# Byzantine Fault Tolerant Coordination for Web Services Business Activities*

Wenbing Zhao and Honglei Zhang
Department of Electrical and Computer Engineering
Cleveland State University, 2121 Euclid Ave, Cleveland, OH 44115
wenbing@ieee.org

## Abstract

*In this paper, we present a comprehensive study on the threats towards the coordination services for Web services business activities and explore the most optimal solution to mitigate such threats. A careful analysis of the state model of the coordination services reveals that it is sufficient to use a lightweight Byzantine fault tolerance algorithm that avoids performing expensive total ordering of all request messages. The algorithm and the associated mechanisms have been incorporated into an open-source framework implementing the standard Web services business activity specification and an extension protocol that enables the separation of the coordination tasks from the business logic. The performance evaluation results obtained using the working prototype confirm the optimality of our solution.*

**Keywords**: Web Services, Business Activity, Distributed Transaction, Byzantine Fault Tolerance

## 1. Introduction

Service-oriented computing and the Web services technology are transforming the World Wide Web from a predominantly publishing platform to a programmable distributed computing platform, which has led to more and more business activities conducted online. The Web services business activity (WS-BA) specification [8] has recently been ratified by OASIS to standardize the activation, registration, propagation and termination of Web services business activities. However, WS-BA does not specify a standard way for an initiator to communicate with the WS-BA coordinator. Even though this design choice makes it easy for vendors to integrate WS-BA coordination functionalities into their business process engines, it is difficult to ensure interoperability among initiators and coordinators from different vendors [6]. Since a business activity almost inevitably will involve multiple enterprises, expecting or requiring all such enterprises to use and trust a single vendor's product seems to be unrealistic. To address this issue, Erven et al. [6] proposed an extension to the WS-BA specification, referred to as Web Services-BusinessActivity-Initiator protocol (WS-BA-I) to enable the separation of the coordination functionality and the business logic, and to standardize the interactions between the initiator and the coordinator.

This extension makes it possible for a third party to offer coordination services to enterprises who wish to conduct Web services business activities with minimum modification to their workflow engines. Obviously, in order for such coordination services to be widely adopted, they must ensure high degree of security and dependability. This work is aimed to provide a lightweight solution to enhance the security and dependability of the WS-BA coordination services.

In this paper, we carefully analyze the threats to the WS-BA coordination services, and explore strategies to mitigate such threats. We choose to avoid using generic Byzantine fault tolerance (BFT) algorithms designed for general-purpose stateful server replication due to their high cost [4]. In stead, we propose to use a lightweight and efficient BFT algorithm to ensure reliable delivery of messages by exploiting the specific state model defined in WS-BA. Our algorithm does not guarantee total ordering of all messages, as it is not needed in our case. It does, however, ensure that all messages are delivered at correct replicas in their source order, despite the presence of Byzantine faulty replicas and clients. The proof of correctness of our algorithm is provided. Furthermore, we implemented our BFT algorithm and the associated mechanisms, and incorporated them into a well-known open source framework that implements the WS-BA standard with the WS-BA-I extension [2]. The performance evaluation of the prototype confirms the efficiency of our algorithm.

## 2. Web Services Business Activity

A Web services business activity consists of an initiator, a coordinator, and several participants, as shown in Figure 1. All business activities are started and terminated by the initiator. The initiator also propagates the business activity to other participants (through a coordination context included in the requests to other participants). The outcome of the

*This work was supported by a Faculty Research Development Award at Cleveland State University.

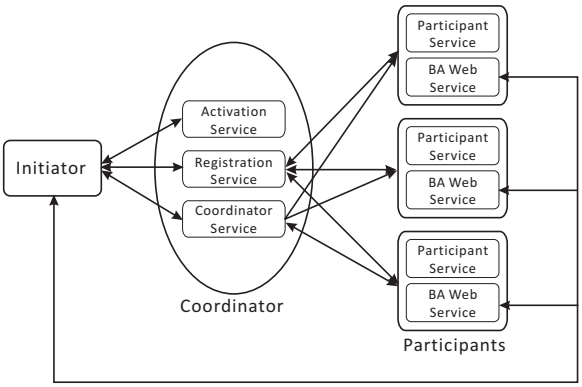


**Figure 1. WS-BA Architecture.**

business activity is determined by the initiator according to its business logic. In this section, we briefly introduce the WS-BA standard and the WS-BA-I extension. We also show an example business activity.

### 2.1. WS-BA

The WS-BA specification [8] describes how to coordinate long running business activities where the atomic transaction model is not appropriate. WS-BA is built on top of the WS-Coordination framework [7]. It specifies two coordination types, Atomic-Outcome and Mixed-Outcome, and two coordination protocols used between the coordinator and a participant, Business-Agreement-with-Participant-Completion (BAwPC) and Business-Agreement-with-Coordinator-Completion (BAwCC).

