Secure Monitoring of Service Level Agreements

K.P. Clark, M.E. Warnier, F.M.T. Brazier
Faculty of Technology, Policy and Management
Delft University of Technology
Delft, The Netherlands
{*k.p.clark, m.e.warnier, f.m.brazier*} *@tudelft.nl*

T.B. Quillinan
D-CIS Lab
Thales Netherlands
Delft, The Netherlands
thomas.quillinan@icis.decis.nl

***Abstract*—Service Level Agreements (SLA) are commonly used to define terms and conditions of service provisioning. WS-Agreement[1] is an SLA specification that addresses the need of both producers and consumers of services to specify and negotiate terms and conditions of access to these services. This specification has gained wide acceptance in both the Grid computing and Web Services communities. WS-Agreement includes support for both negotiating and specifying penalties that arise from violation of these terms and conditions. It does not, however, include support for monitoring these agreements to determine if any such violations have occurred and, if so, determining which parties are responsible.**

This paper proposes a framework and design for secure and reliable monitoring of WS-Agreement specified SLAs. Modifications to WS-Agreement are necessary for effective monitoring. These modifications are outlined, along with an implementation of the framework in the AgentScape middleware system.

***Keywords*-sla, ws-agreement, distributed monitoring, reliability**

I. Introduction

Service Level Agreements (SLA) define the terms and conditions of service provisioning. The Web Service Agreement [1] (WS-Agreement) provides a protocol for negotiating; a language for specifying terms and conditions, and an advertising protocol for services. While the ability to specify penalties and rewards is included in WS-Agreement, the standard does not, as yet, specify how these agreements should be monitored to ensure that the terms and conditions are not violated [2].

When a consumer and provider agree on a service for a specific price, the minimal requirements of this service can be specified as part of an SLA. For instance, an SLA can state that bandwidth must be greater than 1MB, or average up-time must be greater than 99%. If a provider is unable to meet these requirements, a consumer may impose penalties, such as reducing or cancelling payment or reporting the violation to a reputation authority. One method to determine if, when, and by whom, an agreement is violated, is to employ reliable monitoring. Monitoring of active agreements, and the audit trail it provides, can be used to establish whether or not the agreement is violated. Furthermore, the responsible party can be impartially established and a determination reached whether or not sanctions are justified.

One approach to reliable monitoring makes use of a Trusted Third Party (TTP). This TTP can (1) monitor active SLAs by performing measurements at specified intervals and comparing them to threshold values stored in the SLA; (2) record communication between consumer and provider, and (3) take action when a violation is detected, such as notifying the relevant parties, withholding payment or terminating the service.

This paper proposes an SLA monitoring framework that provides reliable data that is accurate and secure.

The principles outlined in this framework are incorporated in a monitor design using the WS-Agreement specification. This design has been implemented in the AgentScape middleware to illustrate the potential of this approach. AgentScape has an established mechanism for SLA negotiation and serves as the TTP.

The main contributions of this paper are (1) the generic framework and design for secure and reliable monitoring of SLAs for violations; (2) a method for specifying violation policies, and (3) an implementation of this framework in the AgentScape middleware with which the feasibility of a decentralised approach is shown.

The remainder of this paper is structured as follows: Section II briefly introduces SLAs and the Web Service Negotiation. Different aspects of monitoring SLAs are examined in Section III. Section IV describes the design of our monitor, including security and reliability considerations. A comparison of centralised and decentralised variants of the monitoring framework, implemented in AgentScape, is presented in Section V. Section VI discusses related research and, finally, Section VII discusses the results and depicts areas for future research.

II. Background

Service Level Agreements (SLAs) represent the main tool for specifying transactions between those providing and those seeking resources. Currently, there are several specifications that describe, not only the structure of an SLA document, but also the processes of advertising and

[1]SLA specification by the Open Grid Forum: http://www.ogf.org

negotiating this document. One of these specifications is WS-Agreement [1].

A. Service Level Agreements

Service Level Agreements (SLA) are agreements between multiple parties to specify terms of service. They involve at least one service provider and at least one service consumer and specify the services to be provided. Additionally, these documents specify Quality of Service (QoS) guarantees; payments for compliance, or penalties for violation of the agreement. QoS attributes 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 object to be measured, how measurements are performed and any actions that take place after measurement.

