Self-adaptive Service Monitoring

Kassidy Clark, Martijn Warnier, Frances M.T. Brazier

Faculty of Technology, Policy and Management
Delft University of Technology, The Netherlands
[K.P.Clark, M.E.Warnier, F.M.Brazier]@tudelft.nl

Abstract. Online marketplaces are emerging in which services are provided and consumed. Parties make online agreements regarding the terms and conditions of service provisioning. For certain kinds of services, it may be necessary to know whether it is being provisioned according to the agreement. To this end, the service may be monitored. For instance, a web application service may be monitored to guarantee that the response time of the application is within acceptable limits. The decision of whether or not to monitor a service is correlated to the perceived level of risk that a violation will occur. If there is a low level of perceived risk, monitoring may not be required, and vice versa. The perceived level of risk associated with a service transaction can change over time. However, traditional monitoring techniques are not able to react to this change. This paper proposes a self-adaptive service monitor that adapts to changes in the perceived level of risk. This monitor combines a traditional service monitor with a self-monitoring protocol, referred to as *passive monitoring.* This monitor is implemented in the AgentScape Middleware.

1 Introduction

Online marketplaces are emerging in which services are provided and consumed. In these marketplaces, parties can advertise, negotiate and purchase services. These services include access to hardware, such as storage or compute power and access to software, such as databases or other applications. Examples of such marketplaces include the Grid services market [2], the Web services markets [4] and the Cloud services market [3].

Parties negotiate the terms and conditions of their services, including the price, Quality of Service and other service obligations. These terms and conditions are embodied in an agreement. This agreement also specifies what actions should be taken if any party violates the agreement. To know if, when and by whom an agreement is violated, services can be monitored. Relevant service metrics (e.g. response time) are measured at periodic intervals to offer assurance that a service is being provided as promised. These measurements are performed either by each party individually or by a separate monitoring service.

In some cases, a Trusted Third Party (TTP) is used as a separate monitoring service. The TTP offers additional assurance that measurements are performed impartially. Dependence on a separate monitoring service, such as a TTP, can be problematic. For instance, it may not always be preferable to give external parties access to local resources and (sensitive) data. However, monitoring services require this access to perform measurements. Moreover, monitoring services generate additional overhead costs (e.g. CPU, messages, and so forth). Furthermore, centralized monitoring services form an obstacle to the scalability of the marketplace. Reducing dependence on a separate monitoring service allows the system to scale better.

It might also be the case that parties do not feel that monitoring is always necessary. For instance, if the parties' past experience with one another has given them a high level of trust in each other, the perceived risk that there will be a violation is very low. In this case, monitoring just produces unnecessary overhead costs. Parties should be able to adjust the monitor. This makes it possible to reduce monitoring overhead when the perceived level of risk is low, and vice versa.

Monitoring systems and services in distributed environments is an area of ongoing research. Monitoring is used for Grid and Cloud environments to monitor a wide spectrum of metrics, from low-level hardware health to high-level service compliance [15]. Two of the most well known monitoring systems are Nagios [11] and Ganglia [9]. Both of these systems offer a wide range of configurable monitoring options, including the ability to modify the interval between measurements. However, these systems were not designed to adjust the level of monitoring dynamically or autonomously.

Some monitoring services are able to dynamically adapt their monitoring policy at run-time based on environmental limits or changes in priorities. For instance, the monitor proposed in [6] collects system notifications from distributed nodes and can dynamically adjust the frequency of the 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 a dynamic monitor is the adaptive system monitor described in [10]. This monitor 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 monitor adapt by increasing the number of related metrics that are continuously monitored. Effectively, this monitor is able to 'zoom in and out' of areas when problems are detected.

This paper proposes an alternative self-adaptive monitoring technique that reacts to changes in the perceived level of risk by dynamically adjusting the level of monitoring. This monitor combines a traditional monitoring service with a self-monitoring protocol, referred to as *passive monitoring* [7]. In addition to switching between these two modes, the self-adaptive monitor is able to decrease the frequency of measurements to decrease overhead, or increase the frequency to offer higher assurance.

