11-15-2011

Towards Trustworthy Coordination for Web Services Business Activities

Hua Chai
Cleveland State University

Hongle Zhang
Cleveland State University

Wenbing Zhao
Cleveland State University, w.zhao1@csuohio.edu

P. M. Melliar-Smith
University of California

L. E. Moser
University of California

Publisher's Statement

Repository Citation

Chai, Hua; Zhang, Hongle; Zhao, Wenbing; Melliar-Smith, P. M.; and Moser, L. E., "Towards Trustworthy Coordination for Web Services Business Activities" (2011). *Electrical Engineering & Computer Science Faculty Publications*. 265.
https://engagedscholarship.csuohio.edu/enece_facpub/265

Toward Trustworthy Coordination of Web Services Business Activities

Hua Chai, Honglei Zhang, Wenbing Zhao, *Member*, *IEEE*,
P. Michael Melliar-Smith, *Member*, *IEEE*, and Louise E. Moser, *Member*, *IEEE*

Abstract—We present a lightweight Byzantine fault tolerance (BFT) algorithm, which can be used to render the coordination of web services business activities (WS-BA) more trustworthy. The lightweight design of the BFT algorithm is the result of a comprehensive study of the threats to the WS-BA coordination services and a careful analysis of the state model of WS-BA. The lightweight BFT algorithm uses source ordering, rather than total ordering, of incoming requests to achieve Byzantine fault tolerant, state-machine replication of the WS-BA coordination services. We have implemented the lightweight BFT algorithm, and incorporated it into the open-source Kandula framework, which implements the WS-BA specification with the WS-BA-I extension. Performance evaluation results obtained from the prototype implementation confirm the efficiency and effectiveness of our lightweight BFT algorithm, compared to traditional BFT techniques.

Index Terms—Business activity, Byzantine fault tolerance, distributed transaction, service-oriented computing, trustworthy computing, web services

1 Introduction

Service-oriented computing and web services are transforming the World Wide Web from a publishing platform into a distributed computing platform, resulting in more and more business activities being conducted online. Such business activities, offered in the open environment of the Internet, often involve financial, healthcare and other web services that their users require to be trustworthy.

The web services business activity (WS-BA) specification [14], developed by OASIS, standardizes the activation, registration, and coordinator services of web service business activities. A business activity consists of an initiator, a coordinator, and one or more participants, as shown in Fig. 1. A business activity is started and terminated by the initiator. The initiator propagates the business activity to the participants using a coordination context in the requests. The outcome of the business activity is determined by the initiator according to the business logic.

In this paper, we use a travel reservation application, shown in Fig. 2, as a running example, to illustrate the problem we address and our proposed solution. The travel reservation application is a composite web service with which the clients interact directly. Behind the scenes, the application consists of an initiator and two web services, an airline reservation web service and a hotel reservation web service. These two web services are typically provided by third-party service providers, independent of the owner of the travel reservation composite web service.

When a client invokes the composite web service, the initiator creates a web service business activity, in which it communicates with the airline reservation web service to make an airline reservation and the hotel reservation web service to reserve a hotel room. The coordinator component ensures that the business activity is properly propagated to the airline reservation and hotel reservation web services, and is properly terminated according to a predefined policy. The coordinator component provides the activation, registration, and coordinator services. More details on the travel reservation application are provided in Section 2.3.

As can be seen from the previous example, the trustworthiness of the coordination service is crucial to the business activities involved. It is very desirable to have the coordinator replicated for Byzantine fault tolerance (BFT). Unfortunately, the WS-BA specification does not specify a standard way for an initiator of a business activity to communicate with the coordinator of the business activity. This design choice makes it easy for vendors to integrate WS-BA coordination functionalities into their business process engines (e.g., the travel reservation application could integrate the initiator and the coordinator into the same process), but makes it difficult to ensure interoperability among the initiators and coordinators of different vendors [12]. Such interoperability is needed because of the benefits of separating the initiator and the coordinator, in particular

- A small company might create a composite web service (such as a travel reservation service) for its customers. However, it might be difficult for them to implement and host robust coordination services. Even if some of them do offer coordination services, other web services providers (such as those for the

- *H. Chai, H. Zhang, and W. Zhao are with the Department of Electrical and Computer Engineering, Cleveland State University, 2121 Euclid Ave. SH332, Cleveland, OH 44115. E-mail: chai.hua2008@gmail.com, honglei_zhang2000@hotmail.com, wenbing@ieee.org.*
- *P.M. Melliar-Smith and L.E. Moser are with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, Santa Barbara, CA 93106. E-mail: {moser, pmms}@ece.ucsb.edu.*

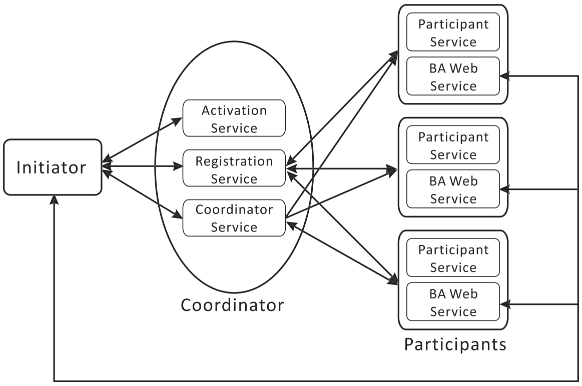

Fig. 1. WS-BA components.

airline reservation and hotel reservation web services) might not trust them.

- On the other hand, a large company, such as Amazon or Google, might offer competing coordination services as part of their cloud services. To avoid vendor lockin, customers of such coordination services would naturally need a standard interface, to ensure interoperability.

To address the interoperability issue, Erven et al. [12] proposed an extension to the WS-BA specification, referred to as the web services-business activity-initiator (WS-BA-I) protocol. The WS-BA-I protocol separates the coordination functionality and the business logic, and standardizes the interactions between the initiator and the coordinator. With the WS-BA-I extension to WS-BA, it becomes possible for a third party to offer coordination services to enterprises that want to conduct WS-BA with minimum modifications to their workflow engines. However, for such coordination services to be widely adopted, they must be trustworthy for their users.

Several groups of researchers have investigated the issue of trust in the Internet and the Web; useful summaries of that research are provided in [16] and [17]. Our use of the term *trustworthiness* in this paper is closest to that of Grandison and Sloman [17], who define trust as "the firm belief in the competence of an entity to act dependably, securely, and reliably within a specified context." In this paper, we are particularly concerned with high availability and high integrity of the WS-BA coordination services.