Traditionally, SLAs are written and signed between legal entities, representing each of the parties involved. In recent years, much attention has been given to automating this negotiation process [3], [4]. Several specifications have been proposed for this purpose, including WS-Agreement [1] and Web Service Level Agreement (WSLA) [5].

B. Web Service Negotiation

The WS-Agreement specification [1] defines a language with which to express agreements, a protocol to advertise available services and a protocol to support negotiation and confirmation of SLAs between resource providers and consumers. This specification defines XML documents that describe the agreement parties and terms. Agreement templates are used to advertise available services and list additional constraints, such as the maximum or minimum QoS levels. The negotiation protocol is defined by the following steps, as depicted in Figure 1.

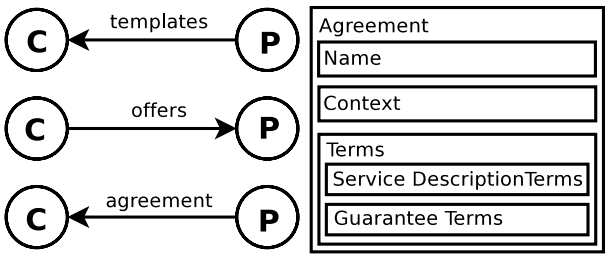

Figure 1. WS-Agreement protocol and SLA specification.

Initially, when a consumer (*Agreement Initiator*) requests an indication of services available from a provider (*Agreement Responder*), the provider produces an overview of available services (*Templates*). Based on this information, the consumer proposes an SLA (*Offer*). The provider decides whether or not to accept the proposed SLA. If accepted, service provisioning begins (*Lease*).

Figure 1 also illustrates the conceptual overview of the SLA document as specified by WS-Agreement. This document has three main components: *Name*, *Context* and *Terms*. The *Name* provides a human-readable identifier. The *Context* contains information concerning the parties involved, the duration of the agreement and information about the original template used. The *Terms* specifies the functional and non-functional requirements of the agreement in *Service Description Terms* (SDT) and *Guarantee Terms* (GT). An SDT identifies the specific services covered by the SLA, for example, the number of processors or amount of bandwidth. A GT specifies the levels (QoS) of those services, for example, availability or response time. A GT also contains additional information regarding the importance of a certain SLO, the reward of compliance or the penalty of violation.

WS-Agreement is an active standards proposal. Since the publication of the original WS-Agreement specification several extensions have been proposed. These include the ability to anticipate violations and renegotiate agreements at run-time [6], finalise the agreement using a two-phase commit [7] and support multiple rounds of negotiation and dynamic re-negotiation due to the changing needs of the consumer or abilities of the provider [8].

III. Monitoring Framework

Monitoring is needed to determine if an agreement is being honoured or not. Once established, appropriate violations can be enforced. The general principles of monitoring SLAs are discussed below.

A. Monitoring Service Level Agreements

Monitoring an agreement involves periodically testing whether the agreement terms have been met by all relevant parties. Depending on the agreement terms, this either requires testing a specific variable, such as network latency, or logging communication between consumer and provider. Monitoring intervals are specified appropriately, such as daily or hourly, depending on the duration of the agreement and the nature of the agreement terms. Monitoring must also support both simple and complex evaluation formulae. For instance, some requirements can be verified by measuring a single variable, such as '*Host is reachable*'. However, other requirements can only be verified once a set of measurements have been performed and their results processed, such as '*Average host uptime is greater than 99%*'.

A monitoring mechanism must be secure against malicious parties, potentially including the parties with whom agreements have been reached. Security mechanisms are necessary to ensure that measurements are accurate and stored data is not tampered with. Such mechanisms should include a secure logging mechanism [9] to accurately record the record of communications and measurements. Protection of this data is needed as stored data also serves as an audit

trail to detect violations offline, after service provisioning has ended.

Furthermore, trust and objectivity considerations determine the placement of a monitoring module. Rana et. al. distinguishes three possible locations for monitoring [2]:

Trusted Third Party (TTP): an independent module that can monitor (and log) all communication between consumers and service providers. Once the SLA is successfully completed, both parties receive a signed certificate from the TTP that can be used for non-repudiation and/or reputation building of the service provider. However, a TTP cannot measure the internal state of either a consumer or provider.