The self-adaptive monitor proposed in this paper differs from the approaches discussed above in two major ways. First, in addition to reacting to changes in system overhead, the motivation to modify the monitoring policy is to dynamically adapt to the level of perceived risk as it changes during service provisioning. Secondly, rather than only adjusting the measurement interval, the monitor is able to switch between two major modes of monitoring: active and passive. In the passive mode, dependence on a separate monitoring service decreases. An effect of this reduced dependence is that the system is able to scale.

This paper is organized in the following way. Section 2 explains both traditional monitoring and the self-monitoring protocol, referred to as *passive monitoring*. Section 3 describes the proposed self-adaptive monitor in more detail. Section 4 presents the adaptation model used to determine when and how to adjust the monitor. This includes a more detailed explanation of the monitoring architecture. Section 5 evaluates an implementation of the self-adaptive monitor in the AgentScape Middleware. Finally, Section 6 concludes the paper.

2 Monitoring Techniques

Monitoring can be performed either by each individual party or by a separate monitoring service. Most traditional monitoring services periodically test various metrics to determine if the system is operating as expected [9,11]. This mode of monitoring is referred to in the remainder of this paper as *active monitoring*. Using this technique, an impartial monitoring service (e.g. TTP) takes measurements on behalf of the parties. If a violation is detected, the monitoring service can take action. Such action could be to cancel the service or penalize the offending party. The chosen action depends on the policy agreed upon by the parties during service negotiation.

An alternative to active monitoring is a *passive monitoring* [7]. When using passive monitoring, each party performs their own measurements. Therefore, this monitoring technique does not require or depend on a separate monitoring service. Essentially, passive monitoring uses cryptographic primitives to build a secure, non-repudiable audit log. Each party to the agreement must commit to add an entry to the audit log. For instance, all parties must be satisfied with the current level of service before an entry can be added. However, if one party is not satisfied, no entry is added. Therefore, this protocol is also referred to as a *mutual commit* monitoring protocol. This section contains a general overview of the protocol. The full protocol, including conflict mediation is detailed in [7].

Each party is responsible for local service measurements and no external monitoring service is used, such as a Trusted Third Party (TTP). Service provisioning is divided into discrete intervals. For provisioning to continue for the next interval, all parties of the agreement must agree that they are thus far satisfied with the service and have not detected any violations. Once all parties agree, a token (e.g. password) is exchanged. The token is cryptographically encrypted and signed to ensure that, once sent and received, no party can deny sending or receiving the token. These tokens are aggregated using other cryptographic primitives to form an audit log of compliance.

If no violations are detected, no intervention is needed. Tokens are exchanged directly between the parties, so no external monitoring services are required. This reduces the costs normally associated with interaction with a separate monitoring service. However, if a party detects a violation, conflict mediation must be performed. There are several possible ways to perform conflict mediation. In the simplest case, the service is immediately canceled. Another option is that the parties work together to examine the audit log to determine which party is responsible for the violation. Optionally, the examination of the audit log can also be performed by a TTP.

The conflict mediation process requires the audit log from each party, along with the last two messages from the interval in dispute. Using the non-repudiation properties of the messages, the last point can be established at which the parties were satisfied with the service. For example, if the service spans 10 intervals and the violation was detected in the last interval, any penalties are limited to the last interval. One possible result of conflict mediation can be to switch to active monitoring and continue the service. This is possible with self-adaptive monitoring.

3 Self-adaptive Monitoring

Monitoring offers assurance that violations are detected. Some service transactions require more assurance than others. The level of assurance required is a reflection of the perceived level of risk associated with the service. For instance, the importance of detecting a violation in a mission critical service is different from that of detecting a violation in a non-mission critical service. These violations ultimately have different levels of impact (e.g. financial, operational and so forth). Different services have different perceived levels of risk and therefore have different monitoring requirements. To this end, monitoring can be expressed in the 'amount' or 'level' of monitoring. In general, a high level of monitoring equates to frequent measurements of all relevant metrics. In contrast, a low level of monitoring equates to less frequent measurements of a subset of the metrics.

