Abstract— Computing resource provisioning through the use of the Cloud computing paradigm has triggered revolutions in modern day computing. It is a new paradigm for deploying services on rented machines. On the other hand, Service Oriented Architecture (SOA) has gained wide adoption among organizations due to the importance of collaborations and outsourcing. Therefore the Cloud’s enormous capacity with comparable low cost makes it an ideal platform for SOA deployment. The overall correctness of the SOA deployed in the Cloud depends on the correctness of all individual participants. As the SOA usually spans multiple administration domains, concluding the faulty service and making the provider responsible become a challenging task. In this paper, we propose a novel design to achieve Trustworthy Service Oriented Architecture (TSOA) in the Cloud through enforcing strong accountability. In such system not only the root of a fault can always be concluded to the guilty participant(s), each conclusion is supported with non-disputable evidence. We also implemented a demonstrative system to show its effectiveness in real practice. Our testing figure indicates the cost of incorporating our design to SOA in the cloud is acceptable.

Keywords—cloud computing; accountability; service orientated architecture; trustworthiness

I. INTRODUCTION

Computing resource provisioning through the adoption of the cloud computing concept has revolutionized service deployment. Rather than owning and maintaining dedicated computing and storage infrastructure, cloud computing provides the user with an outsourcing model, where the provider manages the client’s computing and storage requirements on a pay-as-you-go basis. This outsourcing model is attractive for businesses that wish to minimize their computing and storage infrastructure costs. Because it is offered as a service on the Internet, it also facilitates sharing of information and resources across and between enterprises and clients.

Similarly, the adoption of Service Oriented Architecture (SOA) as a design paradigm by organizations that wish to be flexible in their business either by offering their own services to others, or offering new value added services through the composition of their own services with other services offered by third parties [13].

The adoption of the two (cloud & SOA) result in a serviceful cloud computing environment that enables highly dynamic and effective organizational collaborations. In such collaboration, each of the participants would behave according to the predefined and mutually agreed upon business logic and Service Level Agreement (SLA). As far as the participants are in the collaboration, the correctness of the system operation depends on first, that the agreement and logic are correct, and second, that each participant complies at all times to them. Any deviation from this agreed upon is regarded as violations, and a robust mechanism is needed for its detection, logging and resolution. Once the cause of the violation is found, each violator is regarded as being accountable for its fault. The importance of maintaining participant behaviors according to the respective business logic and SLA is critically essential for such collaborative model to be adopted.

We regard the ability to detect violations as a critical component of developing a trusted computing environment. We adopt the informal definition of trust (based on comments made by Graham Proudler): something can be trusted when (a) it can be unambiguously identified; (b) it operates unhindered and (c) the user has either first hand experience of consistent good behaviour or knows someone who can vouch for consistent good behaviour. A more formal trust definition is presented in IETF RFC4949 Internet Security [12].

However, the detection and prevention of failures under a composed service is complicated by the fact that
composed services (and resultant system) usually span several administrative domains, each of which will have its own interests and priorities. Given the fact that an admission to a violation may cause penalties in some form, it is conceivable that an entity may intend to deceive and hide this fact. This represents a challenge and obstacle to the adoption of cloud computing for inter- and intra-enterprise collaborations.

Recent study of this issue has raised recognition that instead of conventional correctness assurance, trustworthiness plays a more important role in this scenario. Building on the notions of trust presented, a *trustworthy system* [14] is defined as: a system that is already trusted, and continues to warrants that trust because the system’s behaviors can be validated in some convincing way. In this paper, we propose a novel design to achieve Trustworthy Service Oriented Architecture (TSOA) by introducing the concept of *strong accountability*. In such a system, the root of a failure or misbehaviour can always be identified and associated with responsible (or guilty) entity(s), and supported by non-disputable evidence. Then, we propose a method to enforce such trustworthiness with standard definitions of business logic and service level agreements. Finally we deploy the system into a computing cloud to evaluate its effectiveness.