Trusted Module at service provider: functionally equivalent to a TTP but with access to the internal state of the service provider. However, a provider may not reveal all of the internal state or may report incorrect information to the monitor. A module at this location can show that a provider attempted to avoid violations and dealt with them responsibly when they occurred.

Trusted Module on the consumer site: functionally equivalent to a TTP but it can be difficult to distinguish between provider delay and network delay. A module at this location is not particularly useful for measurements, but rather only for establishing the trust level for certain providers.

B. Auditing and Conflict Resolution

In some cases, conflicts arise that cannot be resolved automatically. For example, a service provider contests its liability for failing to meet an SLO. The SLO is known to be violated; however, the cause of the violation is beyond the control of the provider (e.g. force majeure or a Distributed Denial of Service (DDoS) attack [10]). To resolve such cases, it is necessary to record messages and measurements for later processing and also determine which messages to store and the duration of storage. For example, audit logs should be stored until all parties acknowledge that the SLA has been successfully completed. It is also necessary to determine how violations are recorded. Violations can either be stored in the TTP, added explicitly to the SLA document [3] or included in usage records [11]. SLA status updates can also be pushed to parties at intervals or published to a secure site to allow parties on-demand access.

C. Penalising Violations

When a violation occurs, often a penalty is incurred. Penalties can be as simple as terminating the current agreement and finding a different provider, or more complex reputation or monetary based penalties [12]. These penalties are commonly used for service provisioning [13]. In these systems, reputation is a community-wide metric of an entity's trustworthiness. This metric increases if the entity completes transactions without violating the agreement. Conversely, the metric decreases if a term is violated. Reputation based penalties utilise the notion that consumers prefer providers with a higher reputation and try to avoid providers with a lower reputation.

In contrast, monetary based penalties operate on the assumption that consumers pay less for poor service and more for better service.

Both of these mechanisms require additional infrastructure and security measures [13]. A reputation based system requires a persistent record of all transactions, both successful and violated. A monetary based system requires a secure means of payment, whether in currency or credit, that has actual value to the users of the system. Both of these approaches require a means of guaranteeing that identities are unique, persistent and legitimate. For instance, underlying authentication mechanisms can verify that users are indeed whom they claim to be.

Deposits with a jointly agreed TTP can be used in a monetary based system to implement penalties if needed. In the event of violation, the deposit can be used to effectuate penalty payment. The exact penalty terms can be separately negotiated during SLA negotiation or follow known policies, such as the following [2]:

All-or-nothing provisioning: provisioning of a service must meet all SLOs. ALL of the SLO constraints MUST be met to satisfy the SLA;

Partial provisioning: provisioning of a service must meet some SLOs. SOME of the SLO constraints MUST be met to satisfy the SLA;

Weighted Partial provisioning: provision of a service meets SLOs that have a weighting GREATER THAN a [user specified] threshold.

IV. Monitor Design

This section outlines the design of the monitor following the guiding principles discussed above. This includes a description of the components of the monitor and the specification of a monitoring policy.

A. Monitor Overview

The two main conceptual components of the monitoring framework are *Monitor Sensors* and a *Monitor Processes*. Monitor Sensors are positioned at strategic locations to measure services. These sensors must have direct access to local variables of host machines of a provider, as well as a direct connection to any consumers and all communication in between. Sensors are passive in the regard that they should not take action or affect the local system until receiving a request from a Monitor Process. When a request is received, a sensor becomes active, performs a measurement, returns the results and then becomes inactive again.

The bulk of monitoring logic is stored in a separate Monitor Process. The responsibilities of the Monitor Process are: (1) identify which SLAs require monitoring; (2) request measurements from Monitor Sensors; (3) check results for

