Strong Accountability for Service Compliance in the Cloud

Jinhui Yao
School of Electrical Engineering, University of Sydney
Shiping Chen
CSIRO ICT Centre, Australia
Chen Wang
CSIRO ICT Centre, Australia
David Levy
School of Electrical Engineering, University of Sydney
and
John Zic
CSIRO ICT Centre, Australia

In recent years, computing resource provisioning through the adoption of the cloud computing has emerged as a promising paradigm to let companies and enterprises outsource their computational needs. Along with the widely adopted Service Oriented Architecture (SOA), organisations can wrap various kinds of technological product they are offering as a service, to collaborate with services provided by others to form new value-added business products. Facing the ever-escalating global competition in current economy, such collaboration is crucial for their survival. However, it is challenging to achieve trustworthiness in such a dynamic cross-domain environment, as each participant may deceive for individual benefits. As a solution, we propose a novel design to enforce strong accountability to enhance the trustworthiness in the cloud environment. With this accountability, the root of a violation can always be identified and associated with the responsible (or guilty) entity or entities, and this association is supported by non-disputable evidence. We elaborate the approach to incorporate our design into existing business processes defined using standard descriptive languages for business logic and service level agreements. Then we deploy the system into a computing cloud to evaluate its effectiveness.

Keywords: cloud computing, accountability, service oriented architecture, trustworthiness, compliance assurance

1. INTRODUCTION

In recent years, we have witnessed a range of innovations in the 'service' related technologies and concepts. Following the Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS) and many more "as a Service" concepts have been proposed. As one of the outcomes, computing resource provisioning through the adoption of the cloud computing has emerged as a promising paradigm to let companies and enterprises outsource their computational needs. Along with the widely adopted Service Oriented Architecture (SOA), organisations can wrap various kinds of technological product they are offering as a service, to collaborate with services provided by others to form new value-added business products. Facing the ever-escalating global competition in current economy, such collaboration is crucial for their survival [Papazoglou et al. 2007].

The correctness of the inter-organization business processes relies on the individual correctness of all participants, that is, if the collaborator is compliant to the pre-defined business process, or Service Level Agreement (SLA). Any incompliance of the participants can to various extents, negatively affect the value of the business product associated with this business process. It follows

Contact Author's address: J. Yao, School of Electrical and Information Engineering, the University of Sydney, City Rd, Camperdown, NSW 2006, Australia.
Contact Author's email: jin.yao@sydney.edu.au

that, the viability of this paradigm and the willingness of new participants to join collaboration highly depend on the trustworthiness of the behaviors of all collaborators.

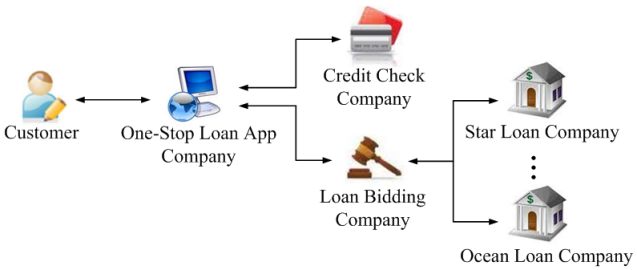
Here we adopt the informal definition of trust based on comments made by Graeme Proudler: something can be trusted when (i) it can be unambiguously identified; (ii) it operates unhindered; and (iii) the user has either first hand experience of consistent good behaviour or knows someone who can vouch for consistent good behaviour. A formal definition of trustworthy systems is presented in IETF RFC4949 Internet Security [Mahbub and Spanoudakis 2004], that a trustworthy system is a system that is already trusted, and continues to warrant that trust because the system's behaviors can be validated in some convincing way.

It is a challenging task to preserve trustworthiness in such a dynamic cross-domain environment. The composed business process usually spans several administrative domains, each of which will have its own interests and priorities. Given that admission to violations may lead to penalties in some form, it is conceivable that they may intend to deceive and hide this fact. Therefore, a mechanism to detect and prove incompliance is urgently needed for this collaboration paradigm to prosper.

As a solution, we propose a novel design to enforce strong accountability to enhance the trustworthiness in the cloud environment. While this will be illustrated shortly in following sections, briefly, accountability provides means to verify compliance according to evidence in a provable and undeniable way. In a system with strong accountability, the root of a violation can always be identified and associated with the responsible (or guilty) entity or entities, and this association is supported by non-disputable evidence. We elaborate the approach to incorporate our design into existing business processes defined using standard descriptive languages for business logic and service level agreements. Finally we deploy the system into a computing cloud to evaluate its effectiveness.

The remaining parts of the paper are structured as follows: In the next section, we describe a motivating scenario which will be used as a running example in this paper. In section 3 we illustrate the core concepts of accountability and show how it can enforce compliance. Section 4 elaborates our design to incorporate accountability into the service collaborations in the cloud. In section 5 a prototype is presented to evaluate its effectiveness and performance. Section 6 compares our approach to related work. Finally, we conclude in section 7 with a summary and a discussion of our future research directions.

2. A MOTIVATING SCENARIO



Online one-stop loan application service composition

We use a one-stop loan application service as the running example in this paper. As shown in Figure 1, the process requires the collaboration of five entities. First, a one-stop loan application service allows the customer to lodge loan application and fill in his personal information. His personal information will first be used to obtain a credit score from the credit rating authority and then the score is attached with other personal information to be sent to the loan bidding company. The bidding company forwards the application to multiple loan companies (Star Loan

& Ocean Loan), and selects the cheapest offer (if any) available to return to the applicant. In this typical collaboration scenario, the overall correctness of the system depends on the correctness of all individual participants. As every of them may be interested to violate the collaboration rules for their own benefit or/and deceive to avoid possible penalties, the causer of a failure may be extremely difficult to determine.

For instance, a loan applicant, Bob, finds out that he could have been offered a cheaper loan through direct contact with one of the loan companies claiming to be involved by the bidding company, proving that the one-stop loan application service has failed in its promised service outcome. Bob cannot determine whether it is the credit rating authority that gave him a bad rating, or whether the loan company is not actually involved during the bidding process. 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 on to the bidding company. In turn, the bidding company could also do the same, and shift the blame either to the credit rating authority or to the loan companies. Even in this simple example it can be seen that a mechanism is required to prevent this "buck passing" or denial of failure, this mechanism is essential for controlling the correctness of a business process established.

3. ACCOUNTABILITY FOR COMPLIANCE

Accountability can be interpreted as the ability to have an entity account for its behaviors to some authorities [Mulgan 2000]. This is achieved by binding each activity conducted to the identity of its actor with proper evidence [Yumerefendi and Chase 2004]. 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, accountability should entail a certain level of stringency in order to maintain a system's trustworthiness. Below, we identify several desirable properties of a fully accountable system:

— Verifiable: The correctness of the conducted process can be verified according to the actions and their bindings recorded.

— Non-repudiable: Actions are bound to the actors through evidence, and this binding is provable, and therefore undeniable.