In WS-BA, a participant registers either one of the two protocols, which are managed by the coordinator of the business activity. The two protocols are briefly summarized below. For detailed state transitions, readers are referred to the WS-BA specification [8].

A participant who has registered the BAwPC protocol informs its coordinator by sending a "Completed" notice when it has done its entire work for a business activity. The coordinator should reply with either a "Close" or a "Compensate" message depending on the circumstances. The participant receives a "Close" instruction if the activity has completed successfully. If it gets a "Compensate" instruction instead, it will undo the completed work and will have to restore the data recorded from the initial condition. The participant may encounter a problem or fail during the processing of the activity, in which case, it must signal the coordinator with a "Fail" message. If it gets the "Fail" message, the coordinator will acknowledge the participant with a "Failed" notification. Upon receiving a "CannotComplete" notification, the coordinator learns that the participant cannot successfully finish its work. On sending out this message, the participant discards all its pending work and cancels all related executions, and exits the current business activity. On receiving such a message, the coordinator is required to notify the participant with a "NotCompleted" message. In the active state, the coordinator could cancel any work by using the "Cancel" notification, and the participant will respond with a "Canceled" message if it receives the message.

In the BAwCC protocol, the completion notification comes from the coordinator. The coordinator sends a "Complete" message to the participants informing them that they won't receive any new requests within the current business activity and it is time to complete the processing. The participant then replies with a "Completed" message if it could successfully finish its work. Other interactions between the coordinator and the participants are similar to those in BAwPC protocol.

WS-BA also defines a set of coordinator-side services and a participant service at the participant-side. The coordinator-side services include Activation, Registration and Coordinator services, and they run in the same address space. For each business activity, all but the Activation Service are provided by a (distinct) coordinator object. Further details regarding these services are introduced in Section 3.

### 2.2. WS-BA-I

The WS-BA-I protocol [6] is an extension to the WS-BA specification. It describes how the initiator should interact with the coordinator, which is lacking in the WS-BA specification. The WS-BA-I protocol is analogous to the completion protocol defined in the Web services atomic transaction specification [11]. To facilitate the WS-BA-I protocol, the coordinator exposes a number of additional operations invocable by the initiator, including one for the initiator to query the state of the business activity, one to create invitation tickets, and a set of operations for the initiator to pass its instructions to the coordinator to complete (for the BAwCC protocol), close, cancel, or compensate a business activity for each participant.

WS-BA-I chooses to use the "pull" model on the initiator side, so that the initiator can operate behind a firewall or a NAT box. To find the latest state of a business activity, the initiator must periodically check with the coordinator.

### 2.3. Example

The normal execution steps of a travel reservation example (adapted from [2] and used in our performance evaluation) are shown in Fig. 2. Due to space limitation, no further explanation is provided.

## 3. Threat Analysis

In this section, we analyze potential threats that could compromise the integrity of the coordination services for Web services business activities. We do not consider the general threats that could target any online services, such as distributed denial of attacks. Furthermore, an entity could refuse to participate the protocols hoping to reduce the availability of the coordination services. This type of threats can be trivially mitigated by replication, therefore, we do not discuss these threats further.

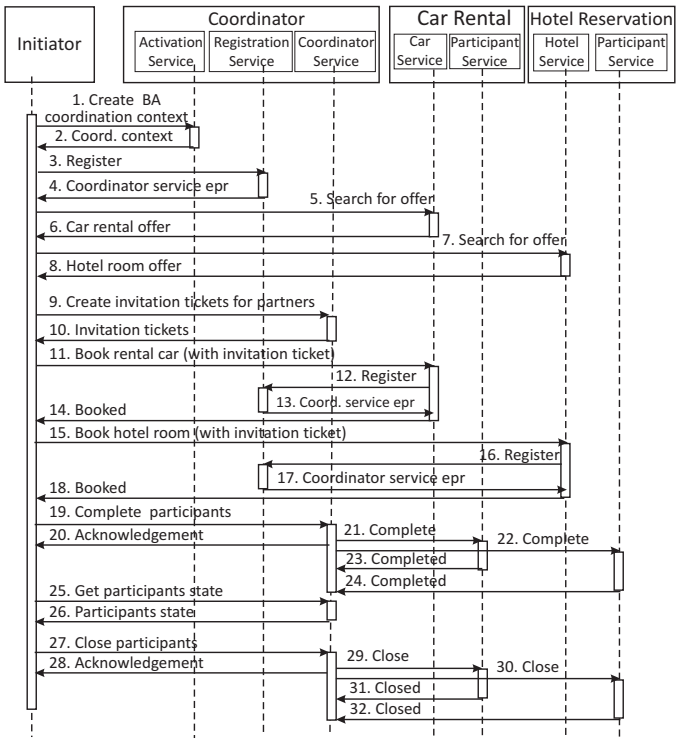


**Figure 2. The sequence diagram showing the detailed steps for a travel reservation example using WS-BA with WS-BA-I extension.**