We use the above scenario to exemplify an ordinary business process with service composition. Each service node takes job request, finishes the job and reply with the result. In such process, a failure from any of the nodes would result complete failure at the final output, and usually the source of the failure may not always be able to be logically determined.

For instance, a loan applier, Bob, later on finds out that he can be offered a cheaper loan through direct contact with one of the loan companies (claimed to be involved by the bidding company), proving that the one-stop loan application service has failed its promised service outcome. Bob is not able to know if it is the credit rating authority which gave him a bad rating, or that loan company is actually not involved during the bidding. Intuitively, Bob may hold the one-stop loan application service responsible. At this point, the application service may attempt to alter its system record to prove its own innocence and push the blame to the bidding company. In turn, the bidding company could also do the same, and shift the blame to either the credit rating authority or to the loan companies. It can be seen even in this simple example that a mechanism is required to prevent this “buck passing” or denial of failure. This mechanism or service is essential to control the correctness of an SOA-based business process. Once a node is determined to have failed to meet expected behaviours, actions must be taken so that it can be excluded or replaced.

### III. ACCOUNTABILITY

Accountability is a concept to make the system accountable and trustworthy by binding each activity to the identity of its actor [15]. Such binding should be achieved under the circumstance that all actors within the system are semi-trusted. That is, each identified actor may lie according to their own interest. Therefore the bindings must be supported by provable or non-disputable evidence.

In our approach, accountability can be incorporated into activity based process by requiring the actor (conductor) of the process to log non-disputable evidence about the
activities in a separate domain from the domain of its own. Fig. 2 shows an example of such incorporation. In the example, domain A is required to perform logging operations before and after conducting the activity in its process. The evidence needs to be logged should contain enough information to describe the conducting activity. In the simple case in our example, intuitive enough, the evidence should include the states of the factors concerning the start of the activity (e.g., input variables) and the factors concerning its completion (e.g., output value).

As aforementioned, the logged evidence needs to be non-disputable so as to undeniably link the activity to its actor. That is, if by the logger or any other entity, the evidence can be proven to be associated with an activity conducted by a domain, no other entities can prove otherwise.

To achieve this, we assume the employment of PKI in all the domains in our example, so that each of them has its own associated public-private key pair issued by certificate authorities. The logging procedure is illustrated in Fig. 3. First, the logger (domain A) signs the evidence (E) by its private key (KA-). The evidence and its signature are then logged at a separate domain (domain B). When received, domain B creates a receipt by signing domain A’s signature with domain B’s private key (KB-). At last, the receipt (Sb) is sent back to the logger (domain A) in the reply. Through this procedure, the digital signature is un-forgeable, the signature of the evidence (Sa) and the receipt (Sb) enable both domains to prove the factor that domain A has logged this evidence at domain B. In another word, even though neither domain trusts the other, both of them can prove its own correctness with the signature it has kept.

As illustrated in previous section, two domains need to be involved for logging the evidence, and the logged evidence should be analyzed continuously so as to evaluate the logger’s correctness. In a cloud computing environment where services are dynamically introduced and dismissed, applying this concept can be tricky. Some literatures address this issue by deploying logging and analyzing mechanisms in all of the services involved in the composition. In such systems, service nodes have to volunteer to sign on each others’ evidence, and send auditing/challenge requests to each other from time to time in order to verify each other’s compliance. This can result in a chaotic situation where there is no global (or overall) common knowledge of the system’s state. This in turn results in inconsistent individual logs, and potentially a large amount of message exchanges caused by entities trying to discover this global state.

