Self-Adaptive Service Level Agreement Monitoring in Cloud Environments*

Kassidy P. Clark† Martijn Warnier Frances M.T. Brazier

Delft University of Technology, Faculty of Technology, Policy and Management, System Engineering Section, Jaffalaan 5, 2628BX Delft, The Netherlands

[K.P.Clark, M.E.Warnier, F.M.Brazier] @tudelft.nl

Abstract

Cloud Service Providers offer access to hardware and software resources across the Internet. Cloud Service Providers and their clients negotiate Service Level Agreements (SLA) that specify terms and conditions of services. To ensure that an SLA is being honored by all parties the service is monitored. This paper proposes a monitoring approach that automatically adapts the monitoring process in real-time, based on user-specified policies. The monitor can both adapt the frequency of measurements and monitoring mode: active and passive. Active mode uses an external, trusted monitoring service. Passive mode minimizes interaction with an external monitoring service, relying instead on a mutual commit protocol to ensure SLA compliance. Monitor adaptation is determined by levels of perceived risk, defined by a user's policy. The self-adaptive SLA monitoring framework is realized using agent technology and is implemented and tested in the AgentScape middleware.

Keywords: Monitoring, Service Level Agreement, Distributed, Agents, AgentScape Middleware, Cloud, Dynamic

*This paper is an extension and revision of a paper previously presented at the 2011 International Conference on Adaptive and Intelligent Systems [1].

†Corresponding author. E-mail: k.p.clark@tudelft.nl, Tel.: +31 15 278 73 59

1 Introduction

Cloud Service Providers (CSP) offer access to hardware and software resources across the Internet. Cloud [2] services range from raw compute resources, such as storage or processing power, to software services, such as databases or other applications. CSPs and their consumers agree upon terms and conditions of a service and specify this in a Service Level Agreement (SLA). Terms and conditions of a service may include price, Quality of Service (QoS) guarantees and penalties that apply if an agreement is violated by one of the parties. For instance, a CSP may guarantee 99% uptime. If this guarantee is not met, a CSP may issue free credits that can be redeemed for services, such as additional computing power.

An SLA violation may also be the responsibility of the consumer. For instance, a consumer is running software on a CSP's hardware and the system crashes. If it can be shown that the crash was the result of faulty hardware, the CSP is responsible. However, if it can be shown that the crash was the result of the software, the consumer is responsible. Other consumer violations may include late payment, overuse of services (e.g. fair usage policies) or incorrect use of services (e.g. using the service for illegal purposes).

Services are often monitored to detect whether an SLA is violated and, if possible, to identify the responsible party. Traditional SLA monitoring involves periodically testing specific service metrics and verifying if results comply with the terms of the SLA. For instance, an SLA specifies an upper limit for network latency. To test this, a monitoring service is used to periodically measure response time of communication between the two parties involved. If response time exceeds an upper limit, a violation occurs. This monitoring technique is referred to throughout this paper as *active* monitoring. To ensure that results are impartial, a Trusted Third Party (TTP) is typically utilized to perform these measurements, mediate conflicts and enforce violation penalties.

Active monitoring has several drawbacks, including additional overhead costs and inherent dependence on a TTP. To address these issues, this paper presents a self-adaptive approach to SLA monitoring in the Cloud (extending

earlier work [1]). Using this approach, parties are able to dynamically adapt the process by which SLAs are monitored. To reduce overhead, frequency of measurements can be adjusted. To reduce dependence on a TTP, a mutual commit protocol, the Service Evidential Protocol (SEP) is used. SEP requires only minimal interaction with a TTP. This monitoring technique is referred to throughout the paper as *passive* monitoring.

The self-adaptive monitoring approach is designed around the notion that the level of monitoring assurance required for a service is a reflection of *perceived risk*. The perceived risk of a service is a product of many factors, including trust between the parties. As these factors, change over the lifetime of a service, so does the level of perceived risk of that service. Parties adapt the monitoring policy based on the perceived level of risk. The monitoring policy specifies frequency of measurements and monitoring mode: active or passive. This self-adaptive monitoring approach often results in lower overhead and less dependence on the TTP when possible, and high monitoring assurance when required.

Monitoring SLAs in distributed environments presents several additional challenges, including scalability. A monitoring solution must be able to scale across multiple machines and handle multiple users. Another challenge is handling multiple administrative domains. A monitoring solution must be able to access monitoring metrics from multiple users (e.g. a consumer and a provider). This may include sensitive data that users wish to keep private from their competitors.

The monitoring approach described in this paper makes use of a distributed agent platform to be scalable and to access sensitive monitoring data while respecting privacy concerns. Consumers and providers are modeled as agents with private data. TTPs are modeled as agents with special access to monitoring data.

This paper is organized as follows: Section 2 discusses some of background concepts, including Service Level Agreements and service monitoring. Section 3 details the Service Evidential Protocol (SEP). Section 4 describes how active and passive modes are combined. Section 5 elaborates on how violations are handled by conflict mediation. Section 6 explains how self-adaptive

monitoring can be deployed in a Cloud environment. Section 7 describes the implementation of a self-adaptive monitoring framework in the AgentScape middleware and presents results of experimentation. Section 8 relates this paper to other works on monitoring SLAs in the Cloud. Finally, Section 9 discusses the results of the paper and proposes future research.

2 Monitoring Service Level Agreements

Cloud services are often monitored to detect SLA violations and (optionally) to identify the responsible party. Cloud services may be monitored by both their consumers and providers. Consumers monitor services to ascertain whether or not the services (their business relies on) are being delivered as agreed. Consumer measurements are typically limited to external metrics, such as response time and uptime (e.g. Is a particular website online? How long does it take to open that website?). In contrast, Cloud Service Providers (CSP) have direct access to system internals and can perform more accurate and extensive measurements than the average consumer.

2.1 Service Level Agreements

Service Level Agreements (SLA) are agreements between multiple parties that specify terms of service. They involve at least one service provider and at least one service consumer and specify the services to be provided. These documents include Quality of Service (QoS) guarantees. QoS can be expressed as a set of (`name`, `value`) pairs where `name` refers to a Service Level Objective (SLO) and `value` represents the requested level of service. An SLO specifies the particular characteristics of the service to be measured, how measurements are performed and any actions that take place after measurement. For instance, an SLA for a Cloud infrastructure may contain the following pairs: (uptime, greater than 99%) and (memory size, greater than 16 GB). An expanded list of SLA terms for the Cloud can be found in [3, 4].

Furthermore, an SLA specifies the payments for compliance or penalties for violation of the agreement. Penalties include either monetary fines or

demerits to a party's reputation [5].

Traditionally, SLAs are written and signed between legal entities, representing each of the parties involved. In recent years, attention has been given to automating this process [6, 7]. For which, several specifications have been proposed to describe and negotiate SLAs, including WS-Agreement [8] from the Open Grid Forum (OGF).

2.2 Active Service Monitoring

A commonly used approach to monitoring is referred to as online, continuous or active monitoring. Active monitoring performs specific measurements at specified intervals. For instance, distributed computer systems often use "heartbeat" monitoring to detect if a node has failed [9]. Nodes are periodically contacted and asked to respond. Failure to respond indicates that a node has failed.

Active monitoring is also used to monitor SLAs [10, 11]. A service is monitored by periodically testing whether the terms of an SLA have been met by all parties. This may require measuring a single variable or a complex aggregation of variables. For instance, *'Host is reachable.'* may be measured by a single request/response action. In contrast, *'Host uptime is greater than 99%.'* is often measured by polling a host multiple times and calculating the average rate of success.

Monitoring is used to detect SLA violations when they occur and to identify the offending party, if possible. In some cases, no responsible party can be identified (e.g. force majeur). For instance, if a lightning strike disables the communication lines between a consumer and a provider, the consumer may incorrectly conclude that the provider has violated the SLA. Monitoring data can be used to show that the provider was not responsible for the violation.