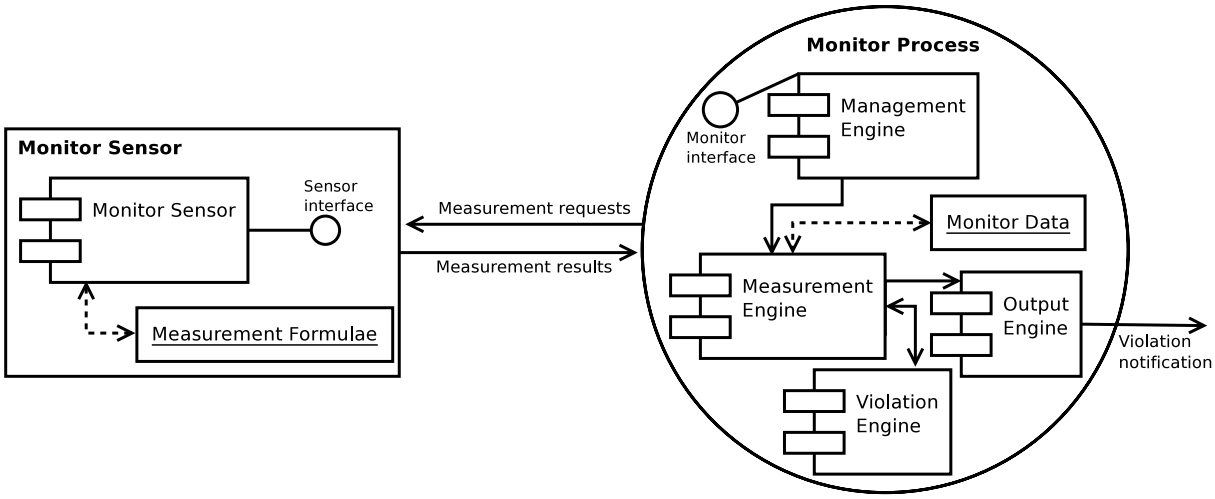

Figure 2. Monitor deployment

violations, and (4) take appropriate action when a violation is detected. The design of the Monitor Sensor and Monitor Process is illustrated in Figure 2.

Each Monitor Sensor provides an interface for communication with a Monitor Process and a local library of measurement formulae. A Monitor Sensor listens for measurement requests, loads measurement formulae from a local library, performs the measurement and returns the results.

A Monitor Process has an interface component and four engines: a Management Engine, a Measurement Engine, a Violation Engine and an Output Engine. An interface component receives new SLAs, sends measurement requests and receives results. A single Management Engine module coordinates and stores the SLAs, measurement results and recorded violations. This module creates a new Measurement Engine for each active SLA. The Measurement Engine receives the SLA information and begins performing measurements and testing results for violations based on the negotiated violation policy. If a violation is detected, the Violation Engine is notified. This module can take action as specified by a violation policy, such as contacting the Output Engine to inform one or more parties.

When an active SLA is received, a Monitor Process checks if any of the clauses require monitoring and, where appropriate, begins monitoring these items by communicating with sensors. These measurements can include static facts, such as the presence or absence of a required item (for example, a boolean value, such as '*host is reachable*'). These measurements can also include dynamic items that require aggregated data to calculate the average, minimum, maximum or complex functions (for example, that return a real number, such as '*average uptime is 99.9%*'). If a violation is detected during active monitoring, action is taken, as specified by the SLA. When the SLA ends, the results are stored, along with the communications log, in encrypted form for possible later auditing or conflict resolution. Additional mechanisms are available to secure this data, such as append-only storage [14].

B. Policy Specification

The current WS-Agreement specification [1] contains a *BusinessValueList* by which the value of a certain SLO can be expressed. This can declare its value explicitly or imply its value through a penalty or reward type. For instance, a penalty or reward type may hold a monetary value that indicates the importance of the SLO. Although this enables a basic mechanism for punishing poor performance and rewarding good performance, this paper proposes the use of a richer and more flexible method to specify violation policies. As opposed to adding more terms to the WS-Agreement standard, this paper proposes adding a separate SDT to specify these policies. This includes the ability to choose a violation policy, such as those mentioned above, as well as the number of acceptable violations and the actions that should be taken.

A portion of this extension is shown in Figure 3. The value of *ViolationPolicy* specifies the violation policy and can be *none*, *allornothing*, *partial* or *weighted*. The value of *ViolationCount* specifies how many violations are detected before action is taken. This must be a positive integer. The value of *ViolationAction* specifies the action to be taken when a violation is detected and can be *none*, *notify*, *penalise* or *cancel*. This allows for no action, notification of the parties, penalty enforcement (as specified by existing penalty clause) or cancellation of the service.