In our design, we propose to use a central accountability service to maintain the evidence logging and analysis for all the concerning service nodes involved in the business process. Fig. 4 shows the system model of our Trustworthy Service Oriented System (TSOA). We can see that services have been divided into two domains: the Accountability Service Domain (ASD) and the Business Service Domain (BSD). In the BSD, business services (BS) from multiple clouds interact with each other to conduct a business process, like the one-stop loan application service composition example we used. Each service in a BSD keeps a close association with the accountability services (AS) in ASD, so as to ensure that this process is indeed accountable.

We assume the AS obtains no benefit of assuring the correctness of the business services, and it plays a neutral role in the SOA system. Therefore, the ASD and BSD are two independent domains, since the correctness of either domain incurs neither benefit nor loss to the other. Further, misbehaviors of a service in either domain inevitably mean that it is willing to take the risk that it will be exposed in another domain. This concept of mutual constraint on services in the two domains is our approach in achieving trustworthiness. In systems with conventional service compositions, we can imagine that a single entity that provides this service and monitoring itself is more likely to misbehave and subsequently deceive. Since the service entity cannot be trusted, neither can the evidence it provides. Below we list the core functionalities of the AS node:

**Logging**: Non-disputable evidence associated with the activities conducted must be logged in real-time. Such history logs should be sufficient enough for any later disputes with respect to the predefined correctness of business processes. Note here that not all operations need to be logged – only identified critical ones (this will be elaborated on later in the paper).

**Monitoring and Auditing**: By analyzing the activity logs, the system’s state is continuously monitored to provide
underlying services or components monitoring information to assist their operations.

**Fault resolution:** Once an exception or violation is detected or reported, the root cause should be discovered in a provable manner and actions taken e.g. forbidding the violating service from further interaction until the dispute/violation is cleared.

In the following sections, we will illustrate the detailed operations of TSOA according to the core functionalities identified above.

**B. Evidence Logging**

Because the domains involved in the process need to log evidence for their activities, logging operations must be incorporated into the logger’s business process. Essentially, this incorporation should require minimal modifications to existing implementations and can be done through automation. In this paper, we propose an approach to allow this automation for the business processes which are orchestrated by process descriptive languages. Process descriptive languages define the business processes that involve activities associating with multiple external/internal services. This definition is usually in the form of scripts, which will be interpreted by orchestration engines [27] to conduct the process accordingly.

A good example of the process descriptive language will be Business Process Execution Language (BPEL) [18]. BPEL models the business activities into several basic activity types, and then compose those types to describe the whole process. The core activity types include: i) receive, receiving the request from a requestor, this activity type will specify the variable to assign the input data to; ii) invoke, invocation to an endpoint (service), invoke activity type will specify the variable used as the input and the variable used to store the output data for this invocation (if there is any); iii) reply, associating to a receive activity, a variable will be specified to be returned to the requestor as the result.

When executing processes defined in BPEL, it is these three activity types that describe the interactive activities with services involved in the process. Intuitively enough, in order to preserve evidence about business process, these are the activities that need to be logged.

To add logging activities into the process, we can insert invoke activity types into the BPEL script to invoke certain endpoint (logging service) with the evidence to be logged. And due to the distinct natures of receive, invoke and reply activity types, the rules used to decide the insertion locations are in fact quite straightforward. For the receive activity, an invoke should be inserted right after it, to log the variable carrying the input data the receive activity has just assigned. For the invoke activity, one invoke should be inserted before this activity and another to be inserted after. The two inserted invoke activities are used to log the input data and the output data of the invocation respectively. And finally for the reply activity, an invoke needs to be inserted right before it to log the result data which is about to be returned to the requestor. The invocation endpoint for the invoke activities inserted should either be a service in the same domain of the logger, or a trusted third party nominated by the logger, which in turn signs the evidence on the logger’s behalf and forward the signed evidence to the AS.