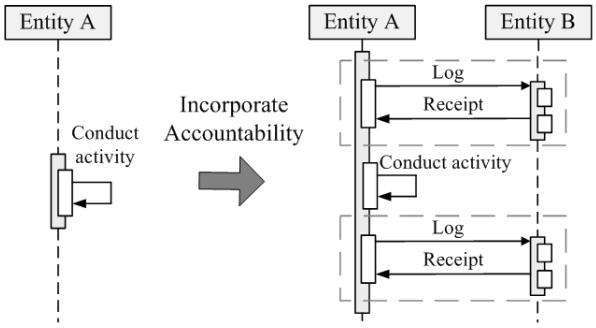
— Tamper-evident: Any attempt to corrupt to recorded evidence inevitably involves the high risk of being detected.

Given above properties, a fully accountable system is in fact a fully provable system. The current state of the participants can be verified by what has been done in the past; and none of them can negate their actions; neither can they change the evidence recorded during the actions.

With these notions, apparently, the handling of evidence is critical for accountability. To verify the correctness of the processes conducted, evidence associated with the conducted activities must be preserved, examples include input, output data, and service internal states. And to make such evidence non-repudiable, cryptographic techniques should be employed to let the conductor of the activities digitally sign on the evidence to make them undeniable.

The third property, tamper-evident, is the most important one as we regard the activity conductors as untrusted. Untrusted entities are likely to alter, corrupt or delete entirely the recorded evidence given that the evidence may lead to severe penalties in some form. Therefore, this fact entails the need to maintain the evidence in a separate entity (entity B) other than the one which generates the evidence (entity A). Ideally, entity B should 1.) have no benefit related to the compliance of entity A; and 2.) be willing to honestly maintain the evidence stored. Assuming entity B is randomly chosen and does not even know entity A, it is still difficult to guarantee the second term, that is, whether it will faithfully deal with the evidence stored, as it can be another untrusted entities. It follows that, entity B also needs to digitally sign on the stored evidence and send this undeniable certificate back to entity B to vouch for the fact that entity A has stored certain evidence in entity B.

We illustrate our proposed approach in Figure 2. In our approach, accountability can be incorporated into activity-based process by requiring the entity conducting the process to log non-disputable evidence about the activities in a separate entity. In the figure, after incorporating accountability into an ordinary process, entity A is now required to perform logging operations before and after conducting the activity in its process. The evidence is logged in a separate entity – entity B – so that entity A cannot access the logged evidence. The evidence needed to be logged should contain enough information to describe the conducting activity. In our simple example, which is intuitive enough, the evidence should include the states of the factors concerning the start of the activity (e.g. the input variables) and the factors concerning its completion (e.g. the output value).



Example of incorporating accountability into process

As aforementioned, the logging operations require the employment of Public Key Infrastructure (PKI) in all involved service entities. Each of them has its own associated public-private key pair issued by certificated authorities. The logging operations are as follows:

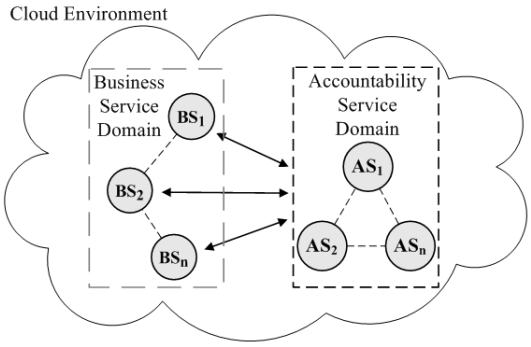
(1) The logger (entity A) signs the evidence (E) by its private key (K_{A-}) to create a digital signature of the evidence (S_A).
(2) The evidence and its signature are then logged in a separate entity (entity B).
(3) When received, entity B creates a receipt by signing entity A's signature with entity B's private key (K_{B-}).
(4) Lastly, the receipt (S_B) is sent back to the logger (domain A) in the reply.

Assuming the digital signature is un-forgeable, the signed evidence in entity B can be used to verify entity A's compliance; and yet any corruption or deletion applied to the evidence will be discovered using the receipt received by entity A. Under the circumstance that neither of the service entities is trusted; and assume they will not conspire to cheat (this assumption will be relaxed later), this structure manages to ensure the proper preservation of evidence associated with the process conducted. It is the basic structure our proposed system is built on.

4. SYSTEM DESIGN

As illustrated in the previous section, two entities need to be involved for preserving the evidence. Further, to verify the compliance of the logger, logged evidence need to analyzed continuously. For a system composed by services, one way to facilitate that, is to select some of the participating services to store and process the evidence submitted by some of the others. However, this approach may not be suitable for our scenario. In a cloud computing environment where services are composed to form workflows, a global view over the composed business process is critical for analyzing the participants' compliance (evidence from multiple service nodes may be needed to verify some individual nodes' correctness). This would require all the selected nodes to observe the activities of all other nodes, resulting in an enormous message exchange overhead.

In our design, we propose to have special service nodes, dedicated to provide accountability to all underlying services involved in the business process. Those special nodes are referred to as the accountability service (AS) nodes. Figure 3 shows the overall model of our system. Let us imagine the cloud to be a space of services, in our designed system, this space has been divided into two domains: the business service domain (BSD) and the accountability service domain (ASD). In the BSD business services (BS) compose with each other to conduct complicated business processes, like the loan application example we used.



Overall system design

Accountability services (AS) in the accountability service domain (ASD) continuously process the evidence received so as to ensure that the BS nodes are held accountable. Each BS node may be associated with several AS nodes, which means it needs to submit evidence to everyone of them. Similarly, each AS node may be in charge of a number BS nodes, it analyzes the evidence submitted by all the BS nodes it monitors.

The use of multiple AS nodes to enforce accountability is mainly due to three concerns. Firstly, according to the scale of the business process, the number of BS nodes involved may range from several to hundreds. To make such systems scalable, a number of AS nodes should be employed to monitor the compliance of different parts of the process. The second concern is to prevent the case that an AS node conspires with certain BS nodes to cheat for hiding compliance violations. When a BS node submits evidence to multiple AS nodes, the determination of its compliance will be the voting result of all of the AS nodes involved, which reduces the impact of individual faulty or deceptive AS nodes. And the third concern is to deal with the possible disputes raised while evaluating BS's compliance. Details of voting and dispute resolution will be elaborated in later sections.

The AS here can either be provided by the cloud, or by other third parties as long as they receive no benefit whether the BS is being compliant or incompliant. It plays a neutral role in the cloud. Therefore, this topology satisfies the concept of accountability previously discussed. The misbehaviors of a service in either domain inevitably mean that service is willing to take the risk that it will be exposed in another domain. This mutual constraint on services in the two domains is the main strength of our approach to achieving trustworthiness. With this topology, the core functionalities of the AS node are as follows:

— *Evidence logging*: Non-disputable evidence associated with the activities conducted must be logged in real time. Such logs should be sufficient enough for any later disputes with respect to the predefined correct behaviors.

— *Compliance validation*: Through the analysis of the activity logs, the system's state is continuously monitored. Once a violation is detected or reported, the root cause should be discovered in a provable manner and actions will be taken to remedy its impact.

— *Dispute resolution*: In special cases where the source of failure cannot be bound to a specific entity, procedures will be carried out to determine the violating entity(s) in a best effort manner. Disputation should never cause minimal delay to the system.