The web services community has developed a number of specifications toward increasing the trustworthiness of web services, namely,

- *WS-ReliableMessaging [8] and WS-Reliability [18].* These two specifications focus on reliable message exchange between two web services, despite transient transport-level and process failures, by defining application-level acknowledgments and sequence numbers. These two specifications are generally regarded as competing standards, but it appears that WS-ReliableMessaging has more widespread support.
- *WS-Security [28].* This specification defines the basic mechanisms for providing secure messaging between different web services.

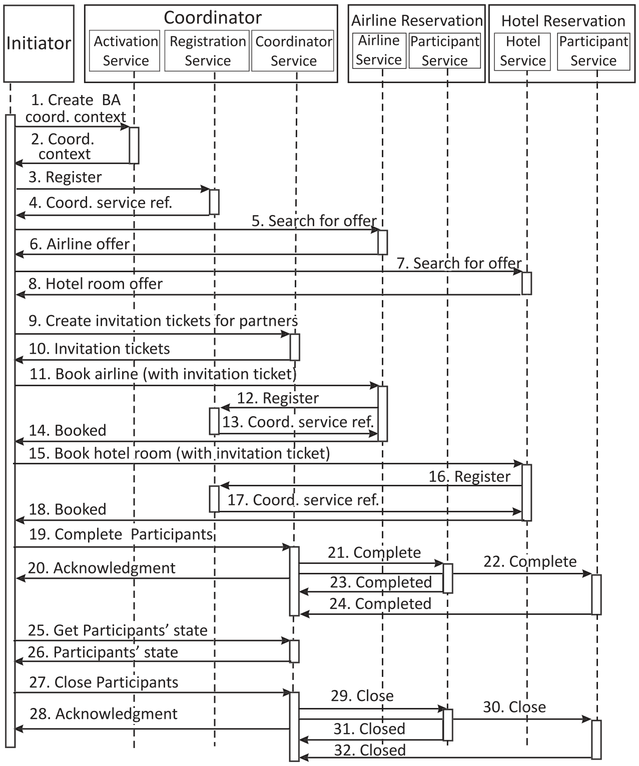

Fig. 2. A sequence diagram showing the steps of the travel reservation application using the WS-BA standard with the WS-BA-I extension.

- *WS-Trust [29].* This specification defines primitives and extensions for security token exchange so that web services of different trust domains can create and disseminate credentials.

The objective of our research is to push further toward trustworthy web services in the scope of WS-BA coordination beyond what has been addressed by the above specifications. It focuses on the high-availability and high-integrity aspects of WS-BA coordination. To see why high availability and high integrity of WS-BA coordination are needed, consider the travel reservation example:

- If the coordination services are not highly available, the airline reservation and hotel reservation web services might not be able to register for the WS-BA. Moreover, the coordination services might not be able to complete and close the WS-BA (more details are provided in Section 2).
- If the integrity of the coordination services are compromised, the WS-BA might have inconsistent outcomes (and other undesirable results) for the airline reservation and hotel reservation web services, as explained in more detail in Section 3.

The solution we present in this paper is complementary to, rather than competing against, the above specifications. In fact, our solution relies on the use of WS-Security for secure messaging. Moreover, it is possible to integrate WS-ReliableMessaging and WS-Trust into our prototype, should that be desired.

To achieve high availability and high integrity, we replicate the coordination services of the business activity. Due to the untrusted operating environment of the

Internet, it is necessary to employ a Byzantine fault model. A Byzantine fault is an arbitrary fault that might be a random fault caused by faulty hardware, or a malicious fault caused by an intrusion into the system. BFT refers to the ability of a system to tolerate Byzantine faults. BFT based on replication requires $3f + 1$ replicas, where at most f replicas are faulty [21]. BFT can be achieved by ensuring that all server replicas reach agreement on the inputs despite Byzantine faulty replicas and clients. Such agreement is referred to as Byzantine agreement. Our solution is based on BFT techniques.

In this paper, first we analyze the threats to the WS-BA coordination services, and explore strategies to mitigate such threats. Due to its high cost, we avoid the use of a traditional BFT algorithm designed for general-purpose stateful server replication. Instead, we present a lightweight BFT algorithm that exploits the state model defined in the WS-BA specification. Our lightweight BFT algorithm does not guarantee that messages are delivered in total order at nonfaulty replicas, as that is not needed in this case. Rather, it ensures that messages are delivered in source order (i.e., the order in which the sender sent them) at nonfaulty replicas, despite the presence of Byzantine faulty replicas and clients. We have implemented our lightweight BFT algorithm and associated mechanisms, and have incorporated them into the open-source Kandula framework [2] that implements the WS-BA standard with the WS-BA-I extension. Our performance evaluation of the prototype implementation confirms the efficiency and effectiveness of the lightweight BFT algorithm and mechanisms, compared to traditional BFT techniques.

2 Web Services Business Activities

In this section, we briefly describe the WS-BA specification and the WS-BA-I extension with a focus on normal operation. We also present more details on the travel reservation application that we introduced previously and refer to throughout the paper.

2.1 WS-BA Specification

The WS-BA specification [14] describes how to coordinate long running business activities where the atomic transaction model is not appropriate. It defines a participant-side service and a set of coordinator-side services. The coordinator-side services include activation, registration, and coordinator services, which run in the same address space of the coordinator.

The initiator of a business activity requests the activation service to generate a coordination context for the business activity and to create a coordinator object for the business activity. The coordinator object provides the registration and coordinator services, within the scope of the business activity. When the business activity is propagated to it, a web service registers with the registration service by providing an endpoint reference to its participant service. The registration service replies with an endpoint reference to the coordinator service. The coordinator service interacts with the participants via the business-agreement-with-coordinator-completion (BAwCC) or business-agreement-with-participant-completion (BAwPC) protocol (discussed below), and it interacts with the initiator via the WS-BA-I protocol (discussed in Section 2.2). For convenience, we refer to these services collectively as the coordinator.

WS-BA is built on top of the WS-coordination framework [13]. It specifies two coordination types: atomic-outcome and mixed-outcome. Moreover, it specifies two coordination protocols that operate between the coordinator and a participant: BAwPC and BAwCC. Either coordination type can be used with either coordination protocol.

If the atomic-outcome coordination type is used, all participants must reach agreement on the outcome of the business activity (i.e., close or compensate). If the mixed-outcome coordination type is used, some participants are directed to close while others are directed to compensate.

A participant registers one of the two coordination protocols (BAwPC or BAwCC) with the coordinator of the business activity. We briefly describe the two coordination protocols below. For detailed state transition diagrams of the coordination protocols, please see the WS-BA specification [14].