C. Security and Reliability

To provide reliable measurements, the monitoring process must be secure against malicious users that attempt to violate agreements or interfere with the provision of services to

```
<ServiceDescriptionTerm ServiceName="SLAMon">
  <Policies xsi:type="Policies">
    <ViolationPolicy/>
    <ViolationCount/>
    <ViolationAction/>
  </Policies>
</ServiceDescriptionTerm>
```

Figure 3. Monitoring policy attributes

others. Thus, the data collected from monitoring should be protected from deletion or modification by unauthorised users. Additionally, audit logs should be used, when necessary, to distinguish violations caused by parties of the agreement from those caused by other factors or users external to the agreement. Furthermore, monitoring should rely as little as possible on data provided by parties and should attempt to rely as much as possible on measurements of independently accessible variables.

The monitor should also be reliable and robust to system failure and overload. Distribution of the monitoring process can usually support both of these requirements. A distributed monitor can remove the risk of a single point of failure. Furthermore, distribution of the monitoring process can remove possible performance bottlenecks and allow for greater scaling. The monitoring process should be self-healing in two ways. Firstly, workload should be automatically balanced across monitors, preventing any single monitor from becoming overloaded. Secondly, monitors should automatically recover from failures and these failures should not affect the accuracy or integrity of the data that has already been collected.

V. Monitoring in AgentScape

AgentScape has been used for implementation and experimentation. The AgentScape middleware [15] platform supports SLA negotiation using the WS-Agreement specification [16]. Moreover, the AgentScape architecture fulfills the requirements for Trusted Computing Base [17], including clearly defined and enforceable security policies, role-based access controls, identification and authentication mechanisms and audit trails [18]. These features make it possible to serve as the TTP. This section introduces the AgentScape middleware and describes how the monitoring framework for WS-Agreement is implemented in AgentScape. This includes an experimental comparison of a centralised and decentralised variant to illustrate some of the benefits of a distributed approach.

A. AgentScape Middleware

AgentScape [15] is a distributed platform for heterogeneous mobile agents designed to be open, scalable, secure and fault-tolerant, the structure of which is depicted in Figure 4. An *AgentScape Location* is an administrative domain that groups one or more host machines together. Each AgentScape Location has a *Location Manager* that regulates access to a location and its resources. These resources may include middleware services such as *Agent Servers* for different programming languages (for example, Java, C, Jason) or an anonymity service. Each host machine has a *Host Manager* that regulates access to a host and its resources. A *Web Service Gateway* provides access to external web services. A group of independent AgentScape Locations that are aware of, and accessible to one another, may be referred to as an *AgentScape World*. This awareness is facilitated by an external *Lookup Service* that is responsible for providing listings of known locations and, optionally, their services.

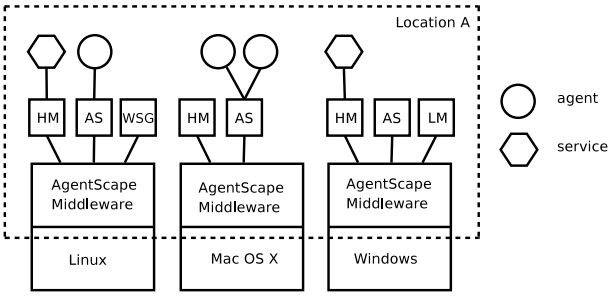

Figure 4. AgentScape architecture

An SLA framework implemented in AgentScape [16] allows agents to negotiate access to a site (that is, a location), the right to migrate to a site and access resources. In practice, agents negotiate leases using the WS-Agreement specification with the Location Managers. After checking an agent's credentials, a Location Manager takes note of an agent's requests and proposes combinations of services in an SLA, as described above. Agents can choose to accept a proposal or not. A Location Manager is, in fact, a *mediator*, that not only interacts with agents, it also interacts with its own hosts to determine which services it can provide. When an agreement is reached, the SLA is stored by the Location Manager, and an agent is provided access to a location and the services specified in the lease for the period of time depicted.

AgentScape is well-suited to distributed environments and is self-healing in the presence of failures. Mechanisms are in place to automatically recover crashed agents without losing data. Data persistence is achieved by storing data in the agent's container stored on the hard disk [19].

This framework has been used for service negotiation applications for both the energy domain as well as the Grid resource management domains [16].

B. Design Implementation

AgentScape currently supports negotiation and creation of SLAs using the WS-Agreement specification [16] as described above. The SLA framework in AgentScape has been extended to support monitoring and penalty enforcement. A

trusted monitoring module has been implemented to measure the provided services and ensure that the GTs in the SLA are fulfilled by both parties. The SLA document has been extended, similar to those described in [20]: the item to be measured, time constraints, and the method to be used for measurement as described in Example 1 below are included.