Furthermore, levels of trust and perceived risk between parties are not constant and can change during the course of their interactions. In this context, trust is defined as a combination of several metrics used in electronic markets [16,8,5]. These include personal metrics, such as transaction history, transaction cost and the ability to verify results. These also include community metrics, such as the popularity, activity and position of a party in the community. These metrics are dynamic in that they change over time. For instance, a successful transaction 10 years ago has less impact on the trust level than a successful transaction yesterday.

Reputation is also an important factor in determining trust levels. This is often the case when determining initial trust levels. For instance, reputation is used when a consumer chooses a service provider [13]. If the provider is well-known (e.g. Amazon Web Services), this increases the initial trust the consumer has and lowers the perceived risk of doing business. Once service provisioning has begun, a consumer dynamically adjusts the level of perceived risk based on the number of successful transactions and the size of transactions (e.g. 1 dollar versus 1000 dollars).

The first time two parties do business together, there is often a low level of trust and thus a high level of perceived risk. This has the effect that both parties may need a high level of assurance from monitoring and accept the extra costs it generates. As these parties interact with one another, they build a relationship based on their experience. If these experiences are bad, in that the parties fail to meet their agreement obligations, the level of trust remains low and perceived risk remains high. However, if these experiences are good, in that parties honor their obligations, the level of trust increases. As the level of trust increases, perceived risk and the need for monitoring decreases.

When using a separate monitoring service, such as [11,9], a single, static monitoring policy is used for the entire session that cannot reflect the dynamic changes in the levels of trust and perceived risk. However, changes in the level of perceived risk between parties can be immediately and directly reflected in the self-adaptive monitor. This is accomplished by dynamically switching between active and passive modes, as well as increasing or decreasing the measurement interval. In effect, these changes increase the level of assurance when needed, or reduce overhead and dependence on the TTP when possible.

4 Adaptation model

The choice of which monitoring level to use for a particular transaction is based on a risk function. This function takes the current *risk level* and a *monitoring policy* and chooses an appropriate monitoring level. The monitoring level is expressed with the mode (e.g. active, passive) and the interval (e.g. time between measurements). Each party to an agreement executes this function before a transaction to determine which level of perceived risk applies, and therefore which level of monitoring is required. In the case that the levels of monitoring required by two or more parties differs, the highest level is used. This guarantees that all parties have at least the minimum level of assurance required.

This model is independent of the particular method used to compute a party's perceived risk level. An example of how the perceived risk level can be computed is illustrated below.

4.1 Risk level

The first input to the adaptive monitoring function is the current perceived risk level. The perceived risk level is determined using both knowledge from the environment and local knowledge. Environmental knowledge can be a reputation authority, such as that proposed in [16], that offers additional information about a particular party, including their activity and popularity in the environment.

Local knowledge includes the price (or cost) of the current transaction and the history (if any) of transactions with the given party. These two variables are combined to form a matrix, similar to the trust matrix presented in [8]. The *transaction cost* of the current transaction and the *transaction history* correspond to a level of perceived risk. Essentially, the higher the cost of a transaction, the higher the perceived risk level. Conversely, the better the transaction history, the lower the perceived risk level.

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. For instance, for a particular party any transaction below 100 euro corresponds to a cost of 1. Whereas transactions between 100 and 200 euro correspond to a cost of 2 and so on. This mapping is specific to each individual party.

Transaction history is 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 discussed in [16]. In this context, this means that the 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.

4.2 Monitoring policy

Each party maintains their own policy that specifies which monitoring mode and interval correspond to which level of perceived risk. For instance, for a particular party, an risk level of 1 may correspond to a low level of monitoring, such as passive mode with an interval of 90 seconds. Whereas an risk level of 10 may correspond to a high level of monitoring, such as active mode with an interval of 5 seconds. Each party's policy therefore specifies this mapping between the level of perceived risk and the level of monitoring.