For the BAwPC protocol, when a participant has finished its work for a business activity, the participant informs the coordinator by sending a completed message. The coordinator replies with either a close message or a compensate message, depending on the circumstances. If the business activity has completed successfully, the participant receives a close message. Otherwise, the participant receives a compensate message, and it must undo the completed work and restore the data it recorded at the outset of the business activity.

A participant might encounter a problem or fail during the processing of the business activity. If an error occurred when the participant was trying to complete its normal activity, it generates a fail message and sends that message to the coordinator, possibly on recovery from a transient fault. Similarly, if an error occurred when the participant was trying to cancel or compensate an activity, it generates a cancelcomplete message and sends that message to the coordinator.

For the BAwCC protocol, the completion notification comes from the coordinator. The coordinator sends a complete message to the participants, informing them that they will not receive any new requests within the current business activity and that they should complete their processing. If a participant has successfully finished its work, it replies by sending a completed message. Other interactions between the coordinator and the participants are similar to those of the BAwPC protocol.

2.2 WS-BA-I Extension

The WS-BA-I protocol [12] is an extension of the WS-BA specification. WS-BA-I 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 [22]. To facilitate the WS-BA-I protocol, the coordinator exposes a number of additional operations for the initiator, including one to query the state of the business activity and one to create invitation tickets. (An invitation ticket is used by the initiator to propagate the business activity to a remote web service. The remote web service then registers with the coordinator using the invitation ticket.) In addition, they include operations that enable the initiator to pass instructions to the coordinator to complete (for the BAwCC protocol), close, cancel, or compensate the business activity.

The WS-BA-I protocol uses the pull model on the initiator side, so that the initiator can operate behind a firewall or a NAT box. To obtain the latest state of a business activity, the initiator periodically polls the coordinator.

2.3 Travel Reservation Example

Fig. 2 shows the normal execution steps of the travel reservation application using the WS-BA specification and the WS-BA-I extension, adapted from [2]. In this example, we use the atomic-outcome coordination type and the BAwCC protocol. The travel reservation application includes the airline reservation and hotel reservation web services. The airline reservation web service comprises an airline service and a participant service, and the hotel reservation service comprises a hotel service and a participant service.

The initiator creates a business activity coordination context using the activation service (1 and 2), and then registers with the registration service (3 and 4). Next, the initiator searches the airline reservation web service for an offer, from which it receives a reply (5 and 6), and then it searches the hotel reservation web service for an offer, from which it receives a reply (7 and 8). Next, it invokes the coordinator service to create tickets for the participants (9 and 10). Using the invitation tickets, it books an airline reservation (11 and 14), and a hotel room (15 and 18). On receiving a reservation request from the initiator, the airline reservation web service registers with the registration service (12 and 13); similarly, the hotel reservation web service registers with the registration service (16 and 17).

The initiator sends a CompleteParticipants message to the coordinator service (19). The coordinator sends an acknowledgment to the initiator (20) as soon as it has sent a complete message to the participant services of the airline reservation and hotel reservation web services (21 and 22) without waiting to receive their completed messages (23 and 24). As such, to know the participants' state, the initiator must query the coordinator service (25 and 26). When all of the participants are in the correct state, the initiator sends a CloseParticipants message (27) to the coordinator service. Similar to the handling of the CompleteParticipants request, the coordinator sends an acknowledgment (28) immediately after it has transmitted a close message to the participant services of the airline reservation and hotel reservation web services (29 and 30), without waiting to receive their completed messages (31 and 32).

3 Threat Analysis

In this section, we analyze threats that can compromise the integrity of the coordination services of WS-BA. We do not consider general threats, such as distributed denial of service attacks, that target any online service. Moreover, we do not consider entities that refuse to participate in the protocols, hoping to reduce the availability of the coordination services.

3.1 Threats from a Faulty Coordinator

First, we consider threats from a faulty coordinator to the activation service, the registration service and the coordinator service.

Threats to the activation service. At the start of a 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 to handle the business activity. The coordination context is included in all requests sent in the scope of the business activity. The coordination context contains two main components: a coordination identifier and an endpoint reference for the registration service. The coordination identifier, which must be unique to the business activity, is used to associate the participants in the same business activity. The endpoint reference for the registration service is used by a web service to register its participant service.

A faulty coordinator could:

1. reuse an old coordination identifier, which could potentially lead to a replay attack, i.e., an adversary could register as a participant for a business activity to which it does not belong, and subsequently replay other messages in the scope of the business activity;
2. use a coordination identifier that can be easily predicted; or
3. use a coordination identifier that belongs to a different business activity (which is equivalent to reusing the coordinator object created for another business activity).

Case 1 can be easily mitigated by transport-level security mechanisms such as the use of a 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. Doing so does not pose a real threat to the business activity, because the outcome of the business activity is determined by the initiator. The initiator can spot and subsequently exclude the unsolicited adversary from the business activity. Case 3 could lead to serious consequences if it is not mitigated, because two unrelated business activities would appear to be the same business activity, which might confuse the initiator and the participants. The lightweight BFT algorithm described in Section 4.3 mitigates such threats. Note that it is quite common for multiple participants to register for the same business activity (e.g., the airline reservation and hotel reservation web services register themselves as participants for the same business activity) and, hence, the registration service cannot detect the reuse of the coordination identifier without additional mechanisms.

Threats to the registration service. A web service registers with the registration service by providing an endpoint reference to its participant service, as soon as the business activity is propagated to it. The registration request must contain a valid coordination identifier, with a matchcode (introduced in WS-BA-I) to authenticate the participant. The reply to the registration request contains an endpoint reference to the coordinator service.

A faulty coordinator could:

1. accept a registration request with an illegitimate credential. For example, a faulty coordinator could

collude with a nonreputable hotel reservation company and take over the business from a reputable hotel reservation company with which the travel reservation company has contracted;

2. accept a correct registration request but assign the participant to a faulty coordinator. For example, a faulty coordinator could ensure that the airline reservation or hotel reservation web service remains under its control (or another faulty coordinator's control) regarding the termination of the business activity; or
3. return an invalid endpoint reference. For example, a faulty coordinator could return an invalid endpoint reference to the airline reservation or hotel reservation web service to deny its participation in the business activity.

These threats cannot be easily mitigated using transport-level security mechanisms. BFT replication and, in particular, the lightweight BFT algorithm described in Section 4.3 is the most appropriate form of control.

Threats to the coordinator service. The coordinator service interacts with the participants via the BAwCC or BAwPC protocol, and it interacts with the initiator via the WS-BA-I protocol. On the coordinator side, the BAwCC protocol notifies the participants about the completion of a business activity, acknowledges the reports sent by them, and informs them of the final decisions regarding the tasks that they completed. The coordinator also receives notifications from the participants.

