REACTIVE MONITORING OF SERVICE LEVEL AGREEMENTS

Dalia Khader, Julian Padget
Department of Computer Science
University of Bath, United Kingdom
ddk20@bath.ac.uk
jap@bath.ac.uk

Martijn Warnier
Systems Engineering
Faculty of Technology, Policy and Management
Delft University of Technology
The Netherlands
m.e.warnier@tudelft.nl

Abstract Service Level Agreements require a monitoring system that checks that no party violates the agreement. Current monitoring techniques either have a high performance overhead or are not reliable enough. This paper proposes a new hybrid monitoring system that we call *reactive monitoring*. It tries to balance the disadvantages of established monitoring techniques, in particular online and offline monitoring. Online monitoring has a relatively high performance overhead and offline monitoring does not identify all possible violations.

Reactive monitoring combines online monitoring, which is used for reactively checking continuous SLA properties with a new *passive monitoring scheme*. This scheme is used for monitoring discrete SLA properties. It is based on cryptographic primitives that provide proof that either a certain stage in an interaction has been reached correctly with all participants in compliance of the service level agreements or that a violation has occurred. In the latter case the violating party can be identified.

A theoretical analysis shows that in the worst case scenario this new approach has the same overhead as online monitoring techniques and in most cases the overhead will be significantly lower.

Keywords: Service Level Agreements, Reactive Monitoring, Passive Monitoring, Multi Party Contract Signing Protocol, Aggregate Signatures, Violations

P. Wieder et al. (eds.), *Grids and Service-Oriented Architectures for Service Level Agreements,*
DOI 10.1007/978-1-4419-7320-7_2, © Springer Science+Business Media, LLC 2010

1. Introduction

Service Level Agreements (SLAs) form an essential part of distributed computing, in particular in environments such as grid, cloud and service oriented computing. A SLA represents an agreement between a client and a provider in the context of a particular service provision. SLAs can be between two (one-to-one) or more (one-to-many, or many-to-many) parties. A SLA typically consists of a number of Service Level Objectives (SLOs) that define Quality of Service (QoS) properties for the agreed upon service. The preceding negotiation and agreement of SLAs are outside the scope of this paper, but see for example [9, 14]for more on these subjects. These QoS properties need to be measurable and must be monitored during the provision of the service that has been agreed in the SLA.

Typically, an independent trusted third party (TTP) is used to monitor the agreement. Two approaches can be distinguished for monitoring SLAs. The first type is *online* monitoring [10–12]. This involves periodically testing whether the agreement terms have been met by all relevant parties. The monitoring interval can vary, depending on the agreement's SLOs, but in general it has to be quite small (of the order of seconds). A property such as network bandwidth, for example, has to be monitored continuously if one wants to ensure that the SLO is not violated. The other approach is *offline* monitoring. In this case all interactions are recorded, typically at the client site [6], and securely logged and stored by the monitor [4]. If a party to the agreed upon SLA thinks that the terms have been violated, the log is examined to establish whether a violation took place.

Both types of monitoring come at a cost. Online monitoring is hard to implement in an efficient way. It has a relatively high performance overhead and the monitoring system typically forms a bottleneck since all parties in all interactions contact it throughout. The disadvantage of offline monitoring is the need for storage and, more importantly, this type of monitoring cannot always prove with certainty that a violation has taken place[11]. If, for example, the network bandwidth at the client site drops, did this happen because the provider violated the SLA or because the client is under a denial of service attack? Some research, most notably by Jurca et. al. [10] has tried to extend monitoring with reputation based mechanisms in order to fix this problem. But reputation mechanisms have their own reliability problems [7].

This paper proposes a new monitoring technique that tries to balance the trade-offs of the monitoring approaches discussed above. In the worst case scenario the new hybrid approach has the same overhead as online monitoring techniques and in most cases the overhead will be significantly lower. The new technique depends on what we refer to as *passive monitoring*. Passive monitoring is an offline monitoring scheme that uses cryptographic primitives to provide proof that a certain stage of an interaction has been reached correctly, i.e., without any of the parties violating the SLA. The proofs are exchanged between the communicating parties without the help of a trusted third party. The proposed *reactive monitoring scheme* is a hybrid approach to monitoring. It combines online monitoring with the new (offline) *passive monitoring scheme.* In the case that a dispute arises an (online) monitor is contacted. At this point the parties either prove they have reached that stage correctly, in full compliance with the SLA, by providing the most recent cryptographic primitive they have. Or, alternatively, one of the parties is in violation, which can be proved from the cryptographic primitive they present. The protocol used in exchanging these primitives is called the service evidential protocol (SEP). In the case that no violation was proven the parties have the option to renegotiate their monitoring policy. At this point they can agree to use online monitoring for some fixed time period before switching back to the passive monitoring scheme.