Additionally, the policy specifies the particular weights that an agent attaches to each element of knowledge. Weights can be used to indicate how important an element of knowledge is based on its age or type. For instance, the 10 most recent transactions are more important than the rest. Weights are also used to indicate the importance of local knowledge (e.g. transaction history, transaction cost) and additional environmental knowledge (e.g. central reputation authority). The policy also contains a threshold to indicate how much the perceived risk level is able to change before the monitor is adapted. In effect, this number allows the monitor to react immediately to any change in perceived risk (e.g. paranoid) or assume that slight changes in the perceived risk level are anomalies and therefore react only if perceived risk consistently increases or decreases (e.g. optimistic). Finally, the policy contains additional information, including an optional name that describes the policy (e.g. optimistic, paranoid and so forth).

Two example policies are illustrated in Figure 1. The *Paranoid* policy has a low reaction threshold and thus quickly adapts the monitor to any changes in the risk level. This policy also gives a larger weight to the most recent 10 transactions. Furthermore, this policy is more sensitive to transaction cost than history. In contrast, the *Optimistic* policy has a higher reaction threshold and thus allows more variation in the perceived risk level before adapting the monitor. This policy balances the weight of the most recent transactions with older transactions. Furthermore, this policy considers the history of transactions more important than the cost of a particular transaction.

```
# context #
PolicyName=paranoid
ReactionThreshold=1

# age weighting #
10MostRecentTransactions=0.8
100MostRecentTransactions=0.1
...

# type weighting #
TransactionHistory=0.2
TransactionCost=0.8

# risk level mapping #
RiskLevel1Mode=passive
RiskLevel1Interval=90
...
RiskLevel10Mode=active
RiskLevel10Interval=5
```

(a) Paranoid

```
# context #
PolicyName=optimistic
ReactionThreshold=3

# age weighting #
10MostRecentTransactions=0.3
100MostRecentTransactions=0.5
...

# type weighting #
TransactionHistory=0.8
TransactionCost=0.2

# risk level mapping #
RiskLevel1Mode=passive
RiskLevel1Interval=90
...
RiskLevel10Mode=active
RiskLevel10Interval=20
```

(b) Optimistic

Fig. 1: Examples of (a) paranoid and (b) optimistic monitor policies.

The policy allows each party to customize their monitoring needs, based on their individual perception of risk. The policy may also change over time. For instance, a party may choose a very paranoid policy when joining an online marketplace for the first time. After gaining experience in this marketplace, the party may choose to modify the policy to reflect the lessons learned (e.g. raise the threshold). In this regard, the self-adaptive monitor can also be manually adjusted, if necessary.

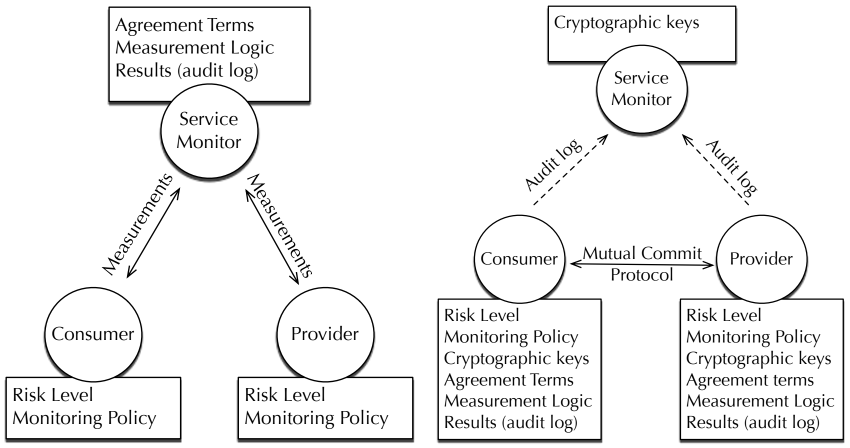
4.3 Monitoring Architecture

There are three main parties to a service agreement: the service provider, the service consumer and the service monitor. Each of these parties requires certain data depending on the monitoring mode: active or passive. Furthermore, communication patterns between parties differ depending on the current mode.

For active mode, shown in Figure 2a, the consumer and provider require only their unique Risk Level calculation and Monitoring Policy. The parties require no further monitoring data. The Service Monitor is responsible for performing measurements and checking for agreement violations. Therefore, this service must have access to the Agreement Terms, the Measurement Logic (e.g. which variables must be tested) and the Results of the measurements. All monitoring communication is initiated by the Service Monitor directly to each party individually.