As depicted in Figure 2, accountability needs to be incorporated into the existing business process. Facilitating above functionalities inevitably involves configuring both the nodes in BSD and ASD. To describe the required configurations, we proposed an XML based definition, called the "Accountability Policy" (AP). While it will be thoroughly elaborated in the following sub-sections, in general, AP tells the AS what evidence will be provided by the BS and how they can be used for compliance verification and dispute resolution. After the BS provider has implemented certain mechanisms for his BS to interact with the AS, an AP will be produced and submitted by the BS provider to the AS for registration. AS will thus create the monitoring logic to get ready for the incoming evidence from that BS. In the following three sub-sections, the accountability functionalities will be discussed, together with their respective specifications in the accountability policy.

4.1 Evidence logging

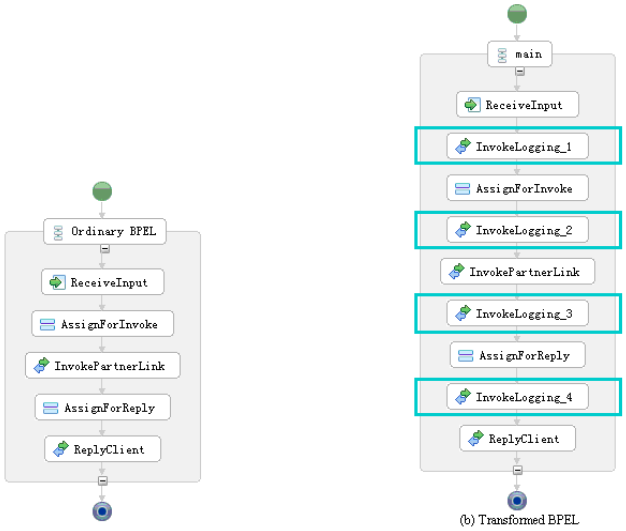
As in a legal setting, evidence plays the most critical role for the determination of one's guilt or innocence. The effectiveness of the accountability mechanisms crucially relies on the collected evidence during the execution of the business process. The logged evidence must be sufficient to prove one's compliance or incompliance when problems occur, and yet not too comprehensive. Therefore, determining the data to collect is a fundamental process for achieving accountability.

A service node conducts its designed activities while communicating with other service nodes via messages. So a service node – S can be defined as $S = (A, I)$, where A is the activities and I is the interactions with other nodes in terms of sent and received messages. A business process is a collection of service nodes $BP = \{S_1, S_2, \ldots, S_i\}$. To preserve evidence for BP, one needs to save both A and I for all the services involved. While interactions can be captured by recording the incoming and outgoing messages, the local activity conducted can only be recorded by logging the internal state of the service node. So we define evidence – $E = (P, M_{in}, M_{out})$ where P is the snapshot of local states,and M_{in} and M_{out} are the received and sent messages. With this setting, during the execution of business service nodes, the messages will be recorded, and periodically the node takes snapshots of its internal states. Note that the state snapshot and some messages may be extremely large and thus difficult to be transmitted to AS. In this case, the digest (hash value) of the evidence can be logged as a replacement.

Incorporating the accountability mechanisms into the business process involves the inclusion of these to capture the evidence and log it to the AS. State snapshots can be implemented in various ways, depending on the architecture of the system. For example, databases often provide specific commands to generate snapshots. In other scenarios, non-trivial ad-hoc solutions may be applied. On the other hand, service interactions play a bigger role for monitoring compliance during the run time, as they show the inputs and outputs of a BS node. Here we propose an approach that allows the automation of interaction capturing and logging for the business processes that 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 (e.g. Apache ODE) to conduct the process accordingly. A good example of the process descriptive language is Business Process Execution Language (BPEL) [Andrews et al. 2003]. BPEL models the business activities into several basic activity types, and then composes those types to describe the whole process. The core activity types include:

(1) *Receive*, receiving the request from a requestor. This activity type will specify the variable to which the input data is to be assigned.



(a) Original BPEL (b) Transformed BPEL

Transformation of BPEL

(2) *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.

(3) *Reply*, replying the invocation. A variable will be specified to be returned to the requestor as the result.

To add logging activities into the process, we can insert *invoke* activity types into the BPEL script to invoke a 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 input data received. For the *invoke* activity, one *invoke* should be inserted before this activity and another to be inserted after, to log the input data and the reply data of the invocation respectively. And finally for the *reply* activity, an *invoke* needs to be inserted just before it to log the result data that 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 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 Figure 4. Figure 4a shows the graphical view of an ordinary sample BPEL. This simple process is started by receiving an input (ReceiveInput); then a partner link (collaborating service) is invoked in turn (InvokePartnerLink), and finally, replies the result to the client (ReplyClient). Figure 4b is the BPEL after the transformation. We can see in Figure 4b that 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 analyzing and inserting activities into it. The details of our implementation of such BPEL transformer will be shown in the evaluation section.

To inform the AS about the evidence which the BS is going to submit, the provider of the BS needs to declare the content and type of the evidence in the accountability policy (AP). Listing 1 is an example of the evidence definition. In the policy the service provider explicitly defines each evidence item the service node will log with AS, and the data structure of each. The first three evidence items are all input and output SOAP messages, so the URLs of their respective WSDL definitions are provided as reference. Note that the "endpoint" is provided for the sending

message "InvokeEvidence". AS will retrieve the WSDL definitions once this policy is registered with it to learn the semantic. The last item is the digest of the system's state snapshot. As its data structure is simply hash, only the log interval is given (which is daily).

Listing 1: Sample policy - Evidence section

```
<Policy issuer=BusinessServiceProvider>
  <Evidence>
    <Item name ="InputEvidence" type="SOAP" url="...wsdl"/>
    <Item name ="InvokeEvidence"  endpoint="ReceivorNode" type="SOAP" url="...wsdl"/>
    <Item name ="ResponseEvidence" type="SOAP" url="...wsdl"/>
    <Item name ="SystemState" type="SnapshotDigest" interval="daily"/>
  </Evidence>
  ...
</Policy>
```

4.2 Compliance validation

As we have discussed ealier, accountability aims to make all the entities answerable for their activities. This implies that an entity needs to be responsible for the consequences of its actions. When the actions violate specific requirements, the entity will be penalized. Pre-established obligations and agreements for business services in the composition can have many forms. Various standards have been proposed to define certain types of requirement. For example, Web Service Agreement (WS-Agreement), Web Service Level Agreement (WSLA), Web Services Choreography Description Language (WSCDL), etc. To illustrate our approach we assume that the compliance requirements are expressed in Web Service Level Agreement (WSLA) and the business process logic is expressed in BPEL. However, our approach can be extended to accommodate different types of description languages.

With this assumption, the validation logic in AS needs to verify that i) the performance of the BS nodes meets the assurances defined in WSLA; and ii) the activities conducted by the BS nodes are compliant with the business process logic defined in the BPEL. A notable feature of WSLA is that it not only defines the SLA assurances, but also specifies the procedure to verify such assurances. Listing 2 below contains some extracts from a sample WSLA. Firstly, the ServiceLevelObjective states that the SLAParameter (AverageResponseTime) needs to be less than 1.6 seconds. This SLAParameter is defined in the second part, which should be the result of the calculation defined by a Metric specified in the third part.