A faulty coordinator could:

1. send a notification, such as a complete, close, or compensate message, to a participant, without the authorization of the initiator;
2. ignore reports from the participants, in particular, the fail and CannotComplete messages; or
3. make arbitrary state transitions without receiving reports from the participants.

Case 1 can be addressed by requiring all notifications (including the original signed authorization for the action from the initiator) to carry a security certificate [33]. Case 2 or 3 would lead to a state at the coordinator that is inconsistent with that of the participants, which might ultimately affect the initiator's decision on the outcome of the business activity. The lightweight BFT algorithm, described in Section 4.3, can effectively handle such threats.

The BAwPC protocol is similar to the BAwCC protocol, except that the coordinator is no longer responsible for notifying the participants of the completion of the business activity. Instead, the coordinator collects completed messages from the participants. Consequently, the threats that a faulty coordinator poses to the coordinator service in the BAwPC protocol are similar to those in the BAwCC protocol, and can be handled in a similar manner.

In the WS-BA-I protocol, the coordinator creates an invitation ticket on request from the initiator. The invitation ticket is an extended coordination context, containing an additional identifier (called the matchcode) used to invite the participant 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 the business activity.

A faulty coordinator could:

1. return an invalid invitation ticket to the initiator, such as an old invitation ticket, an invitation ticket that belongs to another participant, or an invitation ticket that is easy-to-predict;
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 the business activity.

The possible replay attack caused by Case 1 can be mitigated by transport-level security mechanisms, as mentioned previously. For the other threats in Cases 1-3, the lightweight BFT algorithm, described in Section 4.3, is an effective control.

3.2 Threats from a Faulty Participant

Now, we consider threats from a faulty participant to the coordinator and the initiator.

Threats to the coordinator. A faulty participant could:

1. lie about its execution status and send reports, that are inconsistent with its internal state, to the coordinator; or
2. send conflicting reports to different coordinator replicas.

Case 1 can be prevented by replicating each participant using BFT replication techniques and by collecting matching requests from $f + 1$ distinct replicas before accepting them (assuming at most f replicas are faulty). However, imposing such a strong requirement on the participants is not practical. Therefore, this type of threat is best addressed by using digital signatures on the messages exchanged between the participants and the coordinator, and by logging messages to/from the participants at the coordinator. Case 2 can be addressed by the lightweight BFT algorithm described in Section 4.3.

Threats to the initiator. A faulty participant could attack the initiator in a similar way to the attacks on the coordinator, as described above.

Again, these types of threats are best addressed by using digital signatures on messages exchanged between the participants and the initiator, and by the initiator's logging messages to/from the participants. The initiator might also give preference to participants that offer a higher quality of service to minimize such threats.

3.3 Threats from a Faulty Initiator

Because a business activity is initiated, and its progress and outcome are controlled, by the initiator, the integrity of a business activity that originates at a Byzantine faulty initiator cannot be guaranteed. As in the case of a faulty participant, we can address these threats by replicating the initiator and employing BFT techniques. However, based on similar concerns as above, we do not wish to impose such a strong requirement on the initiator. Instead, a more practical approach is for the participants and the coordinator to employ non-repudiation and logging techniques to hold a faulty initiator accountable.

With a replicated coordinator, a new threat might occur, i.e., a faulty initiator might send conflicting requests to different coordinator replicas. Such a threat can be readily addressed by the lightweight BFT algorithm described in Section 4.3.

4 BFT Coordination Services

In this section, we present the lightweight BFT algorithm and associated mechanisms for achieving trustworthy coordination of WS-BA. An important objective is to enable an independent third party to launch trustworthy coordination services for WS-BA that span multiple enterprises. The practicality of the BFT solution is essential for real-world use, which means that it must be lightweight with moderate runtime overhead and good scalability.

Traditional BFT state-machine replication mechanisms [5] can be used to mitigate the threats to the WS-BA coordination services. However, it is not wise to use them naively. Such mechanisms are designed to protect *general-purpose stateful* servers from Byzantine faults. They impose a total order on incoming requests and, thus, are very heavyweight and incur significant runtime overhead, due to the multiple rounds of message exchange that are required.

In our lightweight BFT algorithm, we exploit knowledge of the state model of the WS-BA coordination services, described below, and use a solution that is customized to this specific application. In short, the lightweight BFT algorithm requires only source ordering, instead of total ordering, of incoming requests, which eliminates inter-replica message exchange and hence significantly reduces the communication steps and processing overhead, compared with traditional BFT algorithms.

4.1 State Model Analysis

In WS-BA, requests within *different* business activities are handled independently, i.e., their relative ordering does not affect the state transitions. This independence is obvious for the registration and coordinator requests, because they are handled by different coordinator objects. Even though all activation requests are handled by the same process, their relative ordering does not affect how the coordinator objects are created. However, the generation of coordination identifiers can be a concern for replica consistency. We handle this particular replica nondeterminism issue without resorting to inter-replica coordination (as described in Section 4.4), which obviates the need to run an expensive Byzantine agreement algorithm among the coordinator replicas.

Next, we consider requests within the *same* business activity. The activation and registration requests are causally related, i.e., the activation request must precede the registration request. This relative ordering of requests is programmed directly into the BFT framework without resorting to inter-replica coordination. It is straightforward to ensure source ordering of requests from a participant in the BAwCC or BAwPC protocol (e.g., by using a sequence number). Likewise, it is straightforward to ensure source ordering of requests from the initiator.

The change of one part of the coordinator state associated with one participant has no direct effect on another part of the coordinator state associated with a different participant, as shown in Fig. 3. The coordinator object uses each part to keep track of the state for each participant according to the BAwCC or BAwPC protocol. Requests sent by different participants to the same coordinator object modify only their respective parts of the state. (Duplicate requests from the same participant do not change the coordinator state, although they might trigger the resending of messages.) Therefore, it is unnecessary to order requests from different participants.

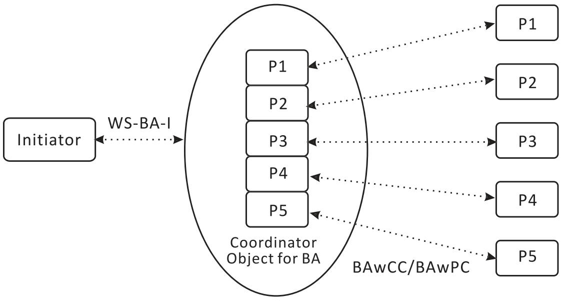

Fig. 3. The state model for WS-BA with the WS-BA-I extension. The participants' states in the coordinator object are completely separate.

The only remaining issue is whether or not to order the requests from the initiator relative to those from the participants within 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 specification, as discussed below.