2. Preliminaries

This section describes two preliminaries used as building blocks for the passive monitoring scheme.

2.1 Contract Signing Protocols

A contract signing protocol (CSP) is a cryptographic protocol that allows two or more parties to exchange signatures on a contract such that no party receives a signed contract unless all of them do, achieving what literature refers to as fairness, as first proposed by Even [2]. The obvious solution to implement such a protocol is to utilize a trusted third party (TTP) that collects a digitally signed contract from all participants and redistributes or aborts. However, this solution is not ideal since the TTP becomes a performance bottleneck. Two solutions have been proposed in the literature in order to address this problem. The first is to eliminate the involvement of the TTP [3], where the general idea is to exchange signatures gradually. However, these solutions are nondeterministic, which in most cases would be a problem for the signatories and is expensive in terms of computation and communication. A second solution is to construct a CSP while minimizing interaction with the TTP. An optimistic protocol only depends on the TTP when there is a dispute. In other words the TTP is never contacted if all signing parties are behaving in compliance with the protocol [4, 13]. For this paper we assume the usage of a CSP to finalize a service level agreement and to support SEP. We leave the decision of which type of CSP to use for future studies, but there are several candidates in the literature [4, 8].

In the remainder of this paper we will not distinguish between a trusted third party and a monitor. All monitors are trusted third parties, for the sake of readability we only speak of monitors in the sequel.

2.2 Aggregate Signatures

An aggregate signature is a digital signature scheme that supports aggregation [1] and is one of the main building blocks for a SEP. Given n signatures for n distinct messages from n different users, it is possible to merge them into one single signature. The following are the algorithms:

- Setup: Each user i has a public key pk_i and a secret key sk_i.
- Sign: Using the message M and secret key sk_i as input create a signature $\sigma_i = S(M, sk_i)$
- Verify: Using the public key pk_i, the message M and the signature σ_i as input verify a signature σ_i. $V(\sigma_i, M, pk_i) = \{accept, reject\}$. This tells us whether σ_i was in fact created by user i.
- Aggregate Signature: Having signature $\sigma_1, ..., \sigma_n$ as inputs create one short signature. $\sigma = A(\sigma_1, ..., \sigma_n)$.
- Verify Aggregation: Having several public keys $pk_1, ..., pk_n$, several messages $M_1, ..., M_n$ and one aggregation σ verify the aggregation: $R(\sigma, pk_1, ..., pk_n, M_1, ..., M_n) = \{accept, reject\}$. Which tells us if all signatures been created by their corresponding users.

3. Service Evidential Protocol

The service evidential protocol (SEP) is a protocol that allows for the collection of evidence of SLA compliant behavior of the communicating parties over the period of their interaction. The general idea is to minimize the usage of the monitoring system for SLA agreement. We start from a presumption of good intentions by all parties to the SLA. However, if at any point one of the parties suspects any of the other parties of non-compliance, it calls on the monitor who is, by definition, trusted by all. At this point one of two situations can occur: (i) one of the parties is in violation of the SLA. The monitor identifies the violating party by inspecting the cryptographic signatures, aborts the service and penalizes the offender. Appropriate penalties can be negotiated and be part of the SLA [11]. Or, (ii) all parties are in compliance with the SLA and can prove this by presenting the appropriate cryptographic signatures. In this case, the parties can either renegotiate a new, possibly online, monitoring scheme or can continue using passive monitoring.

So in the most optimistic outcome the monitor never gets involved. If a dispute occurs, the violating party can be identified and penalized. In addition, the parties can renegotiate the monitoring policy and revert to conventional monitoring (i.e., offline or online depending on the service and SLA). This can be done easily as long as the service provided is discrete and state based. Continuous QoS-like properties cannot be monitored in this manner. In Section 5 we discuss how passive and online monitoring can be combined in a hybrid approach, reactive monitoring, that can monitor both discrete and continuous properties.

The general idea of the protocol is as follows:

1. The service provider starts by sending the service encrypted with the *monitor's* public key to the client
2. The response of the client is a signature on the received ciphertext
3. On receiving the signed ciphertext, the service provider responds with an encrypted service to the client

The client can verify that the receipt he has given out was on the service he requested and the service provider has a signature of the client that provides non-repudiation: the client cannot deny ordering the service.

