshop on Distributed Systems Operation and Management (DSOM'02)*. Venice, Italy: Springer-Verlag, October 2002, pp. 28–41.

[32] R. Hauck and H. Reiser, "Monitoring quality of service across organizational boundaries," in *Trends in Distributed Systems: Torwards a Universal Service Market, International IFIP/GI Working Conference (USM'00)*, C. Linnhoff-Popien and H.-G. Hegering, Eds. Munich, Germany: Springer, September 2000, pp. 124–137.

[33] K. Mahbub and G. Spanoudakis, "A framework for requirements monitoring of service based systems," in *International Conference on Service Oriented Computing (ICSOC'04)*. New York, USA: Springer, November 2004, pp. 84–93.

[34] K. Mahbub and G. Spanoudakis, "Run-time monitoring of requirements for systems composed of web-services: Initial implementation and evaluation experience," in *International Conference on Web Services (ICWS'05)*. Orlando, USA: IEEE Computer Society, July 2005, pp. 257– 265.

[35] W. N. Robinson, "Monitoring web service requirements," in *International Conference on Requirements Engineering (RE'03)*. California, USA: IEEE Computer Society, September 2003, pp. 65–74.

[36] G. Spanoudakis and K. Mahbub, "Non intrusive monitoring of service based systems," *International Journal of Cooperative Information Systems (IJCIS'06)*, vol. 15, pp. 325–358, 2006.

[37] J.-Y. Chen, Y.-J. Wang, and Y. Xiao, "Soa-based service recovery framework," in *International Conference on Web-Age Information Management (WAIM'08)*. Zhangjiajie Hunan, China: IEEE Computer Society, July 2008, pp. 629–635.

[38] F. Rosenberg, C. Platzer, and S. Dustdar, "Bootstrapping performance and dependability attributes of web services," in *International Conference on Web Services (ICWS'06)*. Salt Lake City, USA: IEEE Computer Society, September 2006, pp. 205–212. [Online]. Available: http://dx.doi.org/http://dx.doi.org/10.1109/ICWS.2006.39

[39] M. Tian, A. Gramm, H. Ritter, and J. Schiller, "Efficient selection and monitoring of qos-aware web services with the ws-qos framework," in *International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'04)*. Beijing, China: ACM Press, September 2004, pp. 152 – 158.

[40] M. Rouached and C. Godart, "Analysis of composite web services using logging facilities," in *International Conference on Service Oriented Computing (ICSOC'06)*, D. Georgakopoulos, N. Ritter, B. Benatallah, C. Zirpins, G. Feuerlicht, M. Schnherr, and H. R. M. Nezhad, Eds. Salt Lake City, USA: Springer, September 2006, pp. 74–85. [Online]. Available: http://dblp.uni-trier.de/db/conf/icsoc/icsoc2006w.html

[41] H.-L. Truong, R. Samborski, and T. Fahringer, "Towards a framework for monitoring and analyzing qos metrics of grid services," in *International Conference on e-Science and Grid Computing (E-SCIENCE'06)*. Amsterdam, Netherlands: IEEE Computer Society, December 2006, pp. 65–73.

[42] G. Piccinelli, W. Emmerich, S. L. Williams, and M. Stearns, "A model-driven architecture for electronic service management systems," in *International Conference on Service Oriented Computing (ICSOC'03)*. Trento, Italy: Springer, December 2003, pp. 241–255.

[43] T. Bultan, J. Su, and X. Fu, "Analyzing conversations of web services," *IEEE Internet Computing (IC'06)*, vol. 10, pp. 18–25, 2006.

[44] J. Cardoso, J. Miller, A. Sheth, and J. Arnold, "Modeling quality of service for workflows and web service processes," *Journal of Web Semantics*, vol. 1, pp. 281–308, 2002.

[45] G. Chafle, K. Dasgupta, A. Kumar, S. Mittal, and B. Srivastava, "Adaptation in web service composition and execution," in *International Conference on Web Services (ICWS'06)*. Salt Lake City, USA: IEEE Computer Society, September 2006, pp. 549–557.