Listing 2: Sample web service level agreement (WSLA)

```
(Part 1)
<ServiceLevelObjective name="ResponseTimeObjective">
  <Predicate type="Less">
    <SLAParameter>AverageResponseTime</SLAParameter>
    <Value>1.6</Value>
  </Predicate>
</ServiceLevelObjective>
(Part 2)
<SLAParameter name="AverageResponseTime">
  <Metric>AverageResponseTimeLastHour</Metric>
  <Source>BusinessService</Source>
</SLAParameter>
(Part 3)
<Metric name="AverageResponseTimeLastHour">
  <Function type="Divide" resultType="double">
    <Operand>AccumulatedResponseTime</Operand>
    <Operand>Transactions </Operand>
  </Function>
</Metric>
```

The validation logic in AS therefore needs to be accustomed to verifying the compliance according to the WSLA shown in the example, and the BPEL scripts deployed by the BS nodes. In the Accountability Policy this is expressed in the "validation" section, where "Functions" are defined to verify compliance with different description documents. The locations of the documents will be provided as a URL; and if the verification methods are also available (such as WSLA), their location will be enclosed as well.

An example of the validation section of the policy is shown in Listing 3. Two functions are defined: one is to verify business logic compliance, while the other is to verify SLA compliance. In "BusinessLogic_function" the URL of the corresponding BPEL script is given. The two "validate" specify that AS needs to check from the evidence logged whether the operation of an interaction is defined in the BPEL script, and whether the endpoint of the transmitting message is one of the "partnerLink" of the sender. In "SLA_function", the URL of the WSLA is provided, together with the SLA objective against which to validate. While the "SLAParameter" and the verification "Metric" can be reused by AS, an extra "Metric" needs to be defined to tell AS how to obtain the "SLAParameter" from the evidence logged. WSLA was designed with the assumption that the required parameters can be provided honestly by the service provider. As this assumption is no longer appropriate in our scenario, "SLAparameters" must be computed with the undeniable evidence logged so as to make the validation conclusion non-disputable. Therefore service providers thus need to describe specifically the computation method in "Metric" in the policy.

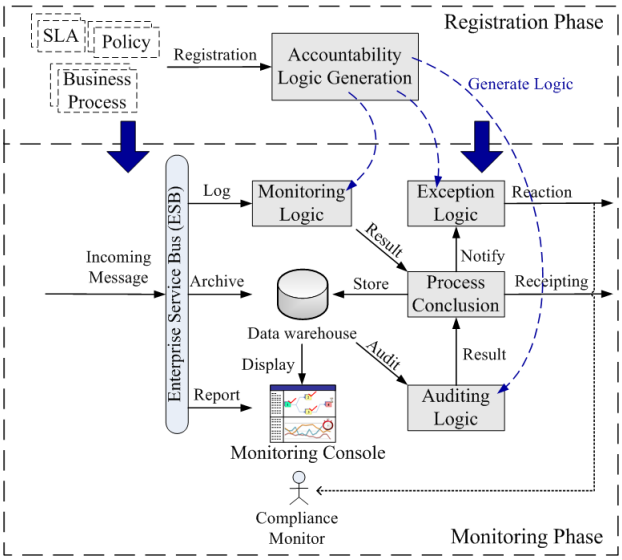
Listing 3: Sample policy - Validation section

```
<Policy issuer="BusinessServiceProvider">
  <Validations>
    <Function name="BusinessLogic_function">
      <document name="BusinessLogic" type="BPEL" url="http:.../BPEL">
      <validate name="OperationTypePermitted">
        ($BusinessLogic.operation)INCLUDE(operationType)
      </validate>
      <validate name="PartnerLinkCorrect">
        ($BusinessLogic.partnerLink)INCLUDE($endpoint)
      </validate>
    </Function>
    <Function name="SLA_function">
      <document name="SLA" type="WSLA" url=".../WSLA">
      <validate name="ResponseTimeObjective" >
        <SLAParameter="AverageResponseTime" >
          <Metric="ResponseTimeFromEvidence">
            <Operation type = "subtraction">
            <Operand EvidenceItem="InputEvidence" field="timestamp"/>
            <Operand EvidenceItem="ResponseEvidence" field="timestamp"/>
          </Metric>
        </SLAParameter>
        <Metric="AverageResponseTimeLastHour">
      </vallidate>
    </Function>
  </Validations>
</Policy>
```

As we have mentioned, BS nodes need to register with AS before submitting evidence. During registration, Business Logic, Policy and SLA (and other description documentation) will be submitted by the business services participating in the business process, to form the validation logic in AS. Note that it is up to the business service providers to agree to supply this information and log evidence at AS; they may choose only to allow AS to validate some of its compliance assurances. In this case, AS will only validate and vouch for the compliance that it can validate. We regard this flexibility as important for allowing services to control their desired level of accountability to suit different circumstances.

Figure 5 displays the inner architecture of the AS node. The registered "Accountability Policy" along with other description documentations (e.g BPEL, WSLA) will generate three components:

"Monitoring Logic", "Auditing Logic" and "Exception Logic". Monitoring logic continuously analyzes the logged evidence to find any obvious compliance violations; this is also called "online monitoring" due to the fact that this component needs to remain active while the business process is being executed and the evidences are processed in real time. This type of monitoring will focus mainly on the validation of less complicated compliance issues identifiable by analyzing a reasonably small amount of evidence entries. For example, to find out the response time of an execution, one simply needs to subtract the timestamp in the "ResponseEvidence" from the timestamp in the "InputEvidence" (as expressed in Listing 3). The analysis result will be stored in the data warehouse for later reference.



Validation components of the Accountability Service

On the other hand, the auditing logic periodically (or upon request) audits all the evidence items and previous analysis results stored in the data warehouse. This is in order to identify suspicious activity patterns, or to validate certain compliance that requires a systematic view across the overall behaviors. The purpose of this audit is to identify or solve those violations which can only be uncovered after a long time (or by reporting). For example, when the customer finds out later on that the loan offer is not the cheapest and reports this to the AS. In this case all the related evidence in the data warehouse will be audited to track the source of the fault.

Once the violation is successfully linked to a guilty service, certain actions need to be taken by the AS in response to stop the misbehavior or to minimize its impact. Violations of the WSLA should be handled according to the compensation rules defined within it. The most common form of compensation is through penalty. In this case, a penalty report will be constructed by the "Exception Logic" and sent to the compensator. The penalty report should contain the details about the violations and the evidence to support these. In case of violations of the business logic, depending on the severity, 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; or instead the AS may send notices to all other service nodes to dismiss the violating service. Again, the notice should contain convincing evidence to notify them about the violation.

The monitoring data and the data received from the BS nodes are archived in the data warehouse. These data are continuously processed and displayed in the monitoring console, an application that displays real time compliance status of the business process. The details of the monitoring console will be shown in the evaluation section.