We adopt the standard naming convention in the cryptographic literature and refer to the service provider as Alice and the service client as Bob.

Consider the following example:

SCENARIO 1 *Alice provides memory storage. Bob is interested in using Alice's service for a week. Alice and Bob sign a contract that states the SLA. The contract indicates that to obtain memory storage Bob will need a password that expires every day. The states of the interaction can be divided to seven stages in which Bob asks every day for a new password from Alice. In the SLA agreement both parties agreed on hiring Matilda as a passive monitor.*

One can assume that the password is the service provided, and we notate it as M to Bob. Alice has the key pair (pk_a, sk_a), Bob has (pk_b, sk_b), and Matilda has (pk_m, sk_m). We refer to the encryption algorithm as E, the decryption algorithm as D and the signing algorithm as S throughout the paper. The steps are as follows:

1. Alice sends Bob the ciphertext $C_1 = E(M, pk_m)$ and $\sigma_a = S(C_1, sk_a)$. If Bob does not respond then Alice has not revealed any information because the service is encrypted with Matilda's key.

2.a Bob verifies σ_a then replies with sending Alice $\sigma_b = S(C_1, sk_b)$. σ_b represents a receipt in the context of SEP (success).

2.b If Bob does not get a reply (σ_a), he can contact Matilda and she can recover the service M (from the previous step). Matilda will at this point give Bob the encrypted service $C_2 = E(M, pk_b)$, which she can construct from message C_1. If Matilda establishes that M is not the service agreed upon in the CSP, she signs an abort message to Bob and the algorithm halts (fail).

3. Alice on receiving the receipt σ_b can verify it and send Bob the encrypted service $C_2 = E(M, pk_b)$.

Figure 1 is a diagram presenting the above protocol.

Alice		Bob
$C_1 = E(M, pk_m)$ $\sigma_a = S(C_1, sk_a)$		
	$\xrightarrow{C_1, \sigma_a}$	
		$V(\sigma_a, C_1, pk_a) = \{accept, reject\}$ $\sigma_b = S(C_1, sk_b)$
	$\xleftarrow{\sigma_b}$	
$V(\sigma_b, C_1, pk_b) = \{accept, reject\}$ $C_2 = E(M, pk_b)$		
	$\xrightarrow{C_2}$	
		$D(C_2, sk_b) = M$

Figure 1. Optimistic Approach - No monitor needed

4. Passive Monitoring Scheme

If two or more parties to a SLA decide to use a passive monitoring scheme (PMS), they should specify the monitoring party to use in case of dispute. Once the negotiation ends a contract signing protocol takes place. This serves to finalize the agreement and helps identify violating parties in case of disputes.

We assume that the service level agreement is a set of states that occur one after another. For example, if we are talking about memory storage as the type of service, a state would be the amount of memory reserved for a certain party over a certain time period. Each time the interaction reaches a new state the parties run SEP and exchange receipts. Each entity aggregates the new receipt with the old ones. If a dispute arises the monitor asks all parties to provide the latest aggregated signature they have calculated and the latest encrypted service.

The aggregate signature will refer to the state the party has reached. If all parties have reached the same state then the monitor concludes that all parties are acting in compliance with the SLA and decrypt the ciphertexts received and distribute them. The monitoring can then continue in passive manner. Figure 1 presents the interaction between Alice and Bob when the monitor Matilda is not contacted while Figure 2 demonstrates the interactions in case the monitor is needed. Note that A_a is the aggregation by Alice throughout the interaction, A_b is the signature aggregation by Bob during the interaction and $Chk(CSP)$ refers to checking the contract signing protocol to see what services should have been provided at each state of the interaction.

Examining Figure 2 in detail, we see that Bob claims he has sent a receipt to Alice but has not got the service he requested. He sends Matilda the ciphertext and signature he got from Alice (C_1, σ_a). Furthermore, he sends Matilda an aggregate signature of his A_b indicating the state he has reached with Alice in the interaction. He also sends a receipt copy σ_b to Matilda as proof of