An important aspect of monitoring is safeguarding objectivity of monitoring results. A party to an agreement may have an incentive to manipulate the results to his or her advantage. For instance, an SLA may stipulate that a consumer receives financial compensation if a specified service is un-

reachable. Regardless of the actual status of the service, that consumer may want to manipulate monitoring results to make it appear unreachable and therefore collect financial compensation. To prevent this situation from occurring, a Trusted Third Party (TTP) is used to perform monitoring measurements [12, 11]. A TTP is an independent party that can access all communication between the parties to the SLA. To prevent parties from manipulating the measurement results collected at their respective locations, a TTP install Trusted Monitoring Modules (TMM) at each party's location.

The use of TMMs allows parties to have more equal access to the same QoS metrics. For instance, a consumer may not allow a provider to access sensitive client data directly. However, a TMM allows a TTP to access this data in a secure way. Thus, the provider has 'indirect' access to the data via the TTP and will be notified if it reveals any SLA violations. Which TMMs are required to access which QoS metrics depends on a specific SLA.

One disadvantage of active monitoring is that processing and communication overhead, together with monitoring accuracy, directly depend on the measurement interval chosen. Measurement intervals can range from a few seconds to hourly or daily measurements, depending on the nature of an SLA. The shorter this interval, the more accurate and comprehensive the results. The longer this interval, the lower the overhead and possibly lower accuracy. Many optimizations can be done to lower the overhead, such as dynamically decreasing the frequency [13] or lowering the complexity [14] of measurements.

Another disadvantage of active monitoring is dependence on a TTP. In some cases, parties may not prefer to give external parties (i.e. a TTP) access to local resources and (sensitive) data. However, the TTP requires this access to perform accurate measurements. In other cases, dependence on a TTP can form an obstacle to scalability of a system. For instance, if a service provider relies on a single TTP and this TTP becomes overloaded or suffers a severe failure, the service provider is affected and cannot ensure monitoring for existing consumers or accept new consumers.

3 Passive Service Monitoring

The passive service monitoring technique described in this section is an alternative approach to SLA monitoring that uses cryptographic protocols to determine SLA compliance. One advantage of this technique is reduced dependence on an external monitoring service or Trusted Third Party (TTP). The assurance provided by the TTP during active mode is provided in passive mode by the Service Evidential Protocol (SEP) [15].

SEP uses cryptographic primitives to ensure confidentiality and non-repudiation of messages. Messages are encrypted using asymmetric (private key) encryption to ensure that messages can only be read by their intended recipients. Messages are signed using digital signatures. Digital signatures mathematically prove the authenticity and integrity of a single message. A valid signature indicates that the message was actually created by the sender and was not modified enroute. The inverse is also true: if a message bears the digital signature of a given party, that party cannot deny or repudiate sending that particular message. This property is referred to as *non-repudiation*. An aggregate signature is a specific type of digital signature that can combine multiple signatures into a single signature [16]. In contrast to a normal digital signature, an aggregate signature can be used to verify a collection of multiple messages in a single operation.

In general terms, this protocol works in the following way. First, all parties of an SLA perform their own measurements. In contrast to active monitoring, no TTP is used to perform these measurements. Then, at specified intervals, all parties exchange commit messages. Commit messages are only exchanged if all parties are satisfied that no violations have yet occurred. Therefore, by sending a commit message, a party is declaring that no violations have occurred and service may continue. Commit messages are required from all parties for the service to proceed. For this reason, this protocol is also referred to as a *mutual commit* protocol.

3.1 Data Collection

This passive monitoring technique uses several cryptographic primitives to collect evidence of SLA compliance or violation. These primitives are: (1) asymmetric key cryptography for encryption and decryption on messages (e.g. RSA [17]), (2) a signing scheme to sign and verify messages (e.g. BLS [16]) and (3) aggregation of signatures for verification of multiple signatures.

A single iteration of SEP is described below. The protocol is a modified version of the original protocol introduced in [15]. A high-level overview of this protocol is shown in Figure 1.

Essentially, a single iteration of the protocol consists of the same message being passed back and forth four times between the consumer and provider. Each time the message is sent, it is encrypted or signed with a different key. The cryptographic encryption and signature scheme provides the required security. These messages are added to an audit log that can later be analysed to prove that a certain message was indeed received by a certain user.

Step 1: a provider encrypts a message (i.e. the access code) with the TTP's encryption key to create a cipher text. The provider then signs the cipher text. The resulting signature is added to the aggregate signature.

Step 2: The provider sends both the signature and the cipher text are sent to the consumer.

Step 3: A consumer receives, but cannot directly decrypt the cipher text. First, the signature is verified to see if it came from the provider and was not altered in transit. The consumer then signs the cipher text using its signing key. Both the resulting signature and the received signature are added to the aggregate signature.

Step 4: The consumer sends the new signature is sent back to the provider as "proof of receipt."

Step 5: The provider receives and verifies the signature. The provider then encrypts the original message again, but now uses the consumer's encryption key. The provider signs this new cipher text. Both the resulting signature and the received signature are added to the aggregate signature.

Step 6: The provider sends both the new signature and the new cipher

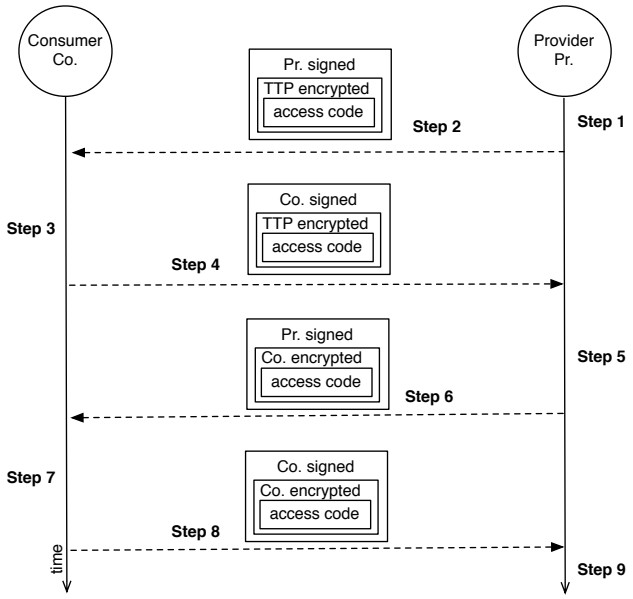


Figure 1: High-level overview of Service Evidential Protocol.

text to the consumer.

Step 7: The consumer receives and verifies the signature. The consumer then decrypts the cipher text to reveal the original message. The consumer then signs the cipher. Both the resulting signature and the received signature are added to the aggregate signature.

Step 8: The new signature is sent back to the provider.

Step 9: The provider receives and verifies this last signature and adds it to the aggregate signature.

At this point in the protocol, both parties have equivalent aggregate signatures Σ_p and Σ_c. These signatures can only be created if a party has access to the original messages, in the original order. Possession of an aggregate signature implies possession of the original messages. In addition, all parties can cryptographically show that all messages have been received and their contents have not been manipulated.

4 Self-adaptive service monitoring

The self-adaptive monitoring approach is designed around the notion that the level of monitoring required for a given service transaction is a reflection of *perceived risk*. For instance, the importance of detecting a violation in a mission critical service differs from that of detecting a violation in a non-mission critical service. Violations ultimately have different levels of impact (e.g. financial, operational).

The level of perceived risk of a service is the product of many factors. These factors include level of trust between parties, likelihood of violation and impact of violation. In this context, trust is defined as a combination of several metrics used in electronic markets [18, 19, 20]. These metrics include personal metrics, such as transaction history, transaction cost and ability to verify results. These also include community metrics, such as popularity, activity and position of a party in a community. These metrics change over time. For instance, a successful transaction 10 years ago has less impact on the level of trust than a successful transaction yesterday.