The first type of request creates invitation tickets for the participants, one at a time. This type of request can be interleaved with the requests from the participants within the business activity, because they are handled by different objects.

The second type of request queries 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 states to the initiator if the query requests are not ordered with respect to the participants' requests. This difference in the states is not a concern because the initiator can repeatedly query the coordinator replicas until their states converge, as guaranteed by the lightweight BFT algorithm (see the proof of Theorem 3 in Section 4.5).

The third type of request directs the coordinator to terminate (close, cancel, or compensate) the business activity. If these requests are not ordered with respect to the messages from the participants, when such a request arrives, some replicas might have evolved into different states since the initiator last queried them. This difference in states is not a concern because, according to the WS-BA-I protocol, 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 message to the coordinator for a participant when the last seen state for that participant is completing. However, when the cancel message is delivered, the completed message from the participant might have arrived, in which case the coordinator should send a compensate message to the participant instead of a cancel message. This mechanism is already built into the standard WS-BA coordination framework.

Thus, there is no need to ensure total ordering of the messages involved in a business activity at the coordinator replicas.

4.2 Assumptions and Requirements

We assume that the WS-BA operate in the asynchronous distributed environment of the Internet. We assume that network communication is reliable. In particular, if a nonfaulty participant/initiator sends a message to a nonfaulty coordinator replica, the replica will eventually receive the message. The same is true for messages exchanged between nonfaulty coordinator replicas. This reliable communication assumption is easily satisfied by using TCP communication and the mechanisms defined in the WS-ReliableMessaging specification [8].

We assume that there are $3f+1$ coordinator replicas, of which at most f are faulty. Each coordinator replica is assigned a unique identifier k, where k ranges from 0 to $3f$. All of the coordinator replicas play an equal role; in particular, no coordinator replica acts as the primary because total ordering is not needed (i.e., the sequence numbers are assigned by the client, rather than by a primary coordinator replica). The initiator and the participants are not replicated (our BFT algorithm can be trivially modified if these entities are in fact replicated). As pointed out earlier, only a subset of faulty behaviors of the initiator and the participants are tolerated by replicating the coordinator. Other types of faults are handled by using non-repudiation and logging techniques.

The high degree of redundancy (four coordinator replicas to tolerate one faulty coordinator replica) required to achieve BFT [21] might be a concern for practitioners. However, with the widespread adoption of virtualization technology, the need for extra physical hardware is minimized because the four coordinator replicas can be deployed on different physical nodes (for fault isolation), which are running other services concurrently.

All WS-BA messages, i.e., the messages exchanged between the coordinator and the initiator and between the coordinator and the participants, carry monotonically increasing sequence numbers. The sequence number is unique only within the respective connection between the coordinator and its client (i.e., the initiator or a participant), but each connection is uniquely identified. For each connection, the first request is assigned sequence number 0 and, for each subsequent request, the sequence number is incremented. The reply, if any, carries a sequence number that matches that of the corresponding request. The same holds true for requests sent from the coordinator to the participants in the BAwCC or BAwPC protocol. Note that all messages defined in the BAwCC or BAwPC protocol are one-way messages.

All messages between the coordinator and the participants are timestamped (to prevent replay attacks) and are digitally signed (to ensure accountability and to prevent spoofing). The initiator, each coordinator replica and each participant has a public/private key pair. The public key is known to all of them, whereas the private key is kept secret by its owner. We assume that the adversaries have limited computing power and, thus, are unable to break the digital signatures.

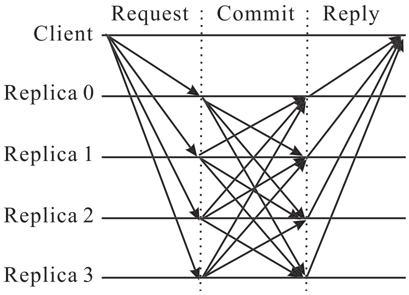

Fig. 4. Normal operation of the lightweight BFT algorithm for $f=1$.

Our lightweight BFT algorithm for trustworthy coordination of WS-BA, described below, satisfies the following three properties:

P1. If a Byzantine faulty sender sends conflicting requests (different requests with the same sequence number) to different nonfaulty coordinator replicas, at most one of those requests is delivered at all nonfaulty coordinator replicas.

P2. If a request is delivered at a nonfaulty coordinator replica, it is eventually delivered at all nonfaulty coordinator replicas in the sender's source order.

P3. The states of the nonfaulty coordinator replicas will eventually converge, and the initiator will eventually receive a response to a query regarding the state of the business activity, that is consistent with the converged state of the coordinator replicas.

The proofs of correctness, demonstrating that the lightweight BFT algorithm satisfies these properties, are given in Section 4.5.

4.3 The Lightweight BFT Algorithm

Based on the previous state model analysis, we have developed the lightweight BFT algorithm described below. The normal operation of the algorithm is shown in Fig. 4 for $f=1$.

A client (i.e., the initiator or a participant) sends a request to all coordinator replicas. The request has the form $<\mathrm{REQUEST}, s, o, c>_{\sigma_c}$, where s is the sequence number of the request message, o is the operation to be invoked (including the coordination context, if needed), c is the client identifier, and σ_c is the digital signature of the request message signed by the client c.

On receiving a request, a coordinator replica validates the signature, and checks the validity of the requested operation and whether the sequence number in the request matches its next expected sequence number. If the request passes this validation test, a coordinator replica multicasts a commit message to all of the other coordinator replicas. The commit message has the form $<\mathrm{COMMIT}, s, d, k>_{\sigma_k}$, where s is the sequence number of the request message, d is the digest of the request message, k is the replica identifier, and σ_k is the signature of the commit message signed by the replica k.

If a coordinator replica receives a request and matching commit messages corresponding to that request from $2f$ other coordinator replicas, it delivers the request and makes the state transition. The reply, if any, has the form $<\mathrm{REPLY}, s, r, k>_{\sigma_k}$, where s is the sequence number of the reply message, r is the response, k is the replica identifier, and σ_k is the signature of the reply message signed by the replica k.

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

If a coordinator replica receives a request and matching commit messages from $f+1$ distinct coordinator replicas, but the digest in a commit message does not correspond to the request, it requests a retransmission of the request from the $f+1$ distinct coordinator replicas, logs the event that the digest in the commit message does not correspond to the original request, and abandons the original request.

If the client (i.e., the initiator or a participant) that issued a request expects a reply from the coordinator replicas, it collects matching replies from $f+1$ distinct coordinator replicas before it delivers the reply. The same mechanism is used for the participants to handle (nested) requests sent by the coordinator replicas. Because at most f coordinator replicas are faulty and all of the replies match, at least one of the replies is sent by a nonfaulty coordinator replica.