4.3 Dispute resolution

One of AS's major tasks is to find the root of a failure. Although, even the failure space may be reduced by a variety of heuristics, in some cases a particular solution pointing to a specific entity may be unachievable. Under such circumstances a dispute is inevitable. Looking again at our earlier example, if the Star Loan Company does not respond to the Loan Bidding Company's bidding request, the bidding company cannot prove it has actually sent the request. Moreover, disputes may be caused by the conflicting conclusions made by different AS nodes, as the evidence stored by them may have different versions. When addressing disputes, simply pausing the service nodes concerned before it is solved may severely impact the operation of the whole business process. This is a challenge to an implementation of an effective accountability service able to resolve suspected bad behaviours in a consistent, non-disputable manner. To tackle this we propose to deploy a complementary best effort mechanism based on *voting* and *probing*.

As we mentioned earlier, one BS node may be monitored by several AS nodes, which will conclude its compliance according to their respective evidence received. Any penalty concluded by an AS node shall not be exercised until it is agreed by the majority. Ideally all AS nodes should have a consistent view upon the underlying BS nodes; however if exceptions happen (e.g. false accusation), the majority of the AS nodes can still prove its innocence. Such a *voting* method tolerates some scope for disputes and failures. Further, it makes the conclusion and accusation from ASD more reliable as it helps to address the concerns that an AS node could also misbehave, either mistakenly or deliberately.

Probing is used to actively test the correctness of the disputed services nodes. The concerned AS nodes send a special probe message to every disputed node to challenge the nodes to prove their correct behaviour. The majority of the probing results from AS nodes would give a conclusion about which node(s) is faulty with a high probability level, and would indicate whether it should be penalized or replaced. The probing operation needs to be declared in the Accountability Policy. An example of probing definition is shown in Listing 4. It describes a probing case for testing the loan bidding process conducted by the LoanBiddingCompany. In the policy, it defines the probing message and the endpoint (LoanBiddingCompany). Then it defines the expected outcomes if that service node is properly functioning. Which are the evidence expected to be logged by the bidding company and all the loan companies involved.

Listing 4: Sample policy - Probing section

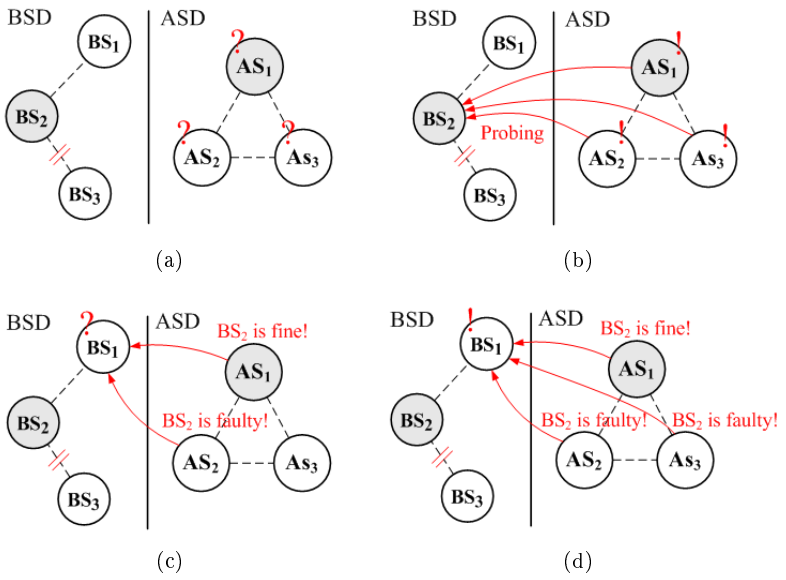
```
<Policy issuer="LoanBiddingCompany">
  <Probing>
    <ProbingCase name="ProbeLoanBiddingProcess">
      <Message name="ProbeBiddingMessage" type="SOAP" url="...wsdl"/>
      <Endpoint name="LoanBiddingCompany"  url="http:..."/>
      <Outcome>
        <Item name = "InputEvidence" Issuer="LoanBiddingCompany".../>
        <Item name = "InvokeEvidence"  endpoint="StarLoanCompany" Issuer="
            LoanBiddingCompany" .../>
        <Item name = "InvokeEvidence"  endpoint="OceanLoanCompany" Issuer="
            LoanBiddingCompany" .../>
      </Outcome>
    </ProbingCase>
  </Probing>
</Policy>
```

Possible violations incurred by any BS nodes as well as AS nodes can be regarded as Byzantine faults [Lamport 1983]. The faulty node under this category may exhibit arbitrary behaviors, such as being non-responsive or sending faulty messages. Early study of Byzantine faults [Haeberlen et al. 2006; 2007] has found that for the diagnostic system to find the incompliant node, a system requires $f + 1$ nodes, where f is the number of possible concurrent Byzantine faults. This is the case when absolute evidence is available to that last healthy node, which can unarguably prove the violations of the others. However, when conspiracy is involved, the conspiring AS node may

be able to forge fake yet seemingly genuine evidence, as a bid to prevent certain BS nodes from being penalized.

Certainly, it is not likely an AS node will be able to forge evidence to claim something the conspiring BS node has not done. For instance, to claim BS_1 has sent a message to BS_2 (which it has not), the AS node will need the private key of BS_2 in order to forge the log submitted by BS_2 after receiving the message. But in some other cases, when the private key of the conspiring BS node is all what it needs, an AS node will be able to generate seemingly undeniable evidence. For example, an AS node can always claim a BS node is not down, by showing the forged signed response from the BS node for its probing message.

Figure 6 shows an example of a probing process. In the example, three BS nodes are composed to form a business process, three AS nodes have been assigned to monitor them. In Figure 6a, a possible fault is noticed when BS_2 failed to send its output to BS_3 in the required time frame. All three AS nodes then send probing message to BS_2, and find out the node is overloaded by the requests from BS_1 (Figure 6b). In Figure 6c, AS_1 forges evidence to claim to BS_1 that BS_2 is working fine, so as to gain more job requests for BS2 (although they will be processed slowly) while AS2 tells the truth. At this moment BS_1 is not able to decide if it should forward more job requests to BS_2. Finally, in Figure 6d, AS_3 also notify BS_1 of BS_2 being faulty. BS_1 is thus convinced and send job requests to alternative BS nodes.



Probing and Voting process

Therefore, in the ideal case where certain evidence can unarguably prove compliance or incompliance, the requirement on the number of AS nodes (N_{AS}) assigned to one BS node is that $N_{AS} \geq f_{AS} + 1$, where f_{AS} is the possible number of AS nodes that are either unhealthy or deceiving. And in the worst case, where conspiring AS nodes are able to forge seemingly genuine evidence, more than half of the AS nodes need to be healthy and functioning justly so that the compliance of the monitored BS nodes can be correctly determined. The requirement on N_{AS} is that $N_{AS} \geq 2f_{AS} + 1$.

In the practice, it may not be easy to determine N_{AS} or the approach to choose the AS nodes with the smallest chance of conspiracy. Domain experts may be needed to make such decisions. An alternative approach may be looking at the historical performance of the system and apply

adjustments which eventually tune the system to its proper settings. In particular, one may be interested to study the cases when AS nodes make conflicting conclusions. In those cases, the ratio between the number of correct conclusions to the number of incorrect ones, indicates the overall correctness of the system and the degree of dominance of the healthy AS nodes. However, a further discussion on the optimisation of AS nodes selection is not in the scope of this paper.