A party's reputation is an important factor in determining levels of trust. For instance, reputation often determines the first service provider a consumer chooses [21]. If a provider is well-known (e.g. Amazon Web Services), this increases a consumer's initial trust. Once service provisioning has begun, a consumer dynamically adjusts his or her level of trust based on factors such as the number of successful transactions completed.

Different services have different levels of perceived risk and, therefore, require different levels of assurance with monitoring. The 'amount' or 'level' of monitoring may need to adapt accordingly. Self-adaptive monitoring is the term used to describe this adaptation. In this paper, high level monitoring equates to frequent active measurements. In contrast, low level monitoring equates to less frequent active measurements or passive measurements.

4.1 Adaptation model

The choice of which monitoring level to use for a particular service transaction or service period is based on a self-adaptive monitoring function. On the

basis of a perceived *risk level (RL)* and a *monitoring policy (P)* this function chooses an appropriate monitoring level. The monitoring level is the combination of (1) a mode (active or passive) and (2) an interval (time between measurements). Each party to an agreement performs two actions before a transaction or period of transactions. First, RL is calculated on the basis of the risk factors. Based on the RL, a party then selects an appropriate monitoring level.

Self-adaptive monitoring requires that all parties use the same level of monitoring for any given transaction. If different parties to an agreement select different levels of monitoring, the highest level is used. For instance, if a service provider selects active mode and a service consumer selects passive mode, then active mode is used for both parties. This guarantees that all parties have at least the minimum level of assurance required.

4.2 Risk level

The self-adaptive monitoring function uses the current level of perceived risk. This level is calculated based partly on knowledge supplied by the service environment. The environment supplies information about a particular party, including their recent activity and popularity in the environment. An example of environmental knowledge is a reputation authority [20].

In addition to environmental knowledge, the level of perceived risk is also influenced by local knowledge. This includes the price (or cost) of the current transaction and history (if any) of transactions with the given party. These two factors can be used to form a trust matrix (e.g. [19]). The *transaction cost* of a current transaction and the *transaction history* correspond to a particular level of perceived risk. In short, the higher the cost of a transaction, the higher the perceived risk. Conversely, the better the transaction history, the lower the perceived risk.

Transaction cost is an artificial value that reflects the negative impact that would occur if a certain transaction were to fail (e.g. the other party violates the agreement). This value is derived by first mapping levels of transaction cost to ranges of actual price. As an example, transactions below 100 euro

may correspond to a cost of 1. Whereas transactions between 100 and 200 euro may correspond to a cost of 2 and so on. This mapping differs between parties, as each party may have his or her own mapping.

Transaction history is defined as a value that reflects the level of satisfaction with a given party, based on past interactions with that party. The higher the number of successful interactions in the past, the higher the transaction history. This value also takes into consideration the effect of *information decay*, as proposed in [20]. In this context, information decay means that recent transactions influence transaction history more than transactions that occurred long ago. Therefore, a weighting scheme is used to give more weight to the outcome of the most recent transaction and less weight to a transaction, the older it is.

4.3 Monitoring policy

Each party maintains a personal monitoring policy that specifies the relationship between a level of perceived risk and a level of monitoring. A monitoring level is defined as a monitoring mode (active or passive) and the frequency of measurements. For instance, a party P may decide that a risk level of 1 corresponds to a low level of monitoring. Party P defines a low level of monitoring to be passive mode with an interval of 90 seconds and a risk level of 10 which corresponds to a high level of monitoring. Party P defines a high level of monitoring to be an active mode with an interval of 5 seconds. Each and every party creates his or her own customized mapping between the level of perceived risk and the level of monitoring. A policy can change over time, based on lessons learned through interactions in the marketplace.

In addition, a policy can specify that weights may be added to specific pieces of information. Weights are used to indicate relative importance of information, based on age or type. For instance, the 10 most recent transactions may be specified to be twice as important as all past transactions. A policy also contains a value to indicate how quickly a monitoring process should react to changes in the level of perceived risk. This value can be used to prevent a monitoring process from constantly adapting to slight fluctu-

ations in risk levels. A policy may contain additional metadata, such as a policy name.

4.4 Use case scenario

This section describes a use case scenario that illustrates how self-adaptive SLA monitoring can be used in the Cloud environment with service provider P and service consumer C and a TTP.

P and C agree upon an SLA for a set of Cloud services. As part of this SLA, P and C agree to use self-adaptive monitoring mediated by a chosen TTP. At this point in the lifetime of service provisioning, both parties have little knowledge about one another, so the perceived risk level is high. According to their individual policies, both parties begin service monitoring in the active mode, with an interval of 10 seconds.

After a period of 5 minutes without violation, the monitoring policy automatically decreases the perceived risk level, as trust grows between the parties. This change in risk level is reflected in the monitoring process with a decrease the measurement interval to 20 seconds.

After another period of 5 minutes of successful interaction, the monitoring policy automatically decreases the perceived risk level again. The change in risk is reflected, according to the monitoring policy, in a switch to a passive mode with an interval of 60 seconds.

After 5 more minutes without violation, trust increases, risk decreases and the measurement interval is extended to 90 seconds. However, during the next period of 5 minutes, C detects that an SLA violation has occurred. The TTP is contacted to perform conflict mediation (see Section 5) as agreed upon in the SLA. The TTP discovers that P has violated the terms of the SLA and P is penalized. Both parties nevertheless decide to reinstate the SLA. The perceived level of risk is affected by the violation, therefore the monitoring policy automatically switches to an active mode of monitoring with an interval of 10 seconds.

This use case has been simulated in the experiments discussed in Section 7 and illustrated in Figure 9.

5 Conflict mediation

During service provisioning, if one of the parties suspects an SLA violation, conflict mediation can be requested. Note that this requires the use of a Trusted Third Party (TTP). The resulting dependence on external parties, however, may be minimal especially if no conflict mediation is requested. The actions taken by a TTP depend on the current monitoring mode. If the parties are currently using the active mode for their SLA, the TTP consults the measurement results it has stored. If the parties are currently using the passive mode for their agreed SLA, the TTP performs the conflict mediation protocol.

When conflict mediation is requested, the TTP uses the data collected by each of the parties individually using the Service Evidential Protocol (SEP). The TTP investigates this data to determine if an SLA has been violated. To this purpose, the TTP first requests all data that has been collected by each party (i.e. signatures and encrypted messages). The TTP then performs cryptographic checks on the data to verify that the data is genuine and has not been tampered with. The TTP then determines which, if any, party has violated the SLA. This protocol is a modified version of the protocol introduced in [15].

Essentially, the TTP collects the audit logs from both the consumer and provider. The TTP then checks if these audit logs are equal or have been falsified or modified in any way. If a modified audit log is detected, that party can be penalized. If both logs pass inspection, the TTP assumes that there was an error in communication, such as a lost message. The TTP generates and resends these missing messages.

If no violation is detected, the service continues without change. If and when a violation is detected, the responsible party is penalized. At this point, the parties decide how to proceed on the basis of the agreed SLA. For example, an SLA may dictate that the responsible party is penalized with a monetary fine and the service continues, but the monitoring process is switched to active mode with a short measurement interval to reflect the heightened level of perceived risk, as proposed in the example above in Sec-

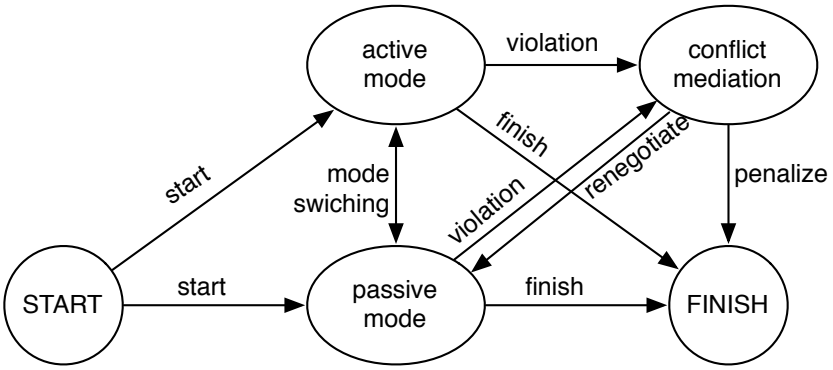


Figure 2: State diagram of self-adaptive monitor (adapted from [15]).

tion 4.4.