Note that some operations might trigger the sending of nested requests to the participants (e.g., the complete/close message from the initiator to a participant will cause a nonfaulty coordinator replica to send the same command to that participant). Such nested requests are sent directly to the participants. A nonfaulty participant accepts a nested request when it has collected matching requests from $f+1$ distinct coordinator replicas. However, the nested reply must be treated as a new request, i.e., an additional communication step is needed before the nested reply is delivered at the coordinator replica because the nested reply might trigger a state change.

4.4 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). When the activation service is replicated, generating the coordination identifier in this way causes replica nondeterminism. Even though such nondeterminism could be controlled by using the mechanism described in [32], it is unnecessary in light of the threat analysis given in Section 3. A more efficient method to handle this nondeterminism is to piggyback the UUID onto the activation request sent by the initiator. One implication of this strategy is that the coordinator replicas must now keep a record of the UUIDs used and verify the uniqueness of a newly proposed UUID against that record.

Another mechanism used by our lightweight BFT solution is the security certificate mentioned in Section 3. A security certificate is piggybacked onto each message sent by the coordinator to a participant in the BAwCC or BAwPC protocol. The security certificate consists of the original signed authorization for the action from the initiator, which includes not only the specific command but also the coordination identifier and timestamp to prevent a replay attack. On receiving such a command, a participant verifies that the command is consistent with that in the security certificate. If a participant detects a mismatch, it ignores the command.

4.5 Proofs of Correctness

Now we provide (informal) proofs of correctness for the lightweight BFT algorithm in terms of the three properties given in Section 4.2. The proofs are based on source ordering of requests, and address the threats identified in Section 3.

Theorem 1. *If a Byzantine faulty sender sends conflicting requests (different requests with the same sequence number) to different nonfaulty coordinator replicas, at most one of those requests is delivered at all nonfaulty coordinator replicas.*

Proof. The proof is by contradiction. Assume that a request message M is delivered at a nonfaulty coordinator replica R and that a different request message M' with the same sequence number from the same sender is delivered at another nonfaulty coordinator replica R'. It must be the case that a set S containing $2f+1$ coordinator replicas have accepted M and sent a commit message for M and that a set S' containing $2f+1$ coordinator replicas have accepted M' and sent a commit message for M'. Because there are $3f+1$ coordinator replicas, S and S' must intersect in $f+1$ or more coordinator replicas. Because at most f coordinator replicas are faulty, at least one coordinator replica in the intersection must be nonfaulty. This is a contradiction, because our BFT algorithm does not allow a nonfaulty coordinator replica to accept two different requests with the same sequence number from the same sender. □

Theorem 2. *If a request is delivered at a nonfaulty coordinator replica, it is eventually delivered at all nonfaulty coordinator replicas in the sender's source order.*

Proof. First, we assume that the sender of the request is *not* Byzantine faulty. By the assumption of reliable communication in Section 4.2, the sender establishes a connection with each nonfaulty coordinator replica and reliably sends the request in source order to that nonfaulty coordinator replica. Consequently, nonfaulty coordinator replicas receive the request, exchange commit messages, and deliver the request. □

If the sender is Byzantine faulty, it might 1) send the same request to only some of the nonfaulty coordinator replicas, or 2) send conflicting requests (different requests with the same sequence number) to different nonfaulty coordinator replicas.

In Case 1, by the hypothesis of the theorem, a request is delivered at a nonfaulty coordinator replica R. That replica R must have received commit messages for the request from $2f$ other coordinator replicas, of which at most f are nonfaulty. If a nonfaulty coordinator replica R' did not receive the request, it will eventually receive the commit messages from $f+1$ other nonfaulty coordinator replicas (including replica R), which will prompt it to request a retransmission of the request. By the reliable communication assumption, eventually R' will receive the request from one of those nonfaulty coordinator replicas. Consequently, all nonfaulty coordinator replicas will eventually receive the request, exchange commit messages, and deliver the request.

In Case 2, by the hypothesis of the theorem, a request is delivered at a nonfaulty coordinator replica R. That replica R must have received commit messages for the *same* request from $2f$ other coordinator replicas, of which at most f are nonfaulty. If a nonfaulty coordinator replica R' receives a request that conflicts with matching commit messages that it received from $f+1$ coordinator replicas, it requests retransmission of the request. At least one of those coordinator replicas must be nonfaulty and will retransmit the request. By the reliable communication assumption, eventually R' will receive the request from one of those nonfaulty coordinator replicas. Consequently, all nonfaulty coordinator replicas will eventually receive the request, exchange commit messages, and deliver the request.

Theorem 3. *The states of the nonfaulty coordinator replicas will eventually converge, and the initiator will eventually receive a response to a query regarding the state of the business activity, that is consistent with the converged state of the coordinator replicas.*

Proof. By Theorem 2, if a request is delivered at a nonfaulty coordinator replica, it is eventually delivered at all nonfaulty coordinator replicas. Therefore, all nonfaulty coordinator replicas will eventually deliver the same set of requests, at which point the states of the nonfaulty coordinator replicas will converge. □

Our BFT algorithm requires the initiator to collect matching responses for a query from $f+1$ distinct coordinator replicas before accepting the response. At least one of those matching responses is sent by a nonfaulty coordinator replica. Consequently, the initiator will eventually receive a response to a query regarding the state of the business activity, that is consistent with the converged state of the coordinator replicas.

5 Implementation and Performance

We have implemented the lightweight BFT algorithm and 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. Besides Kandula, the extended framework is based on several other Apache web services technologies, including Apache Axis [1] and WSS4J [3], which is an implementation of the WS-Security standard [28]. Most of the mechanisms are implemented using Axis handlers that are plugged into the framework without affecting other components. Some of the Kandula code is modified to enable control of its internal state and BFT delivery of requests at the initiator, the coordinator, and the participants. All messages have timestamped digital signatures.

To demonstrate the benefits of our lightweight BFT algorithm over traditional BFT algorithms, we implemented an adapted version of the practical Byzantine fault tolerance (PBFT) algorithm [5] for comparison. In this implementation, we chose to use digital signatures for all messages exchanged (instead of the message authentication code). We launched one instance of the PBFT algorithm for each coordinator, i.e., we allow concurrent total ordering of messages belonging to different business activities for fair comparison with our lightweight BFT algorithm. Otherwise, the throughput for concurrent business activities would be very low for PBFT, because it would not enjoy concurrent processing as our lightweight BFT algorithm does.