To further illustrate this transformation process, we have presented an example in Fig. 5. Fig. 5-a shows the graphical view of an ordinary sample BPEL created by Eclipse BPEL plugin. This simple process is started by receiving an input (ReceiveInput), then a partner link (collaborating service) is invoked in turn (InvokePartnerLink), and finally, reply the result to the client (ReplyClient). Fig. 5-b is the BPEL script after the transformation. We can see in Fig. 5-b four logging invoke activities (the InvokeLogging serie) have been inserted, one after the “ReceiveInput”, one before and one after “InvokePartnerLink” and one before “ReplyClient”. Because BPEL is entirely based on xml schema, any xml schema parser will be capable of analysing and inserting activities into it. The details of our implementation of such BPEL transformer will be shown in the evaluation section.

![Figure 5. Transformation of BPEL](image)

After the BPEL transformation, the logger needs to register with AS to start the logging. Certain documents should be submitted in order to give the AS sufficient knowledge about the logger. These documents can include: i) BPEL scripts, describing the business process so that the AS can verify if the activities the logger is conducting is legitimate; ii) Service Level Agreement (SLA) definitions, stating the logger’s guarantees on the quality of service (Qos) and the method to evaluate the compliance to them with the
evidence logged (e.g. WSLA [19]); iii) identification proof, certificates issued by authorities with the logger’s private key to prove its identity (e.g. X.509). iv) the WSDL of a trusted archive service nominated by the logger to receive the log receipts issued by the AS.

C. System Monitoring and Auditing

Monitoring and auditing (M&A) are used to discover misbehaving or faulty services. Services are bound into a system according to a pre-established Service Level Agreement (SLA) and business process logic (BPEL). It is these two that are used to ensure compliance in TSOA. The M&A determines if the operations performed or proposed by services satisfy the SLA, and are legitimate with respect to the business logic. Moreover this verification must be non-disputable, which means if the correct operation of an entity has been proven by some entity, no other entity can prove otherwise. Violations are regarded as exceptions, and steps to will be immediately taken so as to minimize their impact. A history of violations is kept by the M&A for later investigation of fault.

In our approach, M&A is done through the analysis of the incoming loggings to discover violations. The analyzing mechanism is encapsulated into the Monitoring & Auditing Logic, as shown in Fig. 6. It validates the legitimacy of service operations reflected in logs. The logic continuously analyzes the logged evidence to find if they can prove the logger’s correctness. Some types of violation or errors can be uncovered only after long time. For example, when the customer finds out that the loan offer is not the cheapest and reports to the AS. In this case the related evidence in the Archive will be audited to track the source of the fault.

![Diagram](image.png)

**Figure 6. Monitoring and Auditing mechanism**

M&A logic is generated from the service’s Business Logic (BPEL) and SLA registered.

As previously illustrated, BPEL defines the correct interactions between the services. In particular, it will define which service should be invoked for what purpose. This information can be used when monitoring to evaluate the correctness of the inter-activities conducted by collaborating services. For instance, in the one-stop loan application service composition example we used previously, if the Loan Bidding Company chooses not to involve Star Loan Company in the bidding, the expected loggings (according to the BPEL) from the Star Loan Company will never be received by the AS since that service is not invoked during the bidding, thus the business process error will be noticed.

SLA definitions need to contain the admitted obligations from service provider, the methods to evaluate the compliance to those obligations, and the compensation rules for violations. SLA may be defined in several standard formats. In this paper, we use WS-agreement [19] (WSLA) to illustrate our approach. A sample WSLA is shown in Table 1. The sample WSLA has two parts. The first part defines a ServiceLevelObjective which is a guarantee offered by the CreditCheckCompany. The guarantee states that the response time must be less than 1.6 seconds. The second part elaborates the method to validate the compliance to the previous guarantee, by using the value of the ReceiveTime (the time when CreditCheckCompany receives the request) to minus the value of ReplyTime (the time when the result is ready to be replied to the requestor).