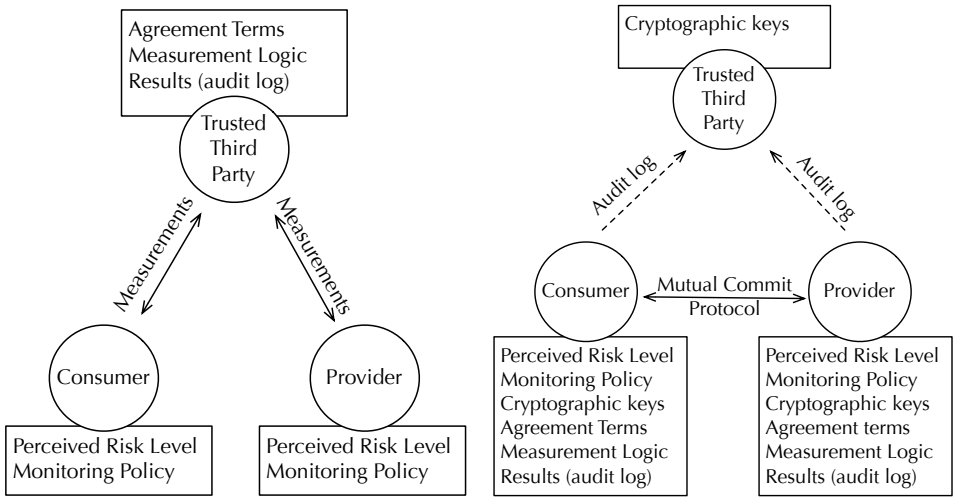
Figure 2 shows the possible state transitions when conflict mediation is requested. Monitoring starts either in active or passive mode. Monitoring policies dictate when to switch between modes (e.g. after a period without violation). If no violations occur, monitoring finishes normally when the agreement expires. If a suspected violation is detected, conflict mediation begins. If the TTP determines that a violation has occurred, multiple actions are possible, depending on the given monitoring policies. On possibility is that the offending party is penalized and the service terminated. Another possibility is that service continues under a renegotiated SLA. For instance, the consumer receives a reduction in price for a reduction in the Quality of Service (QoS).

6 Monitoring Framework Deployment

There are three main parties to an SLA: a Service Provider, a Service Consumer and a TTP. Each party requires access to certain information, depending on the monitoring mode. The monitoring mode also determines which information is exchanged between which parties. These differences can be seen in Figures 3a and 3b.

In active mode, depicted in Figure 3a, the Service Consumer and Service Provider require only a minimum of data. Both parties require their individ-

ually computed Risk Level and their own Monitoring Policy. These parties require no further monitoring data. The TTP is responsible for performing measurements and testing for SLA violations. To perform these responsibilities, the TTP requires a copy of the SLA (Agreement Terms), the Measurement Logic required to perform the necessary measurements. The TTP also stores the Measurement Results (e.g. audit log). All monitoring communication is initiated by the TTP directly to each individual party. All service related communication between Service Consumer and Service Provider is routed via the TTP.



(a) Active mode (b) Passive mode

Figure 3: Data and communication of (a) active and (b) passive monitoring.

In passive mode, depicted in Figure 3b, monitoring responsibility is shifted to the Service Consumer and Service Provider. The TTP therefore requires no knowledge of the SLA (Agreement Terms) or Measurement Logic. This data is stored by each individual party separately. Furthermore, the Measurement Results are stored locally by each individual party. Monitoring communication is directly between Service Consumer and Service Provider. The TTP becomes involved only when conflict mediation is requested in response to an SLA violation at which point the Audit Log sent to the TTP. Passive mode requires additional Cryptographic Keys. Each party maintains

its own set of keys used for encryption and non-repudiation.

6.1 Deployment: Cloud as Provider

One possible deployment of the self-adaptive SLA monitoring framework in the Cloud is when the Cloud Service Provider (CSP) is the service provider. A consumer can agree upon an SLA for a specific Cloud service directly with a CSP. For instance, consumer C is granted access to ten virtual servers with QoS guarantees regarding uptime and response time. This scenario is depicted in Figure 4.

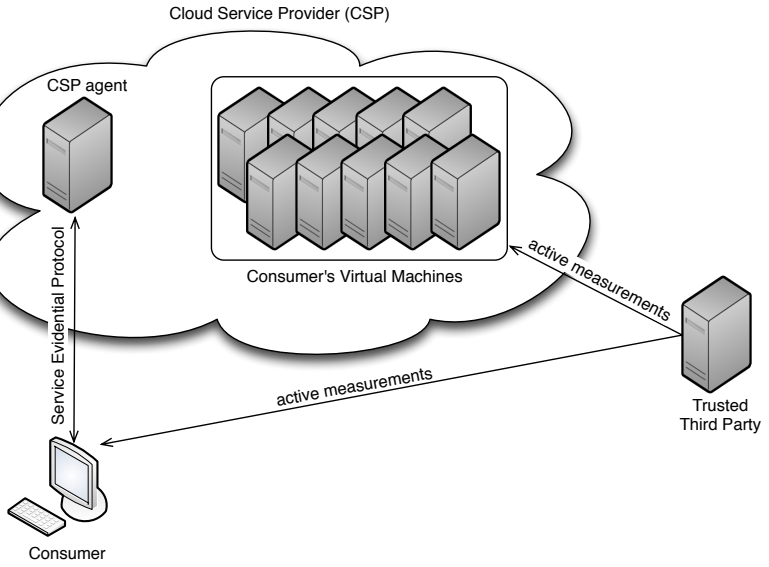


Figure 4: Deployment: Cloud as Provider

A separate, external monitoring service takes the role of TTP. In active mode, a copy of the SLA is given to the TTP. The TTP can then perform active measurements to test the QoS at both the consumer's and provider's side. If a violation is detected, conflict mediation begins and penalties (e.g. fines) are enforced.

In passive mode, the consumer and a CSP agent interact using the Service Evidential Protocol. The CSP agent is located on the CSP Cloud and can directly monitor and verify the QoS (e.g. uptime and response time) of the

consumer's virtual machines. If the consumer detects any violations, the external TTP is contacted.

7 Experimental Validation

Several experiments are performed to measure communication, CPU overhead and scalability. The first set of experiments measures communication overhead. The second set of experiments examines the scalability of each monitoring mode. The final set illustrates how the modes are used together in a self-adapting fashion, as proposed in this paper. An overview of all experiments is given in Table 1.

Experiment	**Mode**	**Agents**	**Interval**	**Mediation %**
Messages_A	Active	2	10, 20, ..., 120	N/A
Messages_P	Passive	2	10, 20, ..., 120	0%, 10%, 30%, 50%
Bytes_A	Active	2	10, 20, ..., 120	N/A
Bytes_P	Passive	2	10, 20, ..., 120	0%, 10%, 30%,50%
ScalingMsgs_A	Active	2-100	30	N/A
ScalingMsgs_P	Passive	2-100	60	0%
ScalingCPU_A	Active	64 - 4096	10	N/A
ScalingCPU_P	Passive	64 - 4096	60	0%
Self-Adaptive	Active & Passive	2	10, 20, 60, 90	N/A

Table 1: Overview of experiments.

These experiments are performed to show the difference in overhead between active and passive modes. However, these experiments do not attempt to define what level of CPU or network overhead is acceptable or unacceptable. This is a subjective threshold that is highly dependent on the context and application. For instance, if resources are cheap and abundant, a user might find it acceptable if the monitor uses 99% of the CPU. However, in an environment with limited network bandwidth, a user may find it unacceptable if the monitor uses more than 5% of the bandwidth. A user can customize his/her monitoring policy to fit a specific context and application.