In contrast, passive mode shifts the monitoring responsibility to the consumer and provider. In this mode, shown in Figure 2b, the Service Monitor requires no knowledge of the Agreement Terms or Measurement Logic. Furthermore, the Results (e.g. audit log) are no longer stored at the Service Monitor, but rather at each of the parties to the agreement. Additionally, the Agreement Terms, Measurement Logic and Results are stored at each individual party, rather than at the Service Monitor. All monitoring communication goes directly between the Consumer and Provider. Only in the case of mediation do the parties optionally communicate with the Service Monitor to send the required audit logs.

The only extra data required are the Cryptographic Keys. These are used by each party to enable the mutual commit protocol and encrypt the audit log. The Service Monitor also maintains a set of keys to enable mediation in the case that a violation is detected.



(a) Active mode (b) Passive mode

Fig. 2: Data and communication of (a) active and (b) passive monitoring.

5 Experimental Evaluation

The feasibility of the self-adaptive monitor is illustrated with an implementation. Several experiments are performed to compare the overhead of the active and passive modes. First, parties use a static active or a static passive monitor. Finally, parties use the self-adaptive monitor. Overhead is measured as the average CPU usage of monitoring activity.

The self-adaptive monitor is implemented in the AgentScape distributed middleware [12]. In this environment, Consumers, Producers and Monitors (TTP) are represented by self-contained, mobile software agents. This implementation performs asymmetric cryptography using the Rivest-Shamir-Adleman (RSA) [14] algorithm and the Boneh-Lynn-Shacham (BLS) [1] algorithm.[1]

[1] The BLS implementation uses existing code provided by Dalia Khader and 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.

Experiments run on 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 the Linux operating system and AgentScape middleware. Two AgentScape Locations are used, one on each machine: *Location C* and *Location P*. Consumer agents are loaded into Location C and Producer agents into Location P.

5.1 Experimental Setup

Three experiments are performed in total: *Baseline A, Baseline P* and *Self-Adaptive*. The first two experiments compare the baseline CPU usage of each monitoring mode (active and passive) separately. The third experiment shows the changes in CPU usage as the monitor dynamically switches between intervals and modes. An overview of these experiments is shown in Table 1.

Table 1: Overview of experiments

Experiment	**Monitor**	**# of Agents**	**Interval**
Baseline A	Active	2	10
Baseline P	Passive	2	60
Self-Adaptive	Active & Passive	2	10,20,60,90

In *Baseline A*, a single consumer and a single provider negotiate an agreement that uses an active monitor with an interval of 10 seconds. In *Baseline P*, a single consumer and a single provider negotiate an agreement that uses a passive monitor with an interval of 60 seconds. No conflict mediation is performed during these experiments. Each baseline experiment is repeated 10 times and the CPU load results are averaged.

In *Self-Adaptive*, one consumer and one provider negotiate an agreement that begins with an active monitor with an interval of 10 seconds. Both parties use the same monitoring policy. This policy specifies that after 5 minutes without violations, the Risk Level decreases. This decrease is reflected in the monitoring level by increasing the interval to 20 seconds. After another 5 minutes without violations, the Risk Level decreases further. This decrease is reflected in the monitoring level by changing the mode to passive with an interval of 60 seconds. After another 5 minutes without violation, the Risk Level decreases further still. This decrease is reflected in the monitoring level by increasing the interval to 90 seconds. When a violation is detected, mediation is requested. The result of mediation is to increase the Risk Level. This increase is reflected in the monitoring level by returning to active mode with an interval of 10 seconds. The self-adaptive experiment is repeated 10 times and the CPU load results are averaged.

5.2 Experimental Results

The *Baseline* experiments, shown in Figure 3, demonstrate the contrast in CPU usage generated by the two different monitors. The active monitor has a consistent CPU usage baseline reflecting the 10 second monitoring interval. Usage peaks are relatively low, but occur often. This steady surge reflects the measurement request, execution, response and evaluation. The passive monitor also has a consistent CPU usage baseline reflecting the 60 second monitoring interval. Usage peaks are relatively high, when compared to the active monitor, but occur less frequently. This steady surge reflects the computationally intense cryptography that is used in the mutual commit monitoring protocol.