### 3.1. Threats from a Faulty Coordinator

*Threats towards the activation service.* At the beginning of each business activity, the initiator requests the activation service to generate a coordination context for the business activity. The activation service also creates a coordinator object designated to handle the new business activity. The coordination context contains two main components: a coordination identifier, and the endpoint reference for the registration service. The coordination identifier must be unique for each business activity, and it is used to associate the participants in the same business activity. The registration service endpoint reference is used by a Web service to register its participant service.

A faulty coordinator could (1) reuse an old coordination identifier, (2) use an identifier that is easily predictable, or (3) use an identifier that belongs to a different business activity (which is equivalent to the reuse of the same coordinator object that is created for another business activity). Case (1) could potentially lead to a replay attack, *i.e.,* an adversary registers as a participant for a business activity that it does not belong to, and subsequently replays some other messages in the scope of the same business activity. This threat can be easily mitigated by transport-level security mechanisms such as the use of nonce or timestamp, without resorting to replication. Case (2) could open the door for an adversary to register with the business activity without being invited. However, this does not pose a real threat to the business activity because the outcome of the business activity is determined by the initiator. The initiator could easily spot and subsequently exclude the unsolicited adversary from the business activity. Case (3) could lead to serious consequences if not mitigated because two unrelated business activities would appear to be a single one to the participants. This may confuse both the participants and the initiators of the two affected business activities. The BFT algorithm described in Section 4 is aimed to mitigate such threats.

*Threats towards the Registration Service.* A Web service registers with the registration service by providing an endpoint reference to its participant service, as soon as a business activity is propagated to the location. The registration request must contain a valid coordination identifier with a matchcode (introduced in WS-BA-I [6]) as a way to authenticate the participant. The response to the registration request contains an endpoint reference to the coordination service.

A faulty coordinator could (1) accept a registration request with a illegitimate credential, (2) accept a correct registration request, but assign the participant to wrong coordinator object, or (3) return an invalid endpoint reference. These threats cannot be easily mitigated using transport-level security mechanisms, and BFT replication seems to be the only viable control.

*Threats towards the Coordination Service.* The coordination service interacts with the participants via the BAwCC or BAwPC protocols, and it interacts with the initiator via the WS-BA-I protocol.

From the coordinator-side, the BAwCC protocol is responsible to notify the participants the completion of the business activity, acknowledge the reports sent by the participants, and inform them the final decisions regarding the tasks they have completed. The coordinator also expects to receive notifications from the participants.

A faulty coordinator could (1) send a notification, such as Complete, Close, Compensate, to a participant, without the authorization from the initiator, (2) ignore some reports from the participants, especially the failure notifications such as Fail and CannotComplete, or (3) make arbitrary state transitions without receiving any report from participants. Case (1) can be addressed by a piggybacking mechanism [16], *i.e.,* all such notifications must carry a security certificate including the original signed authorization for the action from the initiator. Case (2) and (3) would lead to a state at the coordinator inconsistent with the participants, which may ultimately affect the initiator's decision on the outcome of the business activity. BFT replication as described in Section 4.2 is an effective way to handle such threats.

The BAwPC protocol is rather similar to the BAwCC protocol, except that the coordinator is no longer responsible for notifying the participants the completion of the busi-

ness activity, instead, it collects "Completed" reports from the participants. Consequently, the threats that a faulty coordinator posts to the coordination service in the BAwPC protocol is similar to those in the BAwCC protocol, and they can be handled in a similar manner.

In the WS-BA-I protocol, the coordinator is responsible to create an invitation ticket upon request from the initiator. The ticket is an extended coordination context, containing an additional identifier (called matchcode) for the participant to be invited to join the business activity. The coordinator also takes requests from the initiator to obtain the latest status of each participant, and to complete (if the BAwCC protocol is used), close, or compensate a business activity. A faulty coordinator, therefore, could (1) return an invalid invitation ticket to the initiator, such as an old invitation ticket, a ticket that belongs to another participant, or an easy-to-predict ticket, (2) present an incorrect state to the initiator, which might confuse the initiator, or (3) alter the instructions issued by the initiator to complete, close, or compensate a business activity.