Example 1
An SLA specifies that measuredItem *is network latency,* measuredAt *is every 10 seconds,* evalFunc *is average latency is less than 50 milliseconds and* evalAction *is to warn customer if over limit three times in a row. Thus, the customer is alerted if average latency is consistently higher than 50 milliseconds.* △

As average user do not necessarily have trusted hardware modules, our implementation makes use of a TTP to achieve secure and reliable monitoring. The AgentScape middleware is the TTP. The middleware is impartial to the process and has no stake in the outcome of a consumer or provider. Therefore, when a host manager declares that it has a certain service to provide, or when it declares that a violation has occurred based on evidence, these are valid and honest actions.

Additionally, this approach assumes that all work is performed locally for both consumer and provider. The AgentScape SLA framework requires that consumer agents migrate to the host of the provider to use that provider's services. Therefore, measurements performed by either the provider, consumer or a TTP should be identical.

The monitoring process has been implemented as an AgentScape Agent. This allows existing mechanisms to be used for secure communication, access controls and persistent storage of data. Agents are automatically restarted with their current data if they die unintentionally or even if the middleware is restarted. Data persistence is achieved by routinely writing crucial information, such as the state of the SLA, to the agent container.

C. Centralised and Decentralised Monitoring

Two versions of the Monitor Agent have been designed and implemented: one centralised and one decentralised. The centralised variant utilises a Monitor Agent on only one Location for an entire AgentScape World. The decentralised variant utilises a Monitor Agent on each Location. These two variants are contrasted in Figure 5. In this illustration, three Locations form a single AgentScape World. In the centralised variant, only the Monitor Agent at Location A communicates directly with all Monitor Sensors at Locations where monitoring is needed. In the decentralised variant, a Monitor Agent is located at each Location and communicates only with Monitor Sensors at the same Location.

The centralised variant simplifies the management of leases by storing all information in a single place. In the decentralised version, each host only stores information regarding local leases and the load of monitoring responsibility is more evenly balanced across the system. If a monitor fails, all other monitors continue to operate normally. New SLA requests are sent to these monitors. Monitors are automatically respawned by the middleware once a failure is detected. Active SLAs and monitoring data are recovered from the persistent container file and monitoring continues.

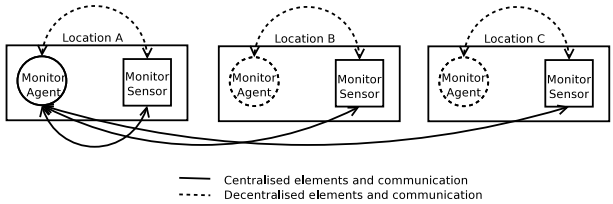

Figure 5. Centralised and decentralised monitor variants

Both variants have costs that cannot be known without experimentation.

D. Experiments

Experimentation tests a monitor's ability to detect violations following the specified policy (e.g. those introduced in Section III-C). These experiments use a web service external to AgentScape as the monitored service, and network latency and network traffic as the monitored items. Both provider and consumer violations are detected. When network latency to this web service is greater than the maximum agreed value, as specified by the SLA, this condition is interpreted by the monitor as a violation for which the provider is responsible. When network traffic is greater than the maximum agreed value, this is interpreted as a violation for which the consumer is responsible. In both cases, the violation is recorded and the parties are notified.

Further experimentation measures the overhead generated by the monitoring framework to judge the feasibility of such a framework in a real-world setting. This experiment compares the centralised and decentralised versions of the monitor. The centralised version uses only one monitor per AgentScape World, thus several Locations share the same monitor. The decentralised version uses one monitor per Location. One AgentScape World is created with five Locations. Each Location runs on a separate physical host machine. Measurements of the number of threads used and the amount of memory being consumed are taken as agents are added and evenly distributed across the Locations. The comparison of the centralised and decentralised versions is depicted in Figure 6.