The *Self-Adaptive* experiment, shown in Figure 4, gives an overview of the self-adaptive monitoring process. In the active modes, the CPU usage has a consistent pattern reflecting the 10 and 20 second monitoring interval, respectively. In the passive modes, the CPU also has a consistent CPU usage that reflects the 60 and 90 second monitoring interval, respectively. A noticeable CPU usage surge occurs when conflict mediation is requested. This surge reflects the additional cryptography required to verify the messages and aggregate signatures used in the audit logs.

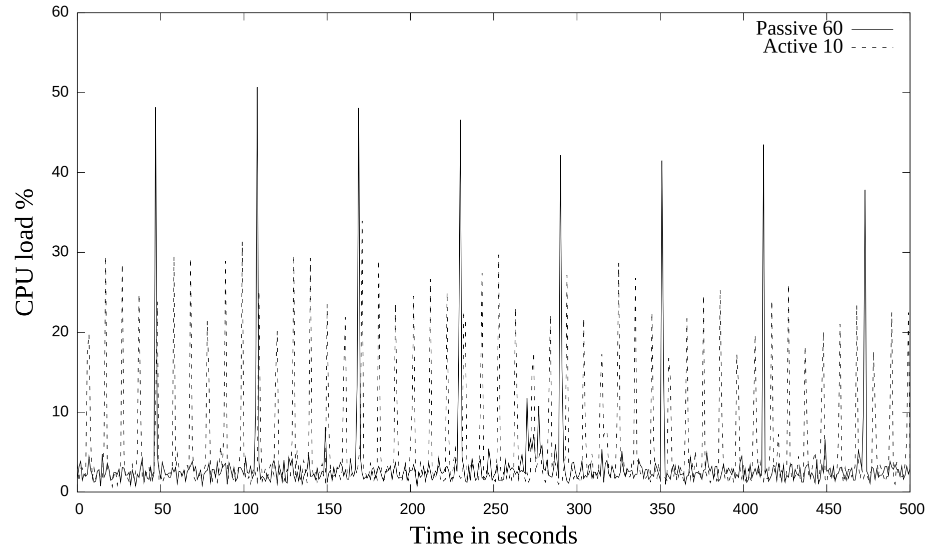


Fig. 3: CPU usage of active and passive monitor with two agents.

5.3 Discussion of Results

In this implementation, it is difficult to compare the monitoring modes, due to the different results that they produce. The passive mode produces a cryptographically secured audit log, whereas the active mode produces an audit log

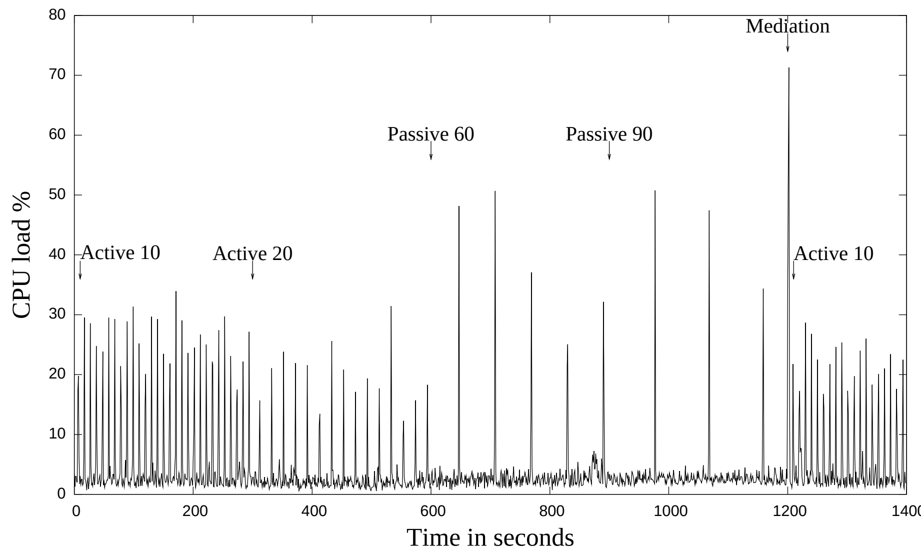


Fig. 4: CPU usage of self-adaptive monitor with two agents.

without any cryptographic security. The cryptography used in passive monitoring increases the computational overhead[2] as can be seen in Figures 3 and 4. In these figures, it can be seen that the peaks in CPU load associated with the passive monitor are higher than those associated with the active monitor. To reduce the impact of this cryptographic overhead, a larger interval is used for the passive mode, in this implementation.