Alice	Matilda	Bob
		$\xleftarrow{C_1,\sigma_a,A_b,\sigma_b}$
$\xrightarrow{A_a}$		
$\xleftarrow{abort,Penalize}$	if$(V(\sigma_a, C_1, pk_a) = reject$	$\xrightarrow{abort}$
	Else	
	$D(C_1, sk_m) = \bar{M}$	
$\xleftarrow{abort,Penalize}$	$Chk(CSP)$ and if $(\bar{M} \neq M)$	$\xrightarrow{abort}$
	Else	
$\xleftarrow{abort,Penalize}$	$Chk(CSP)$ and if $(R(A_a, pk_1, ..., pk_n, M_1, ..., M_n) = reject)$	$\xrightarrow{abort}$
	Else	
$\xleftarrow{abort}$	$Chk(CSP)$ and if $(R(A_b, pk_1, ..., pk_n, M_1, ..., M_n) = reject)$	$\xrightarrow{abort,Penalize}$
	Else	
$\xleftarrow{abort}$	if $(V(\sigma_b, C_1, pk_b))$	$\xrightarrow{abort,Penalize}$
	Else	
$\xleftarrow{\sigma_b}$	$E(M, pk_b) = C_2$	$\xrightarrow{C_2}$
$V(\sigma_b, C_1, pk_b)$=$\{accept, reject\}$		$D(C_2, sk_b) = M$

Figure 2. Monitor mediation required

good intention. Matilda asks Alice to provide her with an aggregate signature too. Matilda can verify the signatures she got from both parties and decrypt the message she got from Bob. She compares the service M, the state of interaction, i.e., comparing aggregate signatures, and the receipts with the SLA contract. If everything seems compatible with the contract, Matilda can assume an unreliable connection between Alice and Bob. She then forwards the service encrypted to Bob and Bob's receipt is sent to Alice.

The proposed passive monitoring scheme uses asymmetric (public key) encryption for all operations. This is expensive and not necessary. A typical optimization would be to use public key encryption to establish the receipts together with a session key and then use this key with a (cheaper) *symmetric* encryption scheme or the actual (encrypted) service.

5. Reactive Monitoring Scheme

The passive monitoring scheme introduced in the previous section is not able to monitor continuous QoS-like properties such as network bandwidth or processing power. Additionally, we observe that some QoS properties, in particular security, are very hard to monitor. None of the existing monitoring techniques, including passive or reactive monitoring, is capable of dealing with these. However, for continuous properties we propose a *reactive* approach. At the moment that one of the parties to an SLA suspects that a continuous property is violated it can contact an online monitor. Using continuous monitoring it tries to establish if a violation has taken place. Since the offending party will not be notified until after the inspection is performed, there is a reasonable probability of detecting most violations in this way. Figure 3 illustrates the complete reactive monitoring scheme.

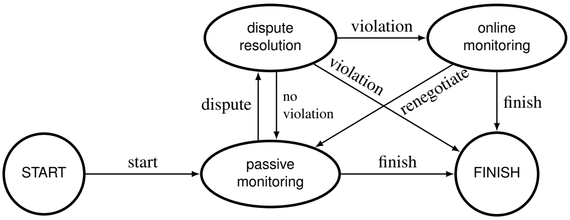

Figure 3. State diagram of reactive monitoring scheme

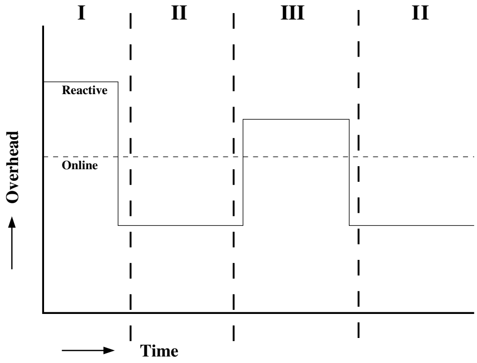

Figure 4. Expected overhead of online and reactive monitoring

The reactive monitoring scheme proposed in this paper tries to balance the disadvantages of both online and offline monitoring. The advantage with respect to offline monitoring should be clear: in contrast to offline monitoring, reactive monitoring detects all occurrences of violations and can always identify the offending party.

The advantage with regards to online monitoring is perhaps less clear. At first glance reactive monitoring seems to be more expensive since it uses some expensive cryptographic operations and also uses online monitoring as a subpart of its process. However, we argue that *on average* reactive monitoring has a smaller performance overhead compared to online monitoring. Figure 4 displays a hypothetical view of the expected overhead over the time period of a typical SLA. However, we have not represented offline monitoring costs on the graph, because it is not clear to us to where to attribute the costs or how large they should be at different times.

Three different phases can be distinguished:

- Phase **I**: This is the initialization phase of the reactive monitoring scheme. Several cryptographic operations have to be performed to use the passive monitoring scheme, and its overhead will thus be higher than online monitoring at this stage.