As these results show, the decentralised version of the monitor utilises slightly more threads than the centralised version. The number of threads for both monitor versions increases at approximately the same rate as more consumer agents are added. The results also show that despite initially

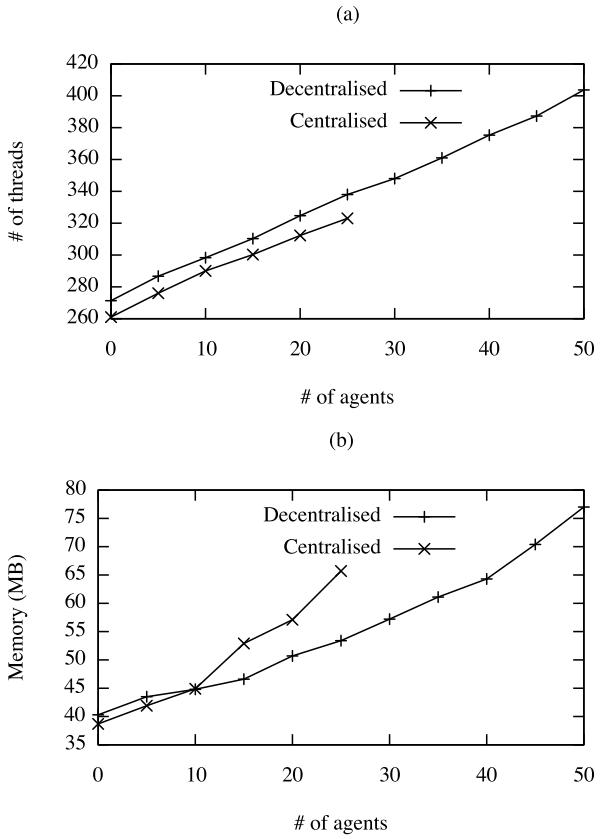

Figure 6. Comparison of centralised and decentralised monitor. (a) shows the number of threads used as the number of agents increases. (b) shows the amount of memory consumed as the number of agents increases.

using less memory, the centralised monitor memory consumption increases sharply as agents are added. In contrast, the decentralised version experiences near linear growth as load increases.

The experiments reveal information about the scalability of both the centralised and decentralised monitor. The decentralised version requires more resources in a predictable way, as load increases. However, the centralised version requires substantially more memory as the number of agents increase, until a hard limit is reached. After this point, adding more agents causes the system to become unresponsive. The decentralised version does not exhibit this characteristic and easily handles twice the load of the centralised version.

In summary, a centralised and a decentralised SLA monitoring framework has been implemented in AgentScape to judge the feasibility of such a framework in a real-world setting. These variants differ in the complexity of information management. The centralised variant with centralised lease management, suffers from performance problems when the number of agents increases. In contrast, the decentralised version utilises a more complex distributed lease management system and scales in a predictable and stable way.

VI. Related Work

The field of monitoring agreements is an active area of research. A number of alternative approaches to this problem are available, including the WSLA framework from IBM [5] and Grid-oriented approaches [3], [20]. These projects offer insights to the considerations and design of monitoring frameworks.

The Web Service Level Agreement (WSLA) framework [5] is a framework for specifying and monitoring agreements. This specification provides a very rich language for specifying monitoring requirements; however, one important drawback is that it does not support the multiple violation policies discussed in this paper. Furthermore, this specification does not appear to be widely used or actively developed. In contrast, the WS-Agreement specification, used in this framework, is currently being used and developed by an active community [8], [6], [21], [22]. This offers a greater chance of incorporation of the extensions recommended in this paper.

Two Grid-oriented approaches [3], [20] also include support for monitoring SLA clauses. Although these approaches provide methods for automated measurements and violation capture for distributed environments, both appear to rely on the assumption of a trusted environment, namely that of a commercial Grid. This assumes that internal resources are protected from external attack and that users are not able to attack the system to prevent detection of their own violations or to cause the violation of other innocent users. Furthermore, these grid-oriented approaches use dedicated machines for monitoring and logging. While these approaches may be well-suited to a controlled environment, this will not scale to open environments, such as the Internet.

In contrast, the approach described in this paper does not assume a trusted network, but rather expects malicious users to attack the system. Furthermore, this approach pursues fully distributing the load of users and the risk of failure across multiple hosts, providing scalability and reliability.

VII. Discussion

While negotiation of SLAs have been examined in detail [16], monitoring and determining how to address violations of these agreements have not received the same level of attention. This paper presents a framework for monitoring SLAs that is secure and reliable even in the presence of malicious users, failures and heavy load. Furthermore, the proposed framework has a broad definition of violation and what penalties should be enforced.