All experiments are performed using the AgentScape middleware[1]. The AgentScape middleware supports SLA creation and monitoring using the

[1]More information, including source code available at: `http://www.agentscape.org`

WS-Agreement specification [22]. In this environment, consumers, producers and TTPs are represented by self-contained, mobile software agents. In this implementation, asymmetric cryptography is accomplished using the Rivest-Shamir-Adleman (RSA) [17] algorithm. The signing scheme is based on the Boneh-Lynn-Shacham (BLS) [16] algorithm[2].

7.1 Communication overhead experiments

The communication overhead experiments include Messages_A, Messages_P, Bytes_A and Bytes_P. These experiments compare active and passive modes in terms of the number of messages per minute and the amount of network traffic in bytes per minute. All experiments run for 30 minutes. All monitoring related messages are counted and measured (size in bytes). The total count and size are then divided by 30 to produce the results per minute. Each experiment is repeated with measurement intervals ranging from 10 to 120 seconds, in increments of 10.

In the *Messages_A and Bytes_A* experiments, one consumer and one provider create an agreement that uses active monitoring with a constant, specified interval. One TTP and two measurement sensors are used.

In the *Messages_P and Bytes_P* experiments, one consumer and one provider create an agreement that uses passive monitoring with a constant, specified interval. One TTP is used. Both of these experiments are repeated with an additional variable to measure the impact of mediation on communication overhead. Each round of experimentation used a different mediation percentage (0%, 10%, 30% and 50%). A mediation percentage of 10 indicates that, on average, conflict mediation will be requested 10 percent of the time.

7.1.1 Experimental environment

The communication experiments are performed on a single machine. The machine is a SUN SPARC Enterprise T5240 with 128 1.2GHz CPUs and

[2]The BLS implementation uses the Java Pairing Based Cryptography Library (jPBC) provided by Angelo de Caro (`http://gas.dia.unisa.it/projects/jpbc/`). Source code is available upon request.

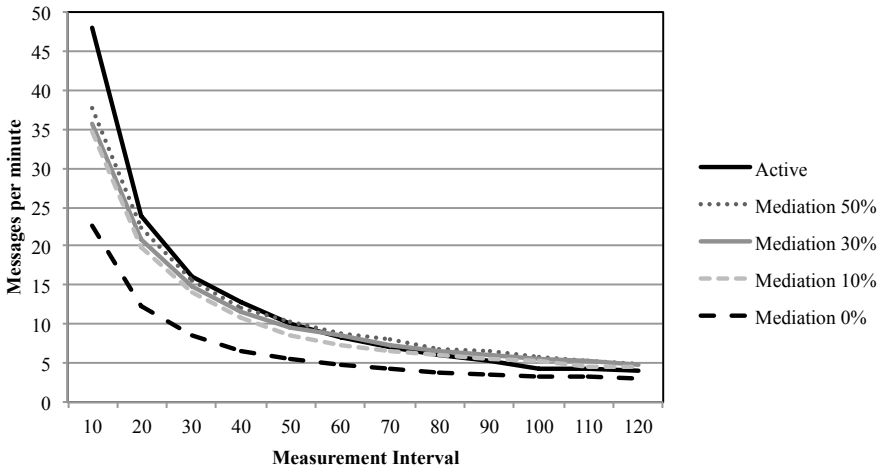


Figure 5: Monitoring messages per minute with 2 agents.

64GB of RAM. This machine runs Solaris 10 and AgentScape middleware. Two AgentScape Locations are created: C and P. A single Consumer agent is hosted at Location C and a single Provider agent is hosted at Location P. A single TTP is hosted at Location P. Exactly one Consumer and one Producer agent are used for this set of experiments. In the case of the active monitoring mode, two sensors (Trusted Monitoring Modules) are used.

7.1.2 Experimental results

The $Messages_A$ experiment results, depicted by the solid, black line in Figure 5, show the number of messages per minute used for active monitoring mode. When the measurement interval is 10 seconds, the monitor uses 48 messages per minute. As the measurement interval increases, the number of messages per minute drops. The number of messages per minute drops below 10 when the measurement interval is greater than or equal to 50 seconds.

The $Messages_P$ experiment results in Figure 5 show the number of messages per minute used for passive monitoring mode. The baseline (Mediation 0%) shows that when the measurement interval is 10 seconds, monitoring uses 22 messages per minute. The average number of messages per minute drops below 10 when the measurement interval is 30 seconds or more.

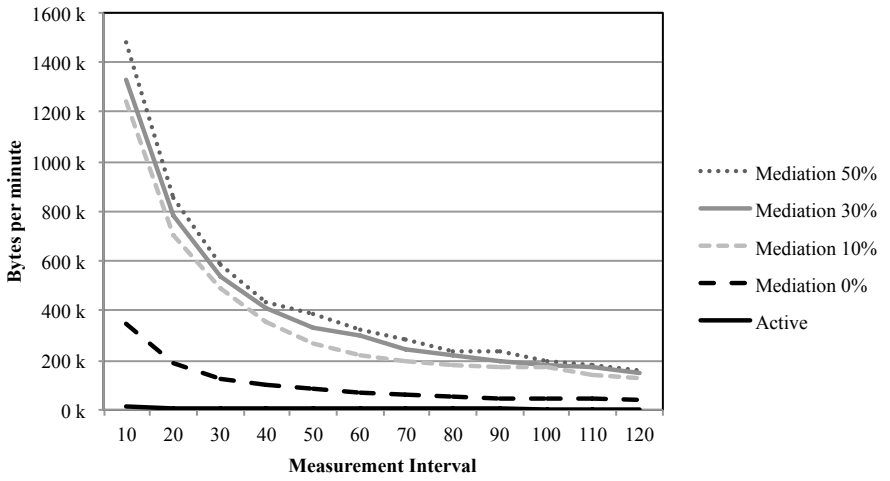


Figure 6: Monitoring bytes per minute with 2 agents.

Several variations of the experiments are also shown with varying percentages of conflict mediation (Mediation 10%, Mediation 30%, Mediation 50%). While there is a significant difference between no mediation at all and 10% or more mediation, there is little difference between the frequency of mediation. When the measurement interval is 10 seconds, the ascending mediation scenarios require an average of 34.6, 35.6 and 37.7 messages per minute, respectively. For all scenarios, this number drops below 10 when the measurement interval is greater than or equal to 50 seconds.

The $Bytes_A$ experiment results, depicted by the solid black line in Figure 6, show the network traffic of active monitoring mode in bytes per minute. When the measurement interval is 10 seconds, network traffic is 16k. Traffic drops below 4k when the measurement interval is greater than 40 seconds.

The $Bytes_P$ experiment results in Figure 6 show the network traffic of passive monitoring mode in bytes per minute. The baseline (Mediation 0%) shows the scenario when there is no conflict mediation requested (e.g. participants do not detect violations). When the measurement interval is 10 seconds, monitoring generates 348k bytes per minute of network traffic. Traffic drops below 100k when the measurement interval is 40 seconds or greater.

Several variations of the experiment are also shown with varying percent-

ages of conflict mediation (Mediation 10%, Mediation 30%, Mediation 50%). These scenarios generate significantly more bytes per minute on average than the baseline scenario. While there is a significant difference between no mediation at all and 10% or more mediation, there is little difference between the frequencies of mediation. When the measurement interval is 10 seconds, the mediation scenarios generate an average network traffic of 1,243k, 1,328k and 1,479k, respectively. For all scenarios, traffic drops below 400k when the measurement interval is 40 seconds or greater.