5. EVALUATION ON AMAZON EC2

The Amazon Elastic Compute Cloud [1] is a computing resource provisioning service that charges the user according to the CPU usage. Users can deploy their services and business processes in the computing instances they rent in EC2. Computing instances can communicate with each other with speed close to a LAN. While making use of this computing environment, users can be quite concerned about other collaborators' compliance as well as their own. In this section, we will elaborate our demonstration system implemented in EC2.

5.1 The demonstration system

We deployed five BS nodes and one AS node on six standard computing instances in Amazon EC2. These are virtual machines with computing power equivalent to 1GHz CPU and 1.7GB memory. The five services in our loan application scenario have been implemented in the five BS nodes. Apache Tomcat 5.5 was used as the Servlet container, and Axis2 1.5 as web service engine in each of the BS nodes. The service nodes are orchestrated using BPEL scripts. Apache Orchestration Director Engine (ODE) has been used to conduct the business process. To incorporate the logging mechanisms into the ordinary BPEL scripts, we implemented a simple BPEL parser/transformer in JAVA using W3C document object model (DOM). 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 them into the ODE process folder, ODE will realize the modifications and use the new processes to retire the outdated ones. Overall, we found that the incorporation of accountability into a running business process is convenient and can be done with little impact or modification on the existing implementation.

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 Listing 2, except that the value of response time guarantees are different for specific business services. And the Accountability Policy we defined has been described in different sections previously. In order to allow the AS node to learn the context in Accountability Policy, BPEL and WSLA, the BPEL parser/transformer was also used in AS to extract the needed information from those definitions. A data warehouse is implemented as a PostgreSQL database in the AS node, which is accessed through the JDBC interface.

An execution of the system is as follows, a client (at the University of Sydney) sends a 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. During which, the input and output of each service are submitted as the evidence to the AS. The AS processes the evidence and updates the compliance status of each BS nodes. The status as well as the evidence are then stored in the data warehouse.

5.2 The monitoring console

In order to visualize the capability of the AS in monitoring the status of the underlying business process, we have implemented a monitoring console in the AS node to show the information it has collected and concluded. A screenshot of the monitoring console is show in Figure 7. The console consists of four panels:

Document panel: displays the documentation registered by the services participating in the busi-

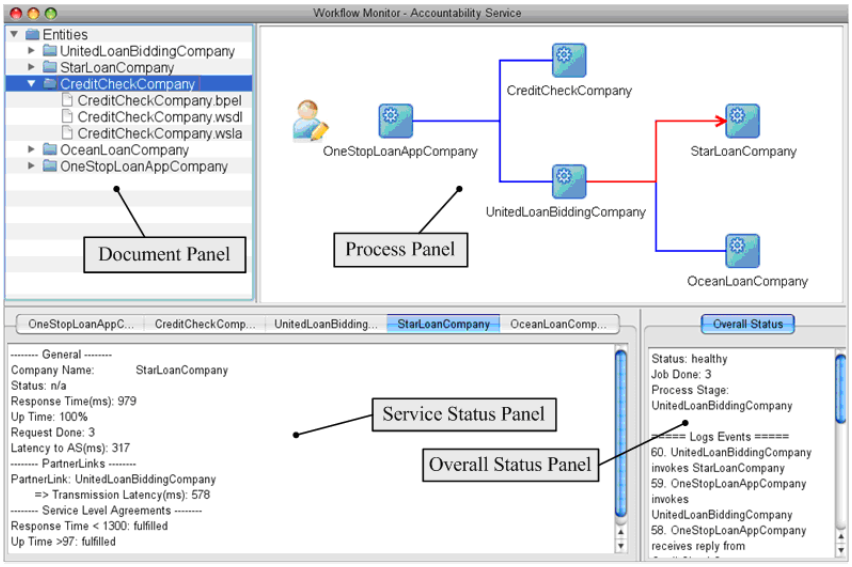
[1] Amazon EC2 http://aws.amazon.com/ec2/

ness process. As in the figure, each of the five services has registered its BPEL, WSDL and WSLA.

Process panel: displays the overview of the business process. Animation is used to show the interactions between the underlying services so that the stage of the current process can be seen.

Service status panel: displays the status and statistics of an individual service. These include general Qos, such as response time and up time; transmission speed with associated PartnerLinks; and fulfillment of the SLAs.

Overall status panel: Overall status panel displays the status and statistics of the whole business process. It shows the number of jobs that have been done, the stage of the current process, and the health of the process concluded by the AS.



Screenshot of mornitoring console

During the operation of the business process, all the logged evidence will be displayed in the console in real time. The console shows the global view across the business process maintained by the AS node. In practice, the console can be deployed remotely to allow the concerned parties or compliance monitoring agents to observe the behavior of the system. A detailed illustration of the monitoring console with a running business process can be seen in our demonstration video [Yao and Chen 2010].

5.3 Performance evaluations

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 those in 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:

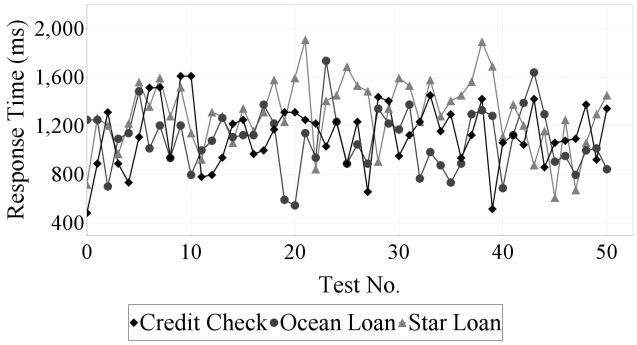
BPEL compliance: According to the business process in Figure 1, the correctness of the operation, as well as the correctness of service invocation are monitored regarding its BPEL.

WS-Agreement compliance: Using the metric in Listing 2, the response time of each of the business services 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 the same as the message received by the invoking service.

Transmission speed: Similar to the metric in Listing 2, another metric can be defined to use the receive time at the invoked service to minus the invoke time at the invoker in order to find the time consumed in the message transmission.

Each of these represents a typical service quality concerns of the client. Figure 8 shows the monitoring results of the response time of three business service nodes. If we apply the WSLA in Listing 2, we can see that although the majority is below the 1.6 seconds guarantee, each of the business services 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.

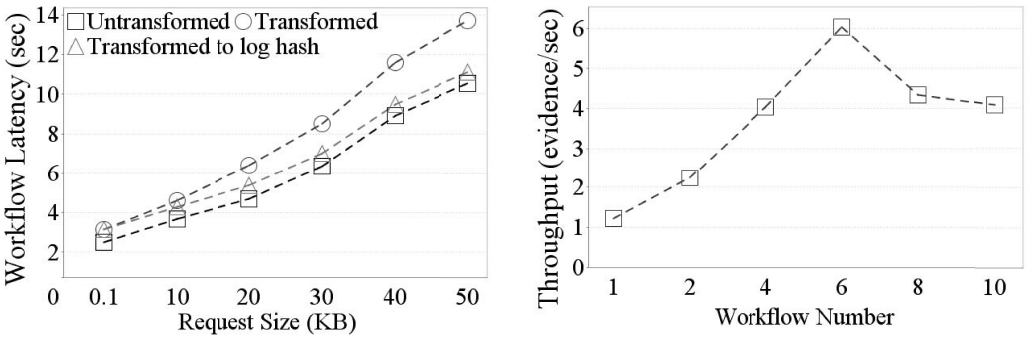


Response time monitoring

We have conducted testing to evaluate the latency introduced by incorporating accountability into business process. Figure 9a shows the overall latency to finish the process 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). For the process with transformed BPEL scripts to log the entire input/output messages (the serie marked with "circles"), the latency introduced compared to the untransformed one (the serie marked with "squares") grows as the request message becomes larger. In percentage terms, on average 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. We can see in the graph, the extra latency is significantly reduced if the BPEL scripts are transformed only to log the hash of the evidence (the serie marked with "triangles"). In fact, the extra latency almost remains constant regardless of the size of the request message, so it becomes more and more negligible when the message size increases.