We have evaluated the performance of our lightweight BFT algorithm both in a local-area network (LAN) testbed and in a wide-area network (WAN) testbed (i.e., PlanetLab). The LAN testbed consists of 14-HP BL460c blade servers connected by a Cisco 3020 Gigabit switch. Each blade server is equipped with two Xeon E5405 (2 GHz) processors and 5-GB RAM, and runs the 64-bit Ubuntu Linux operating system. The hardware specifications of the PlanetLab nodes vary significantly. The nodes we chose to use are generally equipped with Intel Core 2 Duo CPUs (2 to 2.6 GHz). Note that the nodes in PlanetLab are shared among many users, and we have no control over the actual load on the CPUs and the available physical memory.

The test application is the travel reservation application described in Sections 1 and 2.3, and shown in Fig. 2, with a slight modification on how the coordinator handles CompleteParticipants and CloseParticipants requests from the initiator. Instead of sending a response for each request as soon as the nested request has been sent to all of the participants, a coordinator replica waits until it has collected the responses from all of the participants. That is, in Fig. 2, message 20 is sent after the coordinator has received message 24, and message 28 is sent after the coordinator has received message 32. This modification (that the initiator does not need to query the coordinator repeatedly to see whether all of the participants have responded) is purely for the sake of benchmarking, so that the benchmarking results are more consistent across different runs. The coordinator is replicated on $3f+1=4$ processors to tolerate $f=1$ Byzantine faulty replica. The initiator and the participants each run on a distinct processor. The initiator launches and terminates business activities repeatedly within a loop without any think time.

The end-to-end latency of each business activity is measured at the initiator. The throughput of the coordination framework is measured at one of the coordinator replicas. In each run, we obtained 1,000 samples and calculated the median latency and the median throughput. We chose to use the median instead of the mean, because the results obtained in PlanetLab vary significantly across different runs.

5.1 Performance Evaluation in a LAN

The end-to-end latency results in a LAN are shown in Fig. 5. These results are obtained for four different configurations:

1. The test application is used as is without any modification (labeled "Unmodified Application" in the figure).
2. The test application is modified such that all messages exchanged within each business activity are protected by digital signatures (labeled "With Digital Signatures" in the figure).

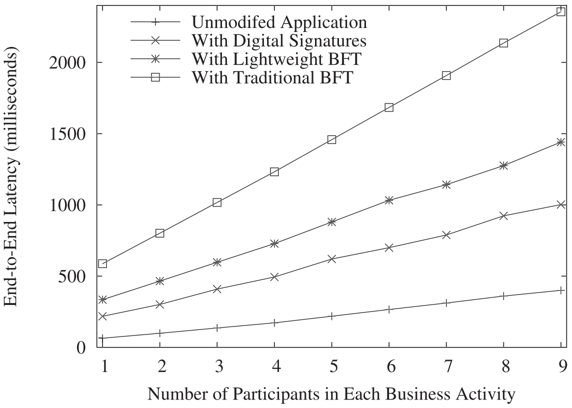

Fig. 5. End-to-end latency in a LAN for business activities with different numbers of participants under normal operation.

3. The test application is protected by our lightweight BFT algorithm with replication of the coordinator (labeled "With Lightweight BFT" in the figure).
4. The test application is protected by the traditional BFT algorithm with replication of the coordinator (labeled "With Traditional BFT" in the figure).

In all four configurations, the end-to-end latency is measured in the presence of a single business activity (i.e., neither the coordinator nor the various web services are shared with other business activities).

As shown in Fig. 5, the end-to-end latency of our lightweight BFT algorithm is significantly larger than that of the unmodified application. However, this increase is due mainly to the use of digital signatures, as revealed by the high end-to-end latency when only digital signatures are used. In our LAN testbed, the cost of digitally signing a message is about 5 ms, and the cost of verifying a digital signature is about 2 ms. In our test application, a single business activity with only one participant involves 22 digital signatures and 22 validation operations during the critical path (eight extra digital signatures and eight extra validation operations are needed for each additional participant). The large number of such operations inevitably increases the end-to-end latency. Nevertheless, it is our conviction that the use of digital signatures is justified (indeed essential) in practice because of the need for nonrepudiation in any serious business activity. Thus, we use this configuration as the baseline for comparison.

The overhead of our lightweight BFT algorithm is due primarily to two factors for each committed message: 1) one extra communication step, and 2) one extra digital signature (for the multicast message) and two validation operations (for the received message). For a business activity with a single participant, nine messages must be committed (three extra messages for each additional participant). These two factors result in a nonnegligible overhead, with about 40-50 percent larger end-to-end latency compared with that of the digital-signatures-only configuration.

As expected, the end-to-end latency with the traditional BFT algorithm is much larger, as shown in Fig. 5. Compared to our lightweight BFT algorithm, the traditional BFT algorithm incurs the following *additional* overhead for each message delivered at the coordinator during normal operation: 1) two communication steps, 2) two digital signatures, and 3) five validation operations. Therefore, the end-to-end latency with the traditional BFT algorithm is about 130-170 percent higher than that of the baseline configuration (more than three times the overhead compared to our lightweight BFT algorithm). Note that this experiment did not consider the cases when some replicas fail. In the traditional BFT algorithm, when the primary replica fails, one or more view changes will be needed, which could result in even larger and more unpredictable latency. Because our lightweight BFT algorithm does not depend on any one replica serving as the primary, the negative impact on the performance is negligible when a replica fails, similar to the impact when a backup replica fails in the traditional BFT algorithm.

The throughput results (in terms of number of business activities per minute) are shown in Figs. 6a, 6b, and 6c for business activities with 2, 5, and 9 participants in a LAN. The throughput experiments are carried out with the same set of configurations as those used in the end-to-end latency study, i.e., the unmodified application, the application with digital signatures, the application protected by our lightweight BFT algorithm, and the application protected by a traditional BFT algorithm. The load on the coordinator and the set of participants is driven by one or more initiators, launching business activities consecutively. As such, the throughput measurements are performed for various numbers of concurrent business activities.