7.1.3 Discussion of results

The communication overhead experiments indicate a significant difference between active and passive mode with regard number of messages per minute and the total network traffic generated, as can be seen in Figure 5. Passive mode clearly uses fewer messages per minute than active mode. This difference can be clarified with an explanation of the inner workings of passive mode. When no conflict mediation is requested, communication occurs directly between consumer and provider. No TTP or TMM are involved. This results in fewer messages per minute than the corresponding active mode.

The difference in the network traffic generated by active and passive mode can be seen in Figure 6. Passive mode clearly generates significantly more bytes per minute than active mode, especially when conflict mediation is requested. In the case that no conflict mediation is requested, consumer and provider exchange cryptographically encrypted portions of their individual audit logs. These audit logs are much larger than the simple metrics exchanged in active mode. In the case that conflict resolution is requested, all parties send their entire audit log to the TTP, resulting in a substantial increase in network traffic.

7.2 Scalability experiments

The scalability experiments include ScalingMsgs$_A$, ScalingMsgs$_P$, ScalingCPU$_A$ and ScalingCPU$_P$. These experiments investigate the ability of each monitoring mode to scale in distributed environments.

In the *ScalingMsgs$_A$ and ScalingMsgs$_P$* experiments, consumers and providers create agreements that use either only the active monitoring mode with a fixed interval of 30 seconds or only the passive monitoring mode with a fixed interval of 10 seconds. These values were chosen based on the performance of previous experiments. One TTP and two TMMs are used. As with the previous set of experiments, each of these experiments is run for 30 minutes and results are averaged. Each experiment is repeated with the number of agents ranging from 2 to 100, in increments of 2.

In the *ScalingCPU$_A$ and ScalingCPU$_P$* experiments, a consumer agent and a provider agent are launched simultaneously on 32 separate Locations on 32 separate nodes. A consumer chooses a provider from amongst the Locations using a random function with a uniform distribution. The chosen provider is always located on a different node than the consumer. After choosing a provider, the consumer creates an agreement that uses monitoring. ScalingCPU$_A$ uses active monitoring mode with an interval of 10 seconds. ScalingCPU$_P$ uses passive mode with an interval of 60 seconds. These values were chosen based on the performance of previous experiments.

A new consumer agent and provider agent are launched every 5 seconds until the desired number is reached. For the first run, this number is 64 (2 per node). The number of agents increases by 64 on each consecutive run until reaching 4096 (128 per node). The entire process is repeated 5 times and the CPU load results for all 32 participating nodes are averaged.

7.2.1 Experimental environment

The first set of scalability experiments (ScalingMsgs$_A$, ScalingMsgs$_P$) are performed on the same machine and configuration as the communication overhead experiments described above.

The second set of scalability experiments (ScalingCPU$_A$, ScalingCPU$_P$) are performed using the Distributed ASCI Supercomputer - version 4 (DAS-4)[3]. An AgentScape World is created using 33 independent nodes of the DAS-4 grid. Each node has a 2.4 GHz processor, 24GB of memory and runs

[3] `http://www.cs.vu.nl/das4/`

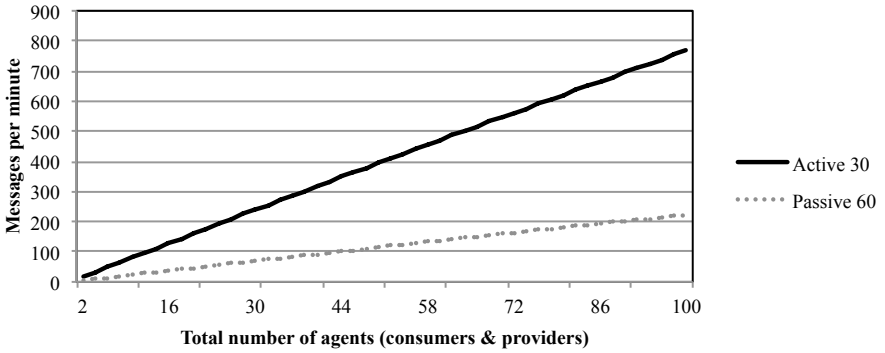


Figure 7: Message overhead with increased scalability

CentOS Linux. The machines are connected with gigabit ethernet. One node is reserved for the AgentScape Lookup Service that provides listings of all known Locations. On the remaining 32 nodes, a single AgentScape Location runs with a single TTP Agent. Thus, there are 32 TTP Agents in the World.

7.2.2 Experimental results

The $ScalingMsgs_A$ and $ScalingMsgs_P$ experiment results are shown in Figure 7. These results indicate the messages per minute of the two monitoring modes as the system scales to 100 agents. Both modes scale linearly, but passive mode requires fewer messages, overall.

The $ScalingCPU_A$ and $ScalingCPU_P$ experiment results, shown in Figure 8, demonstrate the scalability of both modes of monitoring in a highly distributed environment. Active mode performs better than passive mode. Even with 4096 agents in the system, each being monitored at 10 second intervals, the average CPU load remains below 0.5%. The passive mode approaches 4% with 4096 agents being monitored at 60 second intervals.

7.2.3 Discussion of results

The scaling results shown in Figure 7 indicate that both monitoring modes can scale, in terms of communication overhead. Doubling the number of agents doubles the number of messages. The distributed scaling experiments

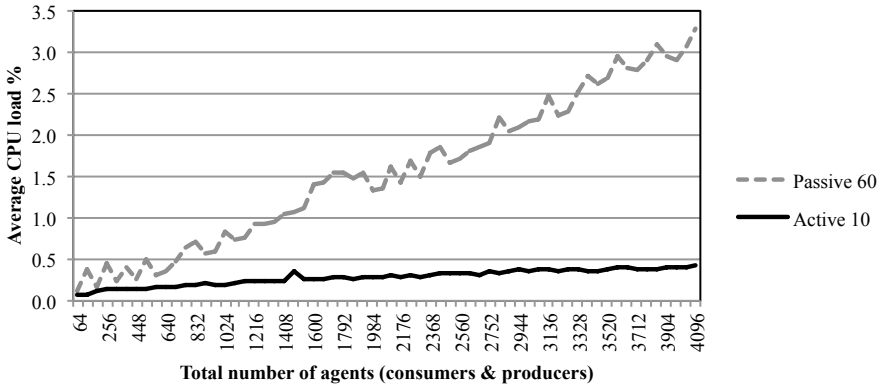


Figure 8: Average CPU load of large scale monitoring on DAS-4.

shown in Figure 8 indicate that both monitoring modes can scale in large, distributed systems with only minimal computational overhead. In active mode, CPU load is affected by the interval of measurement. In passive mode, CPU load is determined by both the interval of measurement and the frequency with which mediation is requested.

The passive mode has a higher average CPU load than the active mode. This is to be expected, given that passive mode requires CPU-intensive cryptographic computations[4]. Active mode uses no cryptography, because security is guaranteed by the TTP.

What cannot be seen in these particular figures is the level of interaction with the TTP. In passive mode, there is only a minimal amount of interaction with the TTP (e.g. exchanging cryptographic keys). All computations, including measurements and cryptography, are performed by the consumer and provider. In contrast, active mode relies on the TTP to perform all measurements. With this in mind, the active mode shown in Figure 8 essentially depicts the CPU load at the TTP. The corresponding CPU load at the consumer and provider is insignificant. In contrast, the passive mode shown in the same figure essentially depicts the average CPU load at the consumer and provider. The corresponding CPU load at the TTP is insignificant. In a

[4]In particular, the BLS implementation is chosen only for its functionality. This code is not optimized for production level systems.

'real world' environment, a party would likely pay for the services of a TTP. In this case, the reduction in TTP interaction associated with passive mode would result in reduced financial costs.

7.3 Self-adaption experiment

The *Self-Adaptive* experiment is performed to combine both modes and give an example of the self-adaptive monitoring approach proposed in this paper. This experiment shows the changes overhead as the monitoring process dynamically switches between intervals and modes.