In practice, it is very rare to log the entire communication message as the evidence. Instead, the hash of the message computed using collision-resistant hash functions (e.g., SHA-1), which is a very small digest (160 bits for SHA-1), can be logged as a substitute. 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. When necessary, such hash can be used to request the logger to submit the evidence. Of course, the hash of the evidence tells little information about the evidence to the AS. Most SLA compliance or business logic compliance cannot be concluded based on hash. But one can choose to submit only the critical part of the evidence and the hash of the rest to effectively reduce the latency while still allowing the AS to validate his compliance.

As aforementioned, one AS node may be monitoring multiple BS nodes, or multiple workflows. Naturally, it is interesting to find out the processing capability of individual AS nodes. To



(a) Latency introduced by incorporating accountability

(b) Throughput of AS under different loads

Performance testing

evaluate this, we replicated the business process we have implemented (the loan application service composition), and invoke multiple business processes replicated concurrently. As such, multiple BS nodes will be submitting evidence to one AS node simultaneously. With this setting, we evaluate the processing throughput of an AS node when it is under different loads (in terms of evidence received per unit time). Figure 9b shows the testing results. In the figure we can see that, the processing throughput of the AS improves as the number of workflows increments, it reaches its peak when the AS is monitoring 6 workflows, and then it decays gradually if more workflows are involved in the monitoring. We tested this with messages of size 50KB, the processing operations conducted by AS involves both SLA and business logic compliance validating, which may need to fetch history data from the data warehouse to make conclusions. Since the computing power of an AS node is fixed, an decrease in message size or processing complexity will shift the peak towards right to occur when more workflows are involved, and vice versa.

Based on above observations, we believe it is reasonable to conclude that the latency introduced by incorporating accountability into existing business processes is acceptable and adjustable. An AS deployed in a small computing instance with limited resources in the cloud is capable of monitoring the compliance of a number of workflows each of which consists of multiple service nodes. Therefore, our approach provides a viable solution to enforce accountability in the practice.

6. RELATED WORK

Service compliance has been decently studied in recent years, conventionally it is referred to as quality of service monitoring (QoS). The conventional QoS monitoring approaches focus on the performance aspect of compliance, such as response time, transmission latency, up-time, etc. and focus on the ease of deployment and measurement accuracy. Typical example of QoS monitoring approach is [Ghezzi et al. 2004], where SLAs requirements are interpreted to generate monitors to be inserted into the process being monitored. These monitors capture related run-time parameters to find out the compliance.

Recent studies in server compliance differ from the conventional QoS monitoring in that, the procedural aspect of compliance is also considered. This implies the capturing or/and analysis of the activities conducted. For example, [Moser et al. 2008] captures the incoming and outgoing messages of service nodes in order to record all the interactions among the services. In [Beeri et al. 2007], procedural requirements are expressed in terms of activity patterns to be matched with the executed actions. COMPAS project [Schumm et al. 2010; Miseldine et al. 2008; Daniel et al. 2009] attempted to extend the business process engine with the functionality to record activity traces, which will be analysed for compliance diagnosis.

In these studies, it is assumed that the collected evidence is not bogus and the incompliant en-

tity will admit the violation once it is discovered. Our work considers a more hostile environment where all service entities are expected to behave in any possible manner and deceive for their own benefit. Some other approaches also utilize cryptography techniques to achieve provability and un-deniability. [Yumerefendi and Chase 2004] 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. [Kim and Kher 2007] 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 amount of usage can not be denied by either party. [Yumerefendi and Chase 2007] uses a similar approach to [Kim and Kher 2007], however it uses such a method to achieve certified accountable tamper-evident storage service. Instead of un-deniable 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.

In our work, we deploy a central authority (AS) for diagnosis. Similar setting is adopted by PlanetFlow [Huang et al. 2006], a layer built into PlanetLab [Spring et al. 2006] to capture and analyse all the network activities conducted on PlanetLab. In LLAMA project [Zhang et al. 2007; Lin and Chang 2009; Lin et al. 2009], evidence are stored locally on the nodes and a central accountability authority will collect the required evidence from the most likely root cause locations when needed. In contrast to central diagnosis, an alternative topology is peer diagnosis, where all the participators collect evidence from each other and verify the compliance for each other. Typical examples includes [Druschel et al. 2007], [Haeberlen et al. 2006] and the study of Byzantine fault tolerance systems [Castro and Liskov 2002].

The concept of our work is generated from [Wang et al. 2009; Wang et al. 2008], in which the authors have proposed the idea that, accountability can be used for verification of services' correctness with regard to established service agreements. Their study has been well developed and refined in our work. We incorporate the AS into the business process through the transformation of BPEL scripts, and the compliance is validated against standard SLA definition.

7. CONCLUSIONS

Collaborations are becoming increasingly important for companies' and organisations' survival. Complex tasks which cannot be done by individuals can be achieved by the joint force when they are combined to form business processes, hence individuals only need to focus on the development of its core strength to maintain its competitiveness. Such business process may span multiple administration domains, therefore the viability of this paradigm critically predicates on a robust mechanism to validate each participants' compliance to the system.

In this paper, we introduce the Accountability Service to enforce compliance on the service providers, who participate in business collaborations in the Cloud. This accountability mechanism can be conveniently incorporated into existing business processes defined with process descriptive languages (e.g. BPEL). With strong accountability enforced, we can build a trustworthy cloud environment, where incompliance can always be concluded with provable and non-disputable evidence. We implemented an evaluation system into Amazon EC2 which shows that, our design i) is easy to be incorporated into existing workflows; ii) can enforce strong accountability in various aspects; iii) does not consume excessive computing resource to provide accountabilty; and iv) introduce acceptable processing latencies.

In the future, we aim to extend our work to adopt various diagnosis techniques to more accurately and efficiently diagnose complex system compliance. For example, QoS compliance is difficult to measure if high accuracy is required, mathematical modelling techniques [Sommers et al. 2010; 2007] can be used to make better estimations. For procedural compliance, the business logic can be fit into a formal model [Aalst et al. 2008] for better analysis of activity conformance. And activity history can be mined to recognize certain patterns [Lou et al. 2010]. Another important capability is the reasoning and inference ability. When confronting disputes,

available evidence shall be gathered for reasoning to deduce the most possible root-of-fault and take appropriate actions [Ruffo and Crispo 2001].