The possible replay attack caused by case (1) can be mitigated by transport-level security mechanisms, as we have discussed before. For other threats in cases (1) to (3), BFT replication as described in Section 4.2 appears to be an effective control.

### 3.2. Threats from a Faulty Participant

*Threats towards the Coordinator.* A faulty participant could (1) lie about its execution status, and send the coordinator reports inconsistent with its internal state, and (2) send conflicting reports to different coordinator replicas. Case (1) can be prevented by replicating each participant using state-of-the-art BFT techniques. However, since this paper focuses on the protection of the coordination services, we do not discuss how this could be achieved. In fact, it is impractical for an independent WS-BA coordination services provider to expect that all participants be protected by BFT techniques, which virtually requires that none of the participants could fail. Consequently, this type of threats is best addressed by insisting on the use of digital signatures with all messages exchanged between the participants and the coordinator, and by logging all messages to and from the participants at the coordinator. Case (2) can be addressed by our BFT algorithm described in Section 4.

*Threats towards the Initiator.* A faulty participant could attack the initiator in a similar way to that on the coordinator. Again, it is suggested that the initiator insists on the use of digital signatures with all messages exchanged with the participants, and logs them. It is conceivable for the initiator to give preference to the participants who offer higher degree of quality of services that minimize such threats.

### 3.3. Threats from the Initiator

Since a business activity is always initiated, and its progress and outcome is controlled, by the initiator, the integrity of the business activities that originated at a Byzantine faulty initiator cannot be guaranteed. Similar to the circumstances for a faulty participant, we can prevent this from happening by replicating the initiator and employing BFT techniques. Again, due to the focus of this paper, we do not elaborate how this could be achieved. Absent from this strong requirement, the participants and the coordinator must resort to non-repudiation and logging techniques to hold the faulty initiator accountable.

When replication is used by the coordination service provider, a new threat scenario could occur, *i.e.,* a faulty initiator could send conflicting requests to different coordinator replicas. This threat could readily be addressed by our BFT algorithm in Section 4.2.

## 4. Byzantine Fault Tolerant Coordination

In this section, we describe a lightweight BFT algorithm and a set of mechanisms necessary to achieve Byzantine fault tolerant coordination for business activities. An important objective is to enable an independent third party to launch a highly dependable and trustworthy coordination service for business activities that might span multiple enterprises. As such, the practicability of the BFT solution is essential, which means that it must be lightweight with low runtime overhead and good scalability.

Traditional state-machine based BFT techniques could be used to mitigate the threats to the coordination of business activities. However, it is not wise to use them naively to protect the coordination services. These techniques are designed to protect *general-purpose stateful* servers from Byzantine faults, and therefore, all incoming requests are being totally ordered, which makes the mechanisms heavyweight and incurs significant runtime overhead. Since the state model of the coordination services is well-defined, there is no reason why we should not exploit this knowledge and use an optimal solution customized for this specific application.

### 4.1. State Model Analysis

Requests for different business activities are handled completely independently, *i.e.,* their relative ordering does not have any impact on the state transitions. This is obvious for registration and coordination related requests since they are handled by different coordinator objects. Even though the activation requests are handled by the same object, their relative ordering does not affect how the coordinator objects will be created. The generation of coordination identifiers can be a concern for replica consistency. We handle this replica nondeterminism issue without resorting to inter-replica coordination (to be described in Section 4.3), which obviates the necessity to run expensive Byzantine agreement among the replicas.

Next we consider the requests within the same business activity. The relative ordering of the activation and registration requests are causally related, *i.e.,* the activation request must precede the registration request. This order is known

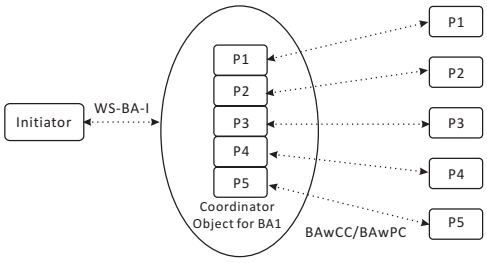


**Figure 3. The state model for WS-BA with WS-BA-I extension. The partitioning of the state for each participant is highlighted.**

and can be programmed directly into the BFT framework without resorting to inter-replica coordination. It is straightforward to ensure the source ordering (*e.g.,* by using a sequence number) for requests sent by each participant in the BAwCC or BAwPC protocol. The same argument holds true for requests from the initiator.