One consumer and one provider create an agreement that begins with an active monitoring mode with an interval of 10 seconds. Both parties use the same monitoring policy that specifies that after 5 minutes without violations, the Risk Level decreases. This decrease is reflected by increasing the monitoring interval or changing modes. When a party detects an SLA violation, mediation is requested. The result of mediation is to increase the Risk Level. This increase is reflected in the monitoring level by resetting to the initial configuration of active mode with an interval of 10 seconds. The self-adaptive experiment is repeated 10 times and the CPU load results are averaged.

7.3.1 Experimental environment

The Self-Adaptive experiment is performed on a network of two machines connected across gigabit ethernet. The first machine has a 2.0GHz dual core CPU and 1GB of RAM. The second machine has a 2.0GHz dual core CPU and 2GB of RAM. Both machines run Ubuntu Linux and AgentScape middleware. Two AgentScape Locations are created, one on each machine: C and P. A consumer agent is hosted at Location C and a Producer agent is hosted at Location P. A single TTP is hosted at Location P.

7.3.2 Experimental results

The Self-Adaptive experiment results in Figure 9, provide an overview of the self-adaptive monitoring process. Average CPU usage is indicated by the

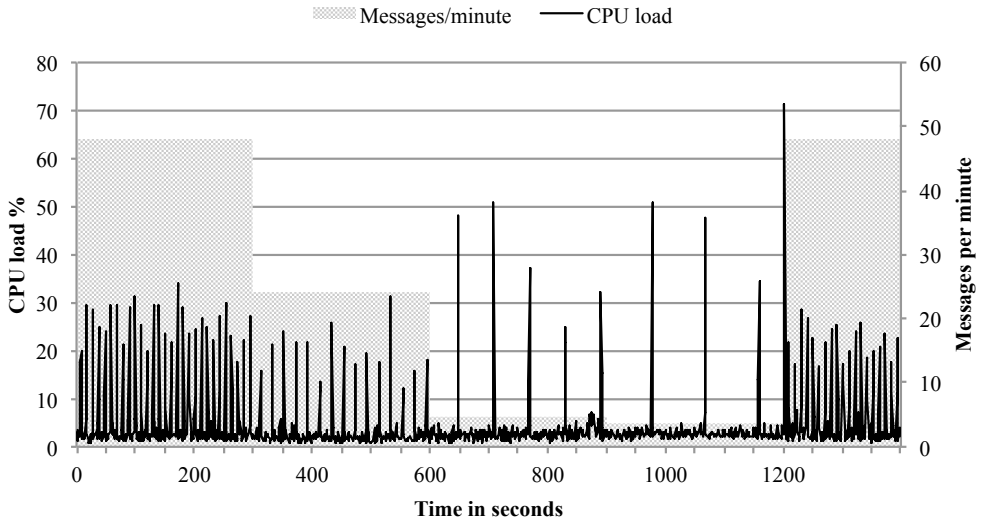


Figure 9: Overhead of self-adaptive monitoring with two agents.

solid, black line. Average messages per minute is indicated with shaded, grey bars. In active modes, CPU usage has a consistent pattern reflecting the 10 and 20 second monitoring interval, respectively. The number of messages per minute is relatively high. In passive modes, consistent CPU usage reflects the 60 and 90 second monitoring interval, respectively. The number of messages per minute is substantially lower. CPU usage surges when conflict mediation is requested. This surge reflects the additional cryptography required to verify the messages and aggregate signatures used in the audit logs.

7.3.3 Discussion of results

This experiment demonstrates how monitoring overhead reflects perceived risk. The self-adaptive monitoring policy used for this experiments starts with a high level of perceived risk. After a period of time without incident, the level of perceived risk lowers. This is reflected by increasing the measurement interval and thus reducing overhead. Eventually, this leads to a switch to passive mode, which reduces dependence on a TTP.

In the case that a violation is detected, mediation is requested. This may result in a higher level of perceived risk and a correspondingly high level

of monitoring. This significantly increases the message and CPU overhead but offers higher assurance. By adjusting the monitor to match the level of perceived risk, overhead can be substantially reduced when possible.

8 Related Works

There is growing interest in the use of SLAs in the Cloud environment, in particular with the specific task of monitoring SLAs. The Lattice Monitoring Framework (LMF) detailed in [23] has been designed as a resource monitor for virtualized environments, such as the Cloud. This framework makes a clear separation of roles between service providers and infrastructure providers. For instance, Dropbox is a service provider and Amazon S3 Storage is the infrastructure provider. LMF aims to monitor services across multiple infrastructure providers. However, it is not clear which party has access to the actual measurements, and therefore, could potentially tamper with measurement results. In contrast to the approach presented in this paper, LMF has been designed to be a tool specifically for the provider and is thus opaque to the consumer. No Trusted Third Party (TTP) is employed to ensure that monitoring results are correct.

The monitoring framework presented in [4] is integrated in the SLA@SOI project. This approach also focusses on the provider's perspective. The framework collects and stores historical monitoring data that is used during negotiation to evaluate the SLA offers made by a customer. Furthermore, the system checks the SLA at runtime to determine if the terms are capable of being monitored at all. This is referred to as the *monitorability* of the SLA. Each service provider has a list of terms for which the required measurement logic and sensors are available. If a new SLA has terms that are not compatible with this list and therefore not *monitorable* by this particular provider, the SLA is rejected. The system is completely controlled by the CSP and is opaque to the consumer. No TTP is used, thus consumers must implicitly trust that the provider does not manipulate monitoring results.

The SLA framework proposed in [24] uses SLA as a mechanism for CSPs to keep track of QoS commitments. CSPs monitor system metrics to de-

termine if QoS commitments are being met and then dynamically provision their resources to prevent over- or under provisioning. A similar framework is described in [25]. The service provider monitors SLAs internally for possible violations. If necessary, the system can automatically replicate services to meet obligations, such as response time and transaction rate. In effect, both of these systems are SLA driven load-balancing systems. They provide tools for the provider to optimize system load while meeting SLA obligations. In contrast to the approach presented in this paper, there is no TTP and the systems are only designed for internal QoS monitoring at the provider. Transparency and objectivity of the monitoring process are not considered.

A mechanism for enforcing SLAs for scientific computing is presented in [26]. In contrast to typical commercial Clouds, this mechanism is tailored to deadline-driven batch jobs, as used in, for example, scientific compute grids. At runtime, a fuzzy prediction algorithms estimates the amount of resources needed for each job. Using this estimation, the system can determine if an SLA can be met and accepts or rejects a job accordingly. If a job is accepted, it is assigned a software agent. This agent monitors a job during its lifetime and dynamically increases resources to ensure that its SLA is not violated. Similar to some of the frameworks described above, this system is not designed to ensure transparency of the monitoring process for consumers. Consumers are forced to trust the monitoring results from the provider.

The self-adaptive SLA monitoring framework proposed in this paper differs from the frameworks described above in several ways. First, the approach proposed in this paper explicitly establishes the role of a dedicated TTP. This ensures that monitoring results are accurate and neither party can manipulate the outcome. Furthermore, the monitoring framework views the consumer and the provider as equal parties. As such, the monitoring framework is equally transparent to both parties. This allows both parties, not just the provider, to trust the monitoring results.

A monitoring technique that uses a different approach to address the issue of trust is the QoS-MONaaS presented by [27]. This is an extension to the SRT-15 middleware[5]. In contrast to some of the approaches described above,

[5]The SRT-15 project to develop a distributed service platform: `http://www.srt-15.eu`

this system is designed to support trustworthiness of monitoring results. This system uses an anonymizer function to ensure fairness in measurement. The claim is that because the monitoring framework does not know the identity of a party that requests measurement results, there is no incentive to tamper with these results. For example, in a scenario where both a consumer and a provider request monitoring results, the monitoring framework cannot provide correct results to the provider and tampered results to the consumer.