REFERENCES

Aalst, W. M. P. v. d., Dumas, M., Ouyang, C., Rozinat, A., and Verbeek, E. 2008. Conformance checking of service behavior. *ACM Trans. Internet Technology 8,* 3, 1–30.

Andrews, T., Curbera, F., Dholakia, H., et al. 2003. Business process execution language for web services (BPEL4WS) specifications.

Beeri, C., Eyal, A., Pilberg, A., and Milo, T. 2007. Monitoring business processes with queries. In *International Conference on Very Large Database.*

Castro, M. and Liskov, B. 2002. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Computer Systems 20,* 4, 398–461.

Daniel, F., Casati, F., D'Andrea, V., Mulo, E., Zdun, U., Dustdar, S., Strauch, S., Schumm, D., Leymann, F., Sebahi, S., Marchi, F., and Hacid, M. 2009. Business compliance governance in service-oriented architectures. In *International Conference on Advanced Informa&on Networking and Applications.* 113–120.

Druschel, P., Haeberlen, A., and Kouznetsov, P. 2007. Peerreview:practical accountability for distributed systems. In *ACM SIGOPS symposium on Operating systems principles.* 175–188.

Ghezzi, C., Baresi, L., and Guinea, S. 2004. Smart monitors for composed services. In *International Conference on Service Oriented Computing.* 193–202.

Haeberlen, A., Kouznetsov, P., and Druschel, P. 2006. The case for byzantine fault detection. In *Conference on Hot Topics in System Dependability.* 5–10.

Haeberlen, A., Kouznetsov, P., and Druschel, P. 2007. Peerreview: Practical accountability for distributed systems. Technical report, Max Planck Institute for Software Systems. March.

Huang, M., Peterson, L., and Bavier, A. 2006. Planetflow:maintaining accountability for network services. In *ACM SIGOPS Operating Systems Review.* 89–94.

Kim, Y. and Kher, V. 2007. Building trust in storage outsourcing:secure accounting of utility storage. In *IEEE International Symposium on Reliable Distributed Systems.* 55–64.

Lamport, L. 1983. The weak byzantine generals problem. *Journal of ACM 30,* 3, 668–676.

Lin, K.-J. and Chang, S. 2009. A service accountability framework for qos service management and engineering. *Information Systems and E-Business Management 7,* 429–446. 10.1007/s10257-009-0109-5.

Lin, K.-J., Panahi, M., Zhang, Y., Zhang, J., and Chang, S.-H. 2009. Building accountability middleware to support dependable soa. *IEEE Trans. Internet Computing 13,* 2 (mar.), 16 –25.

Lou, J.-G., Fu, Q., Yang, S., Li, J., and Wu, B. 2010. Mining program workflow from interleaved traces. In *ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, New York, NY, USA, 613–622.

Mahbub, K. and Spanoudakis, G. 2004. A framework for requirents monitoring of service based systems. In *International Conference on Service Oriented Computing.* 84–93.

Miseldine, P., Flegel, U., and Schaad, A. 2008. Supporting evidence-based compliance evaluation for partial business process outsourcing scenarios. In *Requirements Engineering and Law.* 31–34.

Moser, O., Rosenberg, F., and Dustdar, S. 2008. Non-intrusive monitoring and service adaptation for ws-bpel. In *WWW '08: Proceeding of the 17th international conference on World Wide Web.* ACM, New York, NY, USA, 815–824.

Mulgan, R. 2000. Accountability: An ever-expanding concept? In *Public Administration.* 555–573.

Papazoglou, M., Traverso, P., Dustdar, S., and Leymann, F. 2007. Service-oriented computing: State of the art and research challenges. *In Trans. IEEE Computer 40,* 11 (nov.), 38 –45.

Ruffo, G. and Crispo, B. 2001. Reasoning about accountability within delegation. In *International Conference on Information and Communications Security.* 251–260.

Schumm, D., Leymann, F., Ma, Z., Scheibler, T., and Strauch, S. 2010. Integrating compliance into business processes. In *Multikonferenz Wirtschaftsinformatik.*

Sommers, J., Barford, P., Duffield, N., and Ron, A. 2007. Accurate and efficient sla compliance monitoring. *SIGCOMM Computer Communication Review 37,* 4, 109–120.

Sommers, J., Barford, P., Duffield, N., and Ron, A. 2010. Multiobjective monitoring for sla compliance. *IEEE/ACM Trans. on Networking 18,* 2 (apr.), 652 –665.

Spring, N., Peterson, L., Bavier, A., and Pai, V. 2006. Using planetlab for network research: myths, realities, and best practices. *SIGOPS Operating System Review 40,* 1, 17–24.

Wang, C., Chen, S., and Zic, J. 2009. A contract-based accountability service model. In *IEEE International Conference on Web Services.* 639–646.

Wang, C., Nepal, S., Chen, S., and Zic, J. 2008. Cooperative data management services based on accountable contract. In *International Conference on Cooperative Information Systems*. 301–318.

Yao, J. and Chen, S. 2010. Monitoring by accountability service - demostration video.

Yumerefendi, A. and Chase, J. 2004. Trust but verify:accountability for network services. In *ACM SIGOPS European Workshop*.

Yumerefendi, A. R. and Chase, J. S. 2007. Strong accountability for network storage. *ACM Trans. Storage 3*, 3, 11.

Zhang, Y., Lin, K.-J., and Hsu, J. 2007. Accountability monitoring and reasoning in service-oriented architectures. *Service Oriented Computing and Applications 1*, 35–50. 10.1007/s11761-007-0001-4.

Jinhui Yao is a PhD student in the School of Electrical and Information Engineering at the University of Sydney. He is also undertaking research internship at the Information Engineering Lab of CSIRO ICT Centre, Australia. His main research areas include trustworthy computing platforms, service oriented architecture, service compliance.

Shiping Chen received the Bachelor degree in electrical engineering from the Harbin University of Technology China, the Master degree in computer system engineering from the Chinese Academy of Sciences (CAS), and the PhD degree in computer science from the University of New South Wales (UNSW), Australia. He is a senior research scientist at CSIRO ICT Centre working on middleware and distributed systems. He also holds an honorary associate with the University of Sydney through co-supervising PhD and Master's students. He is actively involved in service computing research community through publications and PC member services (WWW, ICSOC, ICWS, SCC, etc.). His current research interests include software architecture, secure data storage, and compliance assurance technologies. He is a member of the IEEE.

Chen Wang received his PhD from Nanjing University. He is a research scientist in CSIRO ICT Center. His research interests are primarily in distributed, parallel and trustworthy systems. His current work focus on accountable distributed systems, resource management in cloud computing and the smart grid. Prior to joining CSIRO, he worked as a research fellow in School of Information Technologies at University of Sydney. Dr. Chen Wang has industrial experience. He developed a high-throughput event delivery system and a medical image archive system, which are used by many hospitals and medical centers in USA.

David Levy is an associate professor in Faculty of Engineering and Information Technologies at the University of Sydney. His research passion is building dependable systems, that is, systems that do as expected, with a focus on performance, consistency and security issues. He conducts research in the areas of software architectures, distributed components and middleware, real-time systems and performance engineering.