Non-duplicate requests sent by different participants to the same coordinator object only modify their respective partitions of the state, as highlighted in Figure 3. (Duplicate requests participating the BAwCC, BAwPC, and WS-BA-I protocols do not change the coordinator state, although they may trigger the resending of some commands.) The change of one partition of the coordinator object state has no direct impact on another partition associated with a different participant. The coordination object uses each of these partitions to keep track of the state with each participant according to the BAwCC or BAwPC protocol. Therefore, it is unnecessary to order requests from different participants.

The only remaining issue is if we should order the requests from the initiator relative to those from the participants of the same business activity. The answer is no. The requests from the initiator can be categorized into three types, according to the WS-BA-I extension.

The first type of requests is to create invitation tickets for the participants, one at a time. This type of requests obviously can be interleaved with the requests from existing participants of the business activity because they are handled by different objects.

The second type of requests is to query the state of the business activity, which is read-only. Even though these requests do not change the coordinator state, different replicas might report different state status to the initiator if the state-finding requests are not ordered with respect to the requests from the participants. We believe this is not a concern because the initiator can keep querying the coordinator replicas until their state converges.

The third type of requests is to instruct the coordinator to terminate (compensate, close, cancel, or compensate) the business activity. If these requests are not ordered with respect to the messages from the participants, when such an instruction arrives, some replicas might have evolved into a different state since the initiator last queried their state. This is not a concern because according to the WS-BA-I design, the state of the coordinator object might be inconsistent with that of the initiator even without replication. Consider the following scenario. The initiator sends a "Cancel" command to the coordinator for a particular participant when the last seen state for that participant is "completing". When the "Cancel" command is delivered, however, the "Completed" report from the participant might have arrived, in which case, the coordinator should send a "Compensate" command to the participant instead of "Cancel". This mechanism has already been built into the coordination framework.

Therefore, we conclude that there is no need to ensure a total ordering for the messages involved in a business activity at the coordinator replicas.

### 4.2. A Lightweight BFT Algorithm

We propose a lightweight BFT algorithm in light of the state model analysis. The algorithm fulfills the following two objectives:

O1. If a request is delivered at a correct replica, it must eventually be delivered at all correct replicas according to the sender's source order.

O2. In the event of a faulty client sending conflicting requests (that carry the same sequence number) to correct replicas, only one of the requests may be delivered to all correct replicas, if at all.

We assume that $3f+1$ coordinator replicas are available, among which at most $f$ can be faulty. The initiator and the participants are not replicated (our algorithm can be trivially modified if these entities are in fact replicated). There is no limit on the number of faulty participants. The initiator could be faulty as well.

Each coordinator replica is assigned a unique id $k$, where $k$ varies from $0$ to $3f$. We assume that the algorithm operates in an asynchronous distributed environment. However, we assume that the network is reliable. In particular, if a correct participant/initiator sends a message to a correct coordinator replica, the message will reliably arrive at the replica eventually. The same is true for messages exchanged between correct replicas. This assumption can be easily satisfied by using TCP communication, or by using the mechanisms defined in the Web Services Reliable Messaging (WS-RM) standard [5].

All WS-BA messages, *i.e.,* the messages exchanged between the coordinator and the initiator and those between the coordinator and the participants, carries a monotonically increasing sequence number. The sequence number is unique only within the respective connection between the coordinator and its client (*i.e.,* the initiator or the participant), but each connection is uniquely identified. For each connection, the first request is assigned a sequence number $0$, and the sequence number is incremented for each subsequent request. The reply, if any, carries a sequence num-

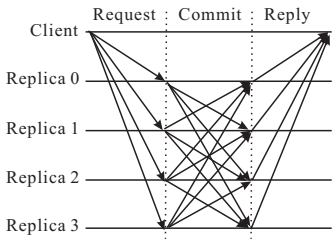


**Figure 4. Normal operation of the lightweight BFT algorithm with $f = 1$.**

ber matching that of the corresponding request. The same holds true for requests sent from the coordinator to the participants in the BAwCC or BAwPC protocols. Note that all messages defined in BAwCC and BAwPC protocols are oneway messages.

All messages between the coordinator and participants are timestamped (to prevent the replay attack) and digitally signed (to ensure accountability and to prevent spoofing). We assume that the coordinator replicas, the initiator, and the participants each has a public/private key pair. The public key is known to all of them, while the private key is kept secret to its owner. We assume that the adversaries have limited computing power so that they cannot break the digital signatures created by non-faulty entities.

The normal operation of the BFT algorithm is shown in Figure 4 (for $f = 1$). A client (*i.e.,* the initiator or a participant) sends the request to all coordinator replicas. The request has the form $<\text{REQUEST}, s, o>_{\sigma_c}$, where $s$ is the sequence number, $o$ is operation to be invoked (including the coordination context if needed) and $\sigma_c$ is the signature of the message signed by the client.