In contrast to the monitoring frameworks described above, some monitoring techniques are able to dynamically adapt their monitoring policy at run-time based on environmental limits or changes in priorities. For instance the monitoring approach proposed in [13] collects system notifications from distributed nodes and can dynamically adjust the frequency of notifications, based on CPU load. The higher the load (e.g. more users in the system), the lower the frequency of notifications. Another example of dynamic monitoring is the system monitor described in [14]. This monitoring process attempts to reduce monitoring overhead by pre-selecting and focusing on key metrics. Only when an anomaly is detected in one of these key metrics, does the monitoring process adapt by increasing the number of related metrics that are continuously monitored. Effectively, this monitoring process is able to 'zoom in and out' of areas when problems are detected.

The self-adaptive SLA monitoring framework proposed in this paper differs from these two approaches in two major ways. First, not only does the approach proposed in this paper react to changes in system overhead, but also to changes in the level of perceived risk. Secondly, rather than only adjusting the measurement interval, the monitoring process is able to switch between two major modes of monitoring: active and passive. In the passive mode, dependence on the TTP is significantly reduced. An effect of this is a reduction in costs and the ability for the system to scale.

9 Conclusion

This paper presents a self-adaptive SLA monitoring framework for distributed computing environments, such as the Cloud, that is capable of automatically

adapting to changing levels of perceived risk. Increased perceived risk results in increased monitoring assurance and vice versa.

This framework has been experimentally validated in a large scale, distributed system. Experiments show that both monitoring modes scale well to at least 4096 agents distributed across 32 separate machines. This demonstrates the applicability of this monitoring approach to large, distributed markets, such as the Cloud.

Future work will explore integration with other trust-based approaches, such as reputation mechanisms. As this monitoring process can react to changes in the level of trust between parties (e.g. the history of successful transactions), this can also be used for trust-building. For instance, choosing a lower monitoring frequency or choosing explicitly for passive mode can be interpreted to be an act of goodwill. Parties can use this monitoring approach to convey trust in one another, which may lead to better self-policing and higher QoS. The effects of this application of this monitoring approach will be investigated in future research.

The feasibility of the self-adaptive monitoring approach in a real-world setting partially depends on the cost structure of a CSP. Monitoring SLAs with an external TTP has associated costs and performance overhead. Likewise, implementing a self-adaptive monitoring framework also has associated costs, but also offers benefits to both a CSP and their consumers. A CSP must analyze these costs to determine if a self-adaptive monitoring framework, and the benefits it offers, are economically appealing.

Acknowledgements

This work is a result of support provided by the NLnet Foundation (`http://www.nlnet.nl`) and Delft University of Technology. The authors would like to thank Dalia Khader, Julian Padget and Angelo de Caro for their contribution to the initial BLS implementation. In addition, the authors would like to thank the anonymous reviewers for the valuable comments and suggestions to earlier drafts of this paper.

References

[1] K. P. Clark, M. Warnier, F. M. T. Brazier, Self-adaptive service monitoring, in: proceedings of the 2011 International Conference on Adaptive and Intelligent Systems (ICAIS 2011), Springer, 2011, pp. 119–130.

[2] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al., A view of cloud computing, Communications of the ACM 53 (4) (2010) 50–58.

[3] M. Alhamad, T. Dillon, E. Chang, Conceptual sla framework for cloud computing, in: Digital Ecosystems and Technologies (DEST), 2010 4th IEEE International Conference on, IEEE, 2010, pp. 606–610.

[4] M. Comuzzi, C. Kotsokalis, G. Spanoudakis, R. Yahyapour, Establishing and monitoring slas in complex service based systems, in: Web Services, 2009. ICWS 2009. IEEE International Conference on, Ieee, 2009, pp. 783–790.

[5] O. Rana, M. Warnier, T. Quillinan, F. Brazier, D. Cojocarasu, Managing violations in service level agreements, Grid Middleware and Services (2008) 349–358.

[6] A. Ludwig, P. Braun, R. Kowalczyk, B. Franczyk, A Framework for Automated Negotiation of Service Level Agreements in Services Grids, Lecture Notes in Computer Science 3812 (2006) 89–101.

[7] A. Keller, H. Ludwig, The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services, Journal of Network and Systems Management 11 (1) (2003) 57–81.

[8] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, M. Xu, Web Services Agreement Specification (WS-Agreement), in: Global Grid Forum, 2011, pp. 1–81.

[9] N. Hayashibara, A. Cherif, T. Katayama, Failure detectors for large-scale distributed systems, 21st IEEE Symposium on Reliable Distributed Systems.

[10] H. Ludwig, A. Dan, R. Kearney, Cremona: an architecture and library for creation and monitoring of WS-agreents, in: Proceedings of the 2nd international conference on Service oriented computing, ACM New York, NY, USA, 2004, pp. 65–74.

[11] T. B. Quillinan, K. P. Clark, M. Warnier, F. M. T. Brazier, O. Rana, Negotiation and monitoring of service level agreements, in: P. Wieder, R. Yahyapour, W. Ziegler (Eds.), Grids and Service-Oriented Architectures for Service Level Agreements, CoreGRID, Springer-Verlag, 2010, pp. 167–176.

[12] K. P. Clark, M. Warnier, T. B. Quillinan, F. M. T. Brazier, Secure monitoring of service level agreements, in: Proceedings of the Fifth International Conference on Availability, Reliability and Security (ARES 2010), IEEE, 2010, pp. 454–461.

[13] H. Keung, J. Dyson, S. Jarvis, G. Nudd, Self-adaptive and self-optimising resource monitoring for dynamic grid environments, in: Database and Expert Systems Applications, 15th International Workshop on, 2004, pp. 689–693.

[14] M. Munawar, P. Ward, Adaptive monitoring in enterprise software systems, Tackling Computer Systems Problems with Machine Learning (SysML).

[15] D. Khader, J. Padget, M. Warnier, Reactive monitoring of service level agreements, Grids and Service-Oriented Architectures for Service Level Agreements (2010) 13–22.

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

[17] R. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (2) (1978) 120–126.

[18] C. Hsu, Dominant factors for online trust, in: Cyberworlds, 2008 International Conference on, 2008, pp. 165–172.

[19] D. Manchala, E-commerce trust metrics and models, Internet Computing, IEEE 4 (2) (2000) 36–44.

[20] A. Tajeddine, A. Kayssi, A. Chehab, H. Artail, A comprehensive reputation-based trust model for distributed systems, in: Security and Privacy for Emerging Areas in Communication Networks, 2005. Workshop of the 1st International Conference on, 2005, pp. 116–125.

[21] P. Resnick, K. Kuwabara, R. Zeckhauser, E. Friedman, Reputation systems, Communications of the ACM 43 (12) (2000) 45–48.

[22] D. Mobach, B. Overeinder, F. Brazier, A WS-Agreement Based Resource Negotiation Framework for Mobile Agents, Scalable Computing: Practice and Experience 7 (1) (2006) 23–36.

[23] S. Clayman, A. Galis, C. Chapman, G. Toffetti, L. Rodero-Merino, L. Vaquero, K. Nagin, B. Rochwerger, Monitoring service clouds in the future internet, Towards the Future Internet-Emerging Trends from European Research (2010) 1–12.

[24] S. Ferretti, V. Ghini, F. Panzieri, M. Pellegrini, E. Turrini, Qos-aware clouds, in: Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, 2010, pp. 321–328.

[25] V. Stantchev, C. Schröpfer, Negotiating and enforcing qos and slas in grid and cloud computing, Advances in Grid and Pervasive Computing (2009) 25–35.

[26] O. Niehorster, A. Brinkmann, G. Fels, J. Kruger, J. Simon, Enforcing slas in scientific clouds, in: Cluster Computing (CLUSTER), 2010 IEEE International Conference on, IEEE, 2010, pp. 178–187.

[27] L. Romano, D. De Mari, Z. Jerzak, C. Fetzer, A novel approach to qos monitoring in the cloud, in: Data Compression, Communications and Processing (CCP), 1st International Conference on, 2011, pp. 45–51.