As can be seen in Figs. 6a, 6b, and 6c, compared to the baseline configuration, the throughput reduction with our lightweight BFT algorithm is only about 30 percent. This

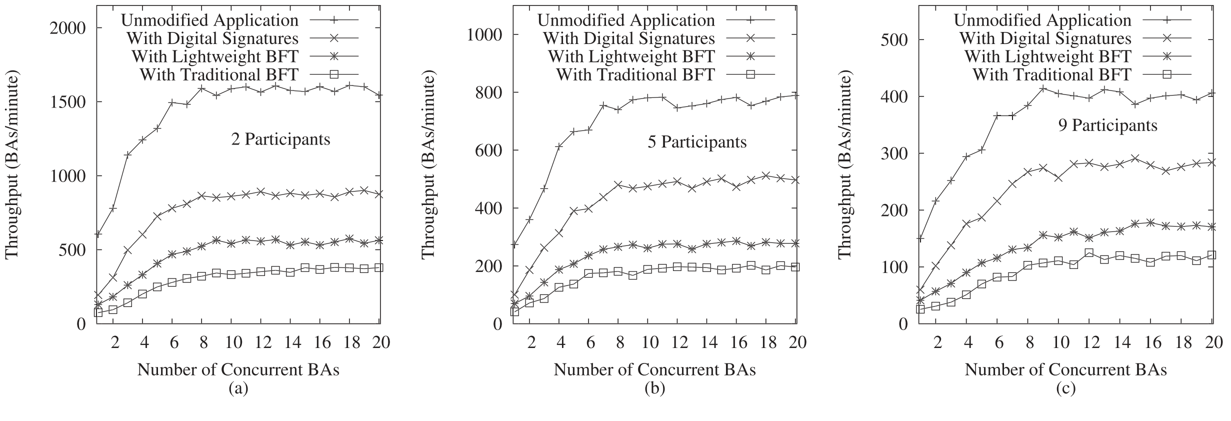

Fig. 6. Throughput of the coordination services in a LAN with different numbers of concurrent business activities.

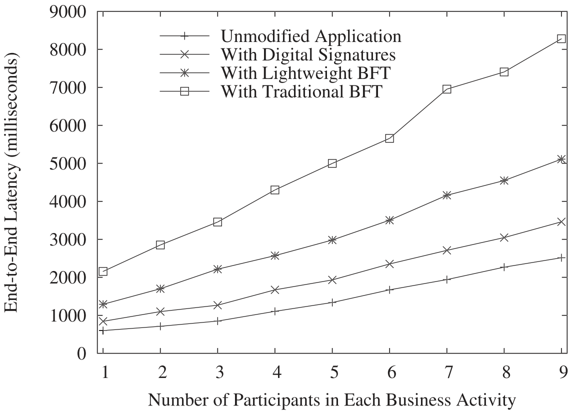

Fig. 7. End-to-end latency in a WAN for business activities with different numbers of participants under normal operation.

reduction is expected because of the increased processing time at the coordinator (for message signatures and validation). Again, not surprisingly, the throughput reduction with the traditional BFT algorithm is about 50 percent, much more significant than that for the lightweight BFT algorithm.

5.2 Performance Evaluation in a WAN

The end-to-end latency results in a WAN are summarized in Fig. 7. As can been seen in the figure, the relative overhead, over the baseline configuration, of the lightweight BFT algorithm remains similar to that in the LAN (about 40-50 percent), and the same is true for the configuration using the traditional BFT algorithm. However, it is interesting to note that, due to the large communication delay inevitable in the WAN, the relative cost of digital signatures and validation operations is reduced. Consequently, the overhead of our lightweight BFT algorithm, compared with that of the unmodified application, drops significantly (from about 250 percent to about 100 percent).

The throughput results in the WAN are shown in Fig. 8. The throughput for all configurations in the WAN is a lot less than that in the LAN primarily for three reasons:

1. In general, the nodes in PlanetLab for the WAN experiment are equipped with less capable CPUs compared to our LAN testbed (e.g., Core 2 Duo E6550 versus Xeon E5405). The passmark for the Core 2 Duo E6550 is 1,448, whereas the passmark for the Xeon E5405 is 2,895.[1]
2. The nodes in PlanetLab are shared among many concurrent users and, hence, our application might not be able to exploit the multiple cores available, while the nodes in our LAN testbed are exclusively used for our experiment.
3. The available network bandwidth between different nodes in PlanetLab is presumably a lot less than that in our LAN testbed, which is connected by a Gigabit Ethernet. As a result, the message transmission time is a lot more for the WAN experiment, which also adversely affects the throughput. For example, if the total size of messages sent and received by the coordinator is 30 KB and if the data rate in PlanetLab is 10 Mbps, then the transmission time in the PlanetLab testbed would be $24\,\mathrm{ms}$ whereas, in our LAN testbed, the transmission time would be a negligible 0.24 ms.

Due to the greater message transmission time, the throughput reduction for both the lightweight BFT algorithm and the traditional BFT algorithm is reduced compared to that of the unmodified application. These results suggest that it might be more appropriate to adopt the BFT approach in the WAN, where WS-BA are typically deployed, rather than in the LAN. In both the WAN and the LAN, the lightweight BFT algorithm significantly outperforms the traditional BFT algorithm, as demonstrated in our experiments.

6 Related Work

The trustworthiness of Internet and web applications has become increasingly important in recent years [16], [17], [29], particularly with the advent of WS-BA that involve interactions between multiple enterprises. The WS-Trust specification [29] focuses on the use of security tokens and credentials within the context of web services, whereas, in this paper, we focus on the high-availability and high-integrity aspects of trustworthy coordination of web service business activities.

BFT has been of great research interest for the last several decades. The seminal research in this field is that of Lamport et al. [21], who demonstrated that four replicas are required to tolerate one Byzantine faulty replica (three replicas do not suffice) and, more generally, that $3f+1$ replicas are required to tolerate f Byzantine faulty replicas. Other significant contributions to Byzantine agreement/consensus about the same time were made by Dolev et al. (see, e.g., [4], [7], [9], [10], [11]).

Subsequently, several groups of researchers developed Byzantine fault-tolerant group communication systems [6], [19], [23], [26], [27], [31] for generic distributed applications. All of those systems deliver messages in total order at the destinations. In contrast, our lightweight BFT algorithm for trustworthy coordination of WS-BA delivers messages in source order, i.e., the order in which the sender sent them, rather than in total order at the destinations. It does so by exploiting the state model of the WS-BA specification.

The PBFT agreement algorithm [5] of Castro and Liskov designates one of the replicas as the primary and the other replicas as the backups. The PBFT algorithm involves three phases to order a request (the pre-prepare phase, the prepare phase, and the commit phase). It also includes a view change algorithm that determines a new primary, if the current primary is deemed to be faulty. Kihlstrom et al. [20] described a BFT consensus algorithm. Their BFT algorithm for achieving consensus operates in three rounds, and uses a coordinator to choose the consensus value and Byzantine fault detectors to detect faults in the coordinator and other processes.

The application of BFT techniques to transactional systems was first reported by Mohan et al. [25]. They enhance the two-phase commit protocol by performing

[1] A passmark is the benchmark score of a computer using the software developed by Passmark Software. The passmark scores were obtained from http://www.cpubenchmark.net/.