<table>
<thead>
<tr>
<th>TABLE 1. SAMPLE WS-AGREEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;ServiceLevelObjective name=&quot;SLO_for_ResponseTime&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>&lt;Metric name=&quot;ResponseTime&quot; unit=&quot;milliseconds&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
D. Fault Resolution

Once an exception is noticed, or reported, the root (causer) of the exception must be determined according to the knowledge the AS has about the composition. Once the exception is successfully linked to a guilty service, certain actions need to be taken by the AS in response to stop the misbehavior or minimize its impact.

Violations to the WSLA should be handled according to the compensation rules defined in the WSLA. The most common form of compensation is through penalty, in this case, a penalty report will be constructed and sent to the compensator. The penalty report should contain the details about the violations and the evidence to support these accusations.

In case of violations to the business process, depending on the severity of the violation, different procedures can be followed. In general, the AS may first send warnings to the violator and temporarily tolerate it, until a violation limit is reached, the AS may send notices to all other service nodes to dismiss the violating service. Again, the notice should contain non-disputable evidence to convince them about the violation.

V. Evaluation on Amazon EC2

The Amazon Elastic Compute Cloud [22] is a computing resource provisioning service that charges the user according to the CPU usage. While leveraging such service, apart from the possible fault from the users who deploy incorrectly operating services on EC2, deployed services are inevitably subjected to quality of service (Qos) issues regarding the computing resource provided. Requirements on the availability, reliability, CPU speed, and network bandwidth are often as important as correct operations. In this section, we will illustrate through this scenario how TSOA makes SOA on EC2 trustworthy. Then we discuss the capability of TSOA to manage the trustworthiness of computing Cloud like Amazon EC2.

A. Implementation details

Our implementation testbed used Apache Tomcat 5.5 [20] as our Servlet container, and Axis2 1.5 [21] as our web service engine. We chose BPEL for defining the business logic and WS-Agreement (WSLA) for SLA in our AS nodes. Apache Orchestration Director Engine (ODE) [27] has been used to conduct the business process defined in BPEL.

The BPEL parser and transformer are implemented in JAVA by using W3C document object model (DOM) [28]. It turns out that it is quite handy to incorporate the logging activities into the business process, because the insertion rules we defined previously are straightforward to apply. Transformed BPEL scripts can be redeployed by simply dropping into the ODE process folder, ODE will realize the modifications and use the new process to retire the out-dated one (so can many other BPEL engines). Overall, we find incorporating the concept of accountability into a running business process is convenient and can be done with little impact or modifications on the business process implementations.

In order to allow the AS node to learn the context in BPEL and WSLA, the parser was also used in AS to extract the needed information from those definitions. The Archive is implemented as a PostgreSQL database [23] which is accessed through JDBC interface.

B. Experimental System

The AS and the five underlying business service nodes have been deployed on six standard computing stances in Amazon EC2. These are virtual machines with computing power equivalent to 1GHz CPU and 1.7GB memory. We then used this system to simulate the scenario that AS agent is deployed in the cloud computing environment to observe the correctness of the operations of services running in the environment.

To simulate our one-stop loan application service example, the composition has been defined in the BPEL scripts with logging activities incorporated and deployed in each of the business service node. The message in the communications are simply random data, the receiver of a message will encrypt the data with the node’s private key, and produce the result cipher text as the return data or the input data used to invoke other services. These messages here represent the loan application request, credit score or loan offers, and the encryption operation creates some processing latency.

The five nodes thus form an ordinary business workflow with embedded logging operations to log evidence for all receive, invoke, and reply activities. The WSLA definitions used for each of the services are similar to the example in table 1, except the value of response time guarantees are different for specific business services. The certifying services as well as the receipt archiving services are deployed in the same computing instances of their associated business services.

A client (at the University of Sydney) sends request to the first service node – one-stop loan application company, and the process goes on until the loan offer is returned back to the client. Note this experiment demonstrates the nature of service composition, in that the correctness of individual nodes directly affects the overall correctness. Any error introduced by any single service node may make final outcome incorrect.