- Phase **II**: Under normal circumstances online monitoring will on average be more expensive than reactive monitoring. The reactive monitoring scheme does not continuously inspect the provided service, but will only occasionally forward and sign some service requests.
- Phase **III**: If a (possible) violation is signalled by one of the parties the monitor either (i) checks the credentials, for discrete SLA properties, to determine if a violation has occurred and to identify the offending party (if any), or (ii) an online monitor is used to determine retrospectively if a continuous property has been violated. In this phase the reactive monitoring scheme will again be more expensive than online monitoring.

We argue that on average phase **II** will be much more common then either phase **I** or **III**. This will in particular be the case if the number of violations is relatively low. From this we conclude that, when measured over a longer time period, the cumulative cost of reactive monitoring will be lower than the costs of online monitoring.

Obviously, this argument is only a first step towards a full analysis of the difference in performance overhead between reactive and online monitoring. For one thing, we have not defined what overhead exactly means: number of used messages, cpu or memory usage or something else? We do believe that reactive monitoring can be beneficial under typical situations and deserves further research. Our next step is to implement the passive monitoring scheme and compare its performance (overhead) with other approaches in different circumstances, possibly using some of the ALIVE (see `http://www.ist-alive.eu`) scenarios as testbeds.

In summary, the reactive monitoring scheme is a hybrid approach that combines passive monitoring, as introduced in Section 4, with online monitoring. This combination should provide reliable monitoring with an overhead that is smaller than with conventional online monitoring.

6. Conclusions

This paper introduces *reactive monitoring*: a monitoring paradigm that combines classical online monitoring with a new passive monitoring scheme based on aggregate contract signing protocols. A theoretical analysis of the new monitoring scheme shows that in typical circumstances reactive monitoring has a lower overhead than online monitoring. The next step is to implement the reactive monitoring scheme and show through empirical experiments that the overhead is indeed lower.

Acknowledgments

This work is partially supported through the ALIVE project (FP7-IST-215890)

References

[1] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. Journal of Cryptology, 17(4):297–319, 2004.

[2] S. Even. Protocol for signing contracts. In CRYPTO'81, pages 148–153, 1981.

[3] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. Commun. ACM, 28(6):637–647, 1985.

[4] J. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In CRYPTO'99, LNCS, Vol. 1666, page 787, Springer, 1999.

[5] L. Gymnopoulos, S. Dritsas, S. Gritzalis, and C. Lambrinoudakis. GRID security review. LNCS, pages 100–111, 2003.

[6] R. Jurca, B. Faltings, and W. Binder. Reliable QoS monitoring based on client feedback. In Proceedings of the 16th international conference on World Wide Web, pages 1003–1012, ACM Press New York, NY, USA, 2007.

[7] R. Kerr and R. Cohen. Smart cheaters do prosper: Defeating trust and reputation systems. In Proceedings of the Eigth International Conference on Autonomous Agents and Multi-Agent Systems, 2009.

[8] A. Mukhamedov and M. Ryan. Improved multi-party contract signing. In Financial Cryptography and Data Security, LNCS, Vol. 4886, pages 179–191, Springer, 2007.

[9] A. Pichot, P. Wieder, O. Wäldrich, and W. Ziegler. Dynamic SLA-negotiation based on WS-Agreement. Technical Report TR-0082, Institute on Resource Management and Scheduling, CoreGRID - Network of Excellence, June 2007.

[10] F. Raimondi, J. Skene, and W. Emmerich. Efficient online monitoring of web-service SLAs. In SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, pages 170–180, New York, USA, 2008.

[11] O. Rana, M. Warnier, T. B. Quillinan, and F. M. T. Brazier Monitoring and Reputation Mechanisms for Service Level Agreements. In Proceedings of the 5th International Workshop on Grid Economics and Business Models (GenCon), Las Palmas, Gran Canaria, Spain, Springer, August 2008.

[12] A. Sahai, S. Graupner, V. Machiraju, and A. van Moorsel. Specifying and monitoring guarantees in commercial grids through SLA. In Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, pages 292–299, 2003.

[13] B. Waidner and M. Waidner. Round-optimal and abuse-free optimistic multi-party contract signing. In Automata Languages and Programming, LNCS, Vol. 1853, pages 524–535, Springer, 2000.

[14] W. Ziegler, P. Wieder, and D. Battre. Extending WS-Agreement for dynamic negotiation of Service Level Agreements. Technical Report TR-0172, Institute on Resource Management and Scheduling, CoreGRID - Network of Excellence, August 2008.