Figure 4 also shows the relatively high computation required for conflict mediation. This is also due to cryptography required to verify the data stored in the audit log. When mediation occurs often, it may present a significant load to system resources. Therefore, the frequency of conflict mediation may influence the choice of monitoring mode. This consideration can be included in the monitoring policy.

6 Conclusion

This paper proposes a self-adaptive monitor that reacts to changes in the level of perceived risk by adjusting the level of monitoring. This monitor can automatically switch between two modes, active and passive, and automatically adjust the polling interval for both. This paper demonstrates how the perceived level of risk can be calculated and how different monitoring policies can be applied.

Future work will examine the scalability of the self-adaptive monitor in a larger, distributed environment. Other trust mechanisms, such as reputation authorities, will also be incorporated to quantify the level of trust between parties.

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

ACKNOWLEDGEMENTS

This work is a result of support provided by the NLnet Foundation (`http://www.nlnet.nl`).

References

1. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. Journal of Cryptology 17(4), 297–319 (2004)
2. Buyya, R., Vazhkudai, S.: Compute power market: Towards a market-oriented grid. In: ccgrid. p. 574. Published by the IEEE Computer Society (2001)
3. Buyya, R., Yeo, C., Venugopal, S.: Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In: High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on. pp. 5–13. Ieee (2008)
4. Cheng, S., Chang, C., Zhang, L., Kim, T.: Towards competitive web service market. In: Future Trends of Distributed Computing Systems, 2007. FTDCS'07. 11th IEEE International Workshop on. pp. 213–219. IEEE (2007)
5. Hsu, C.: Dominant factors for online trust. In: Cyberworlds, 2008 International Conference on. pp. 165 –172 (Sep 2008)
6. Keung, H., Dyson, J., Jarvis, S., Nudd, G.: Self-adaptive and self-optimising resource monitoring for dynamic grid environments. In: Database and Expert Systems Applications, 15th International Workshop on. pp. 689 – 693 (Aug 2004)
7. Khader, D., Padget, J., Warnier, M.: Book chapter in CoreGRID Springer Series, chap. Reactive Monitoring of Service Level Agreements, pp. 13–23. Springer-Verslag (2010)
8. Manchala, D.: E-commerce trust metrics and models. Internet Computing, IEEE 4(2), 36 –44 (Mar 2000)
9. Massie, M., Chun, B., Culler, D.: The ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing 30(7), 817–840 (2004)
10. Munawar, M., Ward, P.: Adaptive monitoring in enterprise software systems. Tackling Computer Systems Problems with Machine Learning (SysML) (2006)
11. NagiosEnterprises: Nagios - the industry standard in it infrastructure monitoring. `http://nagios.org/` (May 2011)
12. Overeinder, B., Brazier, F.: Scalable Middleware Environment for Agent-Based Internet Applications. Lecture Notes in Computer Science 3732, 675 (2005)
13. Resnick, P., Kuwabara, K., Zeckhauser, R., Friedman, E.: Reputation systems. Communications of the ACM 43(12), 45–48 (2000)
14. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21(2), 120–126 (1978)
15. Sahai, A., Machiraju, V., Sayal, M., van Moorsel, A., Casati, F., Jin, L.: Automated SLA Monitoring for Web Services. Lecture Notes in Computer Science pp. 28–41 (2002)
16. Tajeddine, A., Kayssi, A., Chehab, A., Artail, H.: A comprehensive reputation-based trust model for distributed systems. In: Security and Privacy for Emerging Areas in Communication Networks. Workshop of the 1st International Conference on. pp. 116 – 125 (Sep 2005)