Now with the evidence logged with the AS, any source of errors can be efficiently discovered. In this experiment, the operation involves a deterministic encryption process. Given an input, the output must be constant. In real composition, operations will be more complex than our example, however, as long as it is deterministic, verification is possible with proper evidence. This is the main benefit of our non-disputable logging. Below we list several
monitoring aspects we implemented in our experimental system:

**BPEL compliance**: according to the business process in Fig. 1, the correctness of the operation, as well as the correctness of service invocation is monitored regarding its BPEL.

**WS-Agreement compliance**: using the metric in Table 1, the response time of each of the business service is measured and compared to its guarantee in the WSLA.

**Integrity checking**: The evidence logged in AS contains the incoming/outgoing messages. A simple comparison is run to check if the message sent by the invoker is same as the message received by the invoking service.

**Transmission speed**: similar to the metric in Table 1, another metric can be defined to use the receive time at the invoked service to minus the invoke time at the invoker so as to find the time consumed in the message transmission.

Each of these represents service quality concerns from the clients’ point of view. Fig. 7 shows the monitoring results of the response time of three business service nodes. If we apply the WSLA in Table 1, we can see although the majority is below the 1.6 seconds guarantee, each of the business service still has violations from time to time. Because the timestamps contained in the evidence logged are signed by respective service nodes, they serve as undeniable evidence for the client to claim compensation from the business services.

C. **Cost analysis**

We found the cost of validating a value in the logged evidence against a value defined in WSLA or BPEL is negligible, as are the time for constructing the evidence and receipt. The main cost of TSOA comes from the communications between the AS and BS nodes, which is the extra transmissions introduced by incorporating accountability into the business process.

Fig. 8 shows the overall latency to finish the process (workflow) with untransformed BPEL scripts and with transformed ones. We have tested the workflow with request message size from 0.1KB (equivalent to a sentence) to 50KB (equivalent to a medium size document). The extra latency introduced after transforming the BPEL scripts grows as the request message becomes larger. In percentage, averagely we observed a 30% increase in the overall process latency. Intuitively, this latency is significant to the business process, however it can be improved through the use of hash functions. In the real practice, it is very rare to log the entire communication message as the evidence, instead, collision-resistant hash functions (e.g. SHA-1 [17]) can be used to compute the hash of the message which is a very small digest (160 bits for SHA-1) to be logged. Because the hashes computed are collision-resistant, which means it is theoretically impossible to have two different items with the same hash, so the hash can be logged to represent the evidence and be used to request the logger to submit the evidence when needed.

Considering the testing case with message size of 50KB, if we log the hash of the message computed using SHA-1 instead which is 160 bit, the extra latency will be similar to the latency for the testing case with message size of 0.1KB, resulting a latency increase of only 6.3%. Therefore, it is reasonable to conclude according to the testing that our approach introduces little cost in time for business processes involving medium to large volume of data.

**VI. RELATED WORK**

There are a number of approaches to addressing correctness and accountability of service oriented systems. One notable approach is monitoring. In [1][2][3], an embedded monitoring logic is incorporated into the deployed service’s BPEL. The logic is implemented in the form of try-catch blocks that check specific attributes according to an SLA. In [4] a standard language is proposed to describe the monitoring logic for the SLA. This language is used to generate checking scripts to be inserted as WS handlers. A key difference of these approaches to ours is that they assume the service entities will strictly monitor their own behaviors and report exceptions when found. However, our approach does not make this assumption, as all service entities expected to behave in any “good” or known manner.
Attempts like those presented in [5][6][7] require the services to archive the events (Trace in this paper). When a problem occurs, those logs are analyzed to find the responsible entity. Here a trust assumption is made that the services will admit to the events they have archived, even when such admission would lead to penalties due SLA violations. Moreover the archiving mechanism itself is a questionable process. These concerns are however addressed in our TSOA through digital signature and standard definition of archiving evidence.