This monitoring framework has been implemented in AgentScape. The AgentScape middleware has established security mechanisms and thus serves as a trusted third party. A centralised and a decentralised version of the monitoring framework have been implemented and compared.

Future work in this area will include incorporating trusted hardware modules, when available, and making use of reac-

tive monitoring mechanisms [23] to lower overhead costs. While the results of experimentation with the monitoring framework has indicated the suitabilty of a decentalised approach for monitoring SLAs under heavy load, further experimentation is required to investigate more security and reliability issues, such as active attacks against monitors. While the current implementation provides a working basis, more work is also required to standardise penalties and proof of violations.

Acknowledgements

This work is a result of support provided by the NLnet Foundation (http://www.nlnet.nl) and the ALIVE project (FP7-IST-215890).

REFERENCES

[1] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement). In *Global Grid Forum GRAAP-WG, Draft, August*, 2004.

[2] 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., August 2008. Springer Verlag.

[3] J. Padget, K. Djemame, and P. Dew. Grid-Based SLA Management. *Lecture Notes in Computer Science*, 3470:1076, 2005.

[4] A. Ludwig, P. Braun, R. Kowalczyk, and B. Franczyk. A Framework for Automated Negotiation of Service Level Agreements in Services Grids. *Lecture Notes in Computer Science*, 3812:89, 2006.

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

[6] M. Aiello, G. Frankova, and D. Malfatti. What's in an Agreement? An Analysis and an Extension of WS-Agreement. *Lecture Notes in Computer Science*, 3826:424, 2005.

[7] A. Pichot, P. Wieder, O. Waldrich, 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.

[8] 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.

[9] L. Gymnopoulos, S. Dritsas, S. Gritzalis, and C. Lambrinoudakis. GRID security review. *Lecture Notes in Computer Science*, pages 100–111, 2003.

[10] F. Lau, SH Rubin, MH Smith, and L. Trajkovic. Distributed denial of service attacks. In *2000 IEEE International Conference on Systems, Man, and Cybernetics*, volume 3, 2000.

[11] R. Mach, R. Lepro-Metz, S. Jackson, and L. McGinnis. Usage Record Format Recommendation. In *Draft Rec-UR-Usage, Global Grid Forum, Usage Record WG*, 2005.

[12] TB Quillinan, BC Clayton, and SN Foley. GridAdmin: Decentralising grid administration using trust management. In *Parallel and Distributed Computing, 2004. Third International Symposium on/Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, 2004. Third International Workshop on*, pages 184–192, 2004.

[13] A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, 2007.

[14] Y. Wang and Y. Zheng. Fast and Secure Append-Only Storage with Infinite Capacity. In *Second IEEE International Security in Storage Workshop*, pages 11–19. IEEE Computer Society, 2003.

[15] B.J. Overeinder and F.M.T. Brazier. Scalable Middleware Environment for Agent-Based Internet Applications. *Lecture Notes in Computer Science*, 3732:675, 2005.

[16] D. G. A. Mobach. *Agent-Based Mediated Service Negotiation*. PhD thesis, Computer Science Department, Vrije Universiteit Amsterdam, May 2007.

[17] D.C. Latham. Department of Defense Trusted Computer System Evaluation Criteria. *Department of Defense*, 1986.

[18] T. B. Quillinan, M. Warnier, M. A. Oey, R. J. Timmer, and F. M. T. Brazier. Enforcing Security in the AgentScape Middleware. In *Proceedings of the 1st International Workshop on Middleware Security (MidSec)*. ACM, December 2008.

[19] G. van 't Noordende, B. J. Overeinder, R. J. Timmer, F. M. T. Brazier, and A. S. Tanenbaum. Constructing secure mobile agent systems using the agent operating system. *International Journal of Intelligent Information and Database Systems*, 3(4), 2009.

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

[21] G. Di Modica, V. Regalbuto, O. Tomarchio, and L. Vita. Enabling re-negotiations of SLA by extending the WS-Agreement specification. In *IEEE International Conference on Services Computing, 2007. SCC 2007*, pages 248–251, 2007.

[22] G. Frankova, D. Malfatti, and M. Aiello. Semantics and Extensions of WS-Agreement. *Journal of Software*, 1(1), 2006.

[23] D. Khader, J. Padget, and M. Warnier. Reactive monitoring of service level agreements. In *Workshop on Service Level Agreements 2009*, 2009.