Upon receiving a request, a replica verifies the signature, and checks the validity of the requested operation and if the sequence number carried with the request matches the next expected sequence number. A replica broadcasts a commit message to all other replicas if the request passes the verification. The commit message has the form $<\text{COMMIT}, k, s, d>_{\sigma_k}$, where $d$ is the digest of the request message, and $k$ is the replica number. Note that all replicas play an equal role, in particular, no replica acts the primary since total ordering is not needed (*i.e.,* the sequence number is assigned by the client instead of the primary).

When a replica receives both the request and $2f$ matching commit messages from other replicas, it delivers the request and makes the state transition. The reply, if any (recall that many WS-BA messages are oneway), has the form $<\text{REPLY}, k, s, r>_{\sigma_k}$, where $r$ is the response. Some operation might trigger the sending of nested requests to the participants (*e.g.,* the "Complete" instruction from the initiator will cause the coordinator to send the same command to the designated participant). Such nested requests (not shown in Figure 4) have the form $<\text{NESTED\_REQUEST}, k, s, o>_{\sigma_k}$, which is similar to that of regular requests with an additional field indicating the sending replica number.

If a replica receives a commit message before it receives the referenced request, it requests a retransmission of the request from the replica that sent it the commit message.

If a replica receives $f + 1$ consistent commit messages, but the digest in the commit message is different from the request it has received, it requests a retransmission of the missing request from the $f + 1$ replicas, logs the event, and abandon the original request.

If the client (*i.e.,* the initiator or the participant) that issued the request expects a reply, it must collect at least $f+1$ matching replies sent by different replicas before it delivers the reply. The same mechanism is used for the participants to handle (nested) requests issued by the coordinator replicas. By collecting $f + 1$ matching replies, it is guaranteed that at least one of them is sent by a correct replica.

### 4.3. Additional Mechanisms

In a typical implementation of the WS-BA standard, the coordination identifier is generated by the activation service (*e.g.,* by using a UUID). This would cause replica inconsistency when the activation service is replicated. Even though this replica nondeterminism could be controlled by using the mechanism described in [15], it appears to be unnecessary in light of our threat analysis in Section 3. A more efficient method to handle this issue is to piggyback a UUID in the activation request sent by the initiator. One implication of this strategy is that the replicas must now keep a history of the UUID used and verify the uniqueness of the newly proposed UUID against its record.

Another mechanism is the piggybacking of security certificates as mentioned in Section 3. A security certificate is included in every command issued by the coordinator to the participants in the BAwCC or BAwPC protocol. It consists of the original signed authorization for the action from the initiator, which includes not only the specific command, but the coordination identifier and timestamp as well to prevent the replay attack. On receiving such a command, a participant verifies if the command from the replica is consistent with that in the certificate. The command is ignored if a mismatch is detected.

### 4.4. Proof of Correctness

We now provide an informal proof of the correctness of our Byzantine faulty tolerance algorithm with respect to the two objectives laid out in Section 4.2.

*Proof of O1:* We first consider the case when the sender is correct. Because we assume the communication is reliable, O1 is obviously satisfied because the correct sender can establish a connection with each correct replica and send the request reliably to all correct replicas.

If the sender is faulty, it may send the request to only a fraction of correct replicas, or it may send conflicting requests (with the same sequence number) to different correct replicas. In both cases, the message exchange in the commit phase ensures that if a correct replica delivers a request, it will propagate the message to any other correct replica if

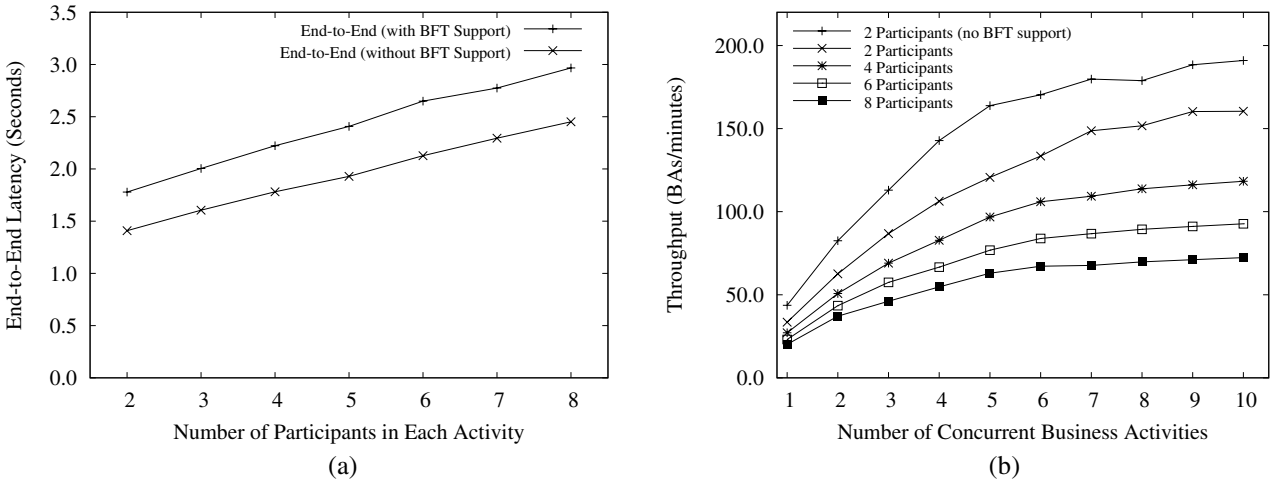


**Figure 5. (a) End-to-end latency measurements for business activities with different number of participants under normal operation. (b) Throughput of the coordination services for business activities with different number of concurrent business activities.**

necessary. This is because before a correct replica delivers a request, it must have received $2f$ matching commit messages, in addition to the request. Since at most $f$ replicas are faulty, this means at least $f + 1$ correct replicas have received the same request. In the former case, if a replica did not receive the request, it will eventually receive the commit messages sent by the $f + 1$ correct replicas, which will prompt it to request a retransmission of the request. In the latter case, using the same reasoning, we know that at least $f + 1$ correct replicas have received the same request. The commit messages from these replicas will arrive at every other correct replica, which will trigger them to request for retransmission of the request that had been delivered at other replicas.

*Proof of O2:* This is a special case that we have considered while proving O1. We have shown that if the faulty client has sent the same request to $f + 1$ or more correct replicas, it will be delivered by all correct replicas. Otherwise, the request is effectively ignored since no correct replica can collect $2f$ matching commit messages. More rigorously, O2 can be proved by contradiction. Assume that a correct replica $i$ delivers a request $R$ and another correct replica $j$ delivers a different request $R'$ sent by the same faulty client with the same sequence number and connection id. It must be true that a quorum of $2f+1$ replicas, $Q1$, have accepted $R$ and sent the commit message for $R$, and similarly, a quorum of $2f + 1$ replicas, $Q2$, have accepted $R'$ and sent the commit message for $R'$. Because there are $3f + 1$ replicas, $Q1$ and $Q2$ must intersect at $f + 1$ or more replicas. Since at most $f$ replicas are faulty, at least one of the replicas in the intersection must be correct. This contradicts our assumption about the correct replica because it will never accept two different requests with the same sequence number and connection id. The above arguments also show the optimality of using $3f + 1$ replicas to achieve Byzantine fault tolerance.

## 5. Implementation and Performance

We have implemented the lightweight BFT algorithm and the associated mechanisms, and incorporated them into the Kandula framework [2], which is a Java-based, open source implementation of the WS-BA specification with the WS-BA-I extension. The extended framework is based on a number of Apache Web services projects besides Kandula, including WSS4J [3], and Apache Axis [1]. Most of the mechanisms are implemented in terms of Axis handlers that can be plugged into the framework without affecting other components. Some of the Kandula code is modified to enable the control of its internal state, to enable Byzantine fault tolerant delivery of requests at the coordinator, and to enable voting at the initiator and the participants. Due to space limitation, the implementation details are omitted.

Our experiment is carried out on a testbed consisting of 20 Dell SC440 servers connected via an 100Mbps Ethernet. Each server is equipped with an Intel Pentium D 2.8GHz processor and 1GB memory running SuSE 10.2 Linux.

The test application is the travel reservation example that we have shown in Figure 2. The coordinator is replicated on 4 nodes to tolerate one faulty replica. The set of initiators and participants run on distinct nodes. Each initiator launches and terminates business activities continuously in a loop without any think time. In each run, 1000 samples are obtained. In our experiment, we use the Atomic-Outcome with the BAwCC protocol. Furthermore, all messages are protected using timestamped digital signatures. The end-to-end latency for each business activity is measured at the initiator. The throughput of the coordination framework is measured at the coordinator.

The end-to-end latency measurement results with and without replication for a single business activity are shown

in Figure 5(a). Comparing with the non-replicated case, the end-to-end latency increases only modestly when replication is enabled using our algorithm. As can be seen, the latency overhead is consistently less than 20% in our measurements. The throughput (in terms of number of business activities per minute) measurement results are summarized in Fig. 5(b). To avoid cluttering, only the 2-participant case is shown for the no-replication configuration. As can be seen, the throughput reduction is less than 20% as well when replication is enabled, which confirms the optimality of our algorithm design and implementation.

## 6. Related Work

Byzantine fault tolerance has been of great research interest for the passed two decade [4, 9, 10, 12, 13, 16, 15, 14]. Most of the prominent research work (*e.g.,* [4]) has focused on the developing of generic BFT algorithms for general-purpose stateful servers.

The application of BFT techniques to transactional systems is first reported by Mohan et al. [13]. In [13], the two-phase commit is enhanced by performing a Byzantine agreement among all nodes in the root-cluster on the outcome of the atomic transaction. Recently, we revisited this problem and proposed a more efficient solution [16] by restricting the members who participate the Byzantine agreement to the coordinator replicas.

Even though the problem of coordination for atomic transactions bears some similarity with that for business activities, there are a number of significant differences. In atomic transactions, the coordinator and the participants are much tightly coupled. Any participant could unilaterally abort a transaction. Furthermore, the coordinator could decide on the outcome of the transactions, based on the votes collected in the two-phase commit protocol. For business activities, however, the outcome is solely decided by the initiator according to its business logic, and a faulty participant at best might be able to exert some influence, but not decide, on the activity outcome. This is why we could use a much lighter weight BFT solution.

The application of BFT techniques to general Web services has been reported in [12, 14]. Even though the solutions proposed could be used to protect the coordination services against Byzantine faults, they are unnecessarily too expensive (such systems typically incur 100% or higher overhead in end-to-end latency in similar scenarios, as reported in [14]). As shown in this paper, by considering the specific state model of the coordination services, our solution is much more lightweight and performs significantly better (20% vs. 100%).

## 7. Conclusion

In this paper, we presented a comprehensive study on the potential threats towards the coordination services for Web services business activities and proposed an optimal solution to mitigate such threats. We carefully examined the state model of the coordination services and argued that it is unnecessary to perform the total ordering of all requests. It is sufficient to ensure reliable delivery of messages according to their source order. This enabled us to design a lightweight BFT algorithm that avoids most of the runtime overhead associated with generic BFT algorithms. We have implemented and incorporated the algorithm into an open-source framework implementing the WS-BA standard and the WS-BA-I extension. The performance evaluation results obtained using the working prototype show very moderate overhead, confirming the optimality of our solution.

## References


[1] Apache Axis. http://ws.apache.org/axis/.

[2] Apache Kandula. http://ws.apache.org/kandula/.

[3] Apache WSS4J. http://ws.apache.org/wss4j/.

[4] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, November 2002.

[5] D. Davis, A. Karmarkar, G. Pilz, S. Winkler, and U. Yalcinalp. *Web Services Reliable Messaging, Version 1.1, OASIS Standard*, January 2008.

[6] H. Erven, H. Hicker, C. Huemer, and M. Zapletal. The Web Services-BusinessActivity-Initiator (WS-BA-I) Protocol: an extension to the Web Services-BusinessActivity specification. In *Proceedings of the IEEE International Conference on Web Services*, Salt Lake City, Utah, July 2007.

[7] M. Feingold and R. Jeyaraman. *Web Services Coordination Version 1.1, OASIS standard*, July 2007.

[8] T. Freund and M. Little. *Web Services Business Activity Version 1.1, OASIS standard*, April 2007.

[9] H. Garcia-Molina, F. Pittelli, and S. Davidson. Applications of Byzantine agreement in database systems. *ACM Transactions on Database Systems*, 11(1):27–47, 1986.

[10] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[11] M. Little and A. Wilkinson. *Web Services Atomic Transactions Version 1.1, OASIS standard*, April 2007.

[12] M. Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou, and P. Narasimhan. Thema: Byzantine-fault-tolerant middleware for web services applications. In *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, pages 131–142, 2005.

[13] C. Mohan, R. Strong, and S. Finkelstein. Method for distributed transaction commit and recovery using byzantine agreement within clusters of processors. In *Proceedings of the ACM symposium on Principles of Distributed Computing*, pages 89–103, Montreal, Quebec, Canada, 1983.

[14] W. Zhao. BFT-WS: A Byzantine fault tolerant framework for web services. In *Proceedings of the Middleware for Web Services Workshop*, Annapolis, MD, October 2007.

[15] W. Zhao. Byzantine fault tolerance for nondeterministic applications. In *Proceedings of the IEEE International Symposium on Dependable, Autonomous and Secure Computing*, pages 108–115, Columbia, MD, September 2007.

[16] W. Zhao. A Byzantine fault tolerant distributed commit protocol. In *Proceedings of the IEEE International Symposium on Dependable, Autonomous and Secure Computing*, pages 37–44, Columbia, MD, September 2007.