There are some approaches have closer correlation to our idea. [8] is an attempt to achieve secure accounting of utility storage, and it requires the storage service provider as well as the client to sign for every request so that the usage can not be denied by either party. [9] uses a similar approach to [8], however it uses such a method to achieve certified accountable tamper-evident storage service. Instead of undeniable usage, it uses the signed actions logged to verify the correct state of stored data. Any changes cast by both client and the service is provable and undeniable. [10][11] require service nodes to maintain temper-evident logs about their receiving and sending requests as well as service state digests. Correctness is verified through service nodes volunteer to send challenge and audit requests to each other from time to time to collect those logs for validation. These attempts to certain degrees share the point with us that system correctness is verified through accountability where provability becomes one of the major concerns.

Our work is originated from [25][26], in which accountability is provided by services as a monitor of the system’s correctness. This idea has been well developed and refined in our work. We incorporate the AS into the business process through the transformation of BPEL scripts, and standard SLA definition has been used for Qos measuring.

VII. CONCLUSIONS AND FUTURE WORKS

In this paper, we proposed to use accountability to achieve trustworthiness in cloud computing environment. Our design separates service space in the cloud into two domains, ASD and BSD. In BSD, business services incorporate accountability into the business process to log non-disputable evidence in ASD. The AS in the ASD analyzes the evidence logged and validates them against the business logic and SLA registered by the business services. Such design enables the system to conclude service faults and violations in both domains with non-disputable and provable evidence. Also any service is able to prove its own correctness against false accusations. We have proposed a method to incorporate this accountability into business process orchestrated by business descriptive languages. The incorporation incurs minimal impact on the implementations of the business process and can be done in automation. We implemented TSOA into a cloud computing case study which shows its capability to enforce strong accountability in various aspects. Our testing figures indicate the cost of incorporating our design to SOA in the cloud is acceptable.

In the future work, we plan to improve our design by utilizing distributed storage and parallel computing techniques in the cloud. Imagine a business process involving hundreds of service nodes (computing instances) in the cloud, AS nodes need to be deployed in multiple physical locations so as to minimize the bandwidth consumed by logging operations. For those deployed AS nodes, the amount of logs from business services may be overwhelming and need to be distributed to several nearby computer nodes, in this case, a distributed file system like Google File System (GFS) [29] or Hadoop [30] can be used. Then, in order to regularly audit all the logged evidence stored in such distributed manner to find any potential faults, multiple physical storage entities may be queried to conduct analysis on their logs and forward the result to a target judgment/decision entity, this will be an ideal scenario to apply map-reduce [31] techniques to reduce the overall time cost.

REFERENCES

[14] Trustworthy system definition, at IETF RFC 4949

Amazon EC2 SLA: http://aws.amazon.com/ec2-sla


Apache Tomcat: http://tomcat.apache.org/

Apache Axis2: http://ws.apache.org/axis2/

Amazon EC2: http://aws.amazon.com/ec2

PostgreSQL database: http://www.postgresql.org/


Apache ODE: http://ode.apache.org/

W3C Document Object Model: http://www.w3.org/DOM/


Apache Hadoop: http://www.hadoop.apache.org/


Demonstrative material:

Power Point Slides: Towards Trustworthiness
URL: https://accountability-service-for-the-cloud.s3.amazonaws.com/Towards_Trustworthiness.pdf

Video 1: BPEL Transformation Example
URL: http://www.youtube.com/watch?v=69pFVJk5oB4

Video 2: Monitoring by Accountability Service
URL: http://www.youtube.com/watch?v=Js-5gDGJU5s

Source code of the core part of the project:
URL: https://accountability-service-for-the-cloud.s3.amazonaws.com/AS_Project_Code.zip

Performance testing data regarding the latency of original business process and the transformed one:
URL: https://accountability-service-for-the-cloud.s3.amazonaws.com/Process_Overall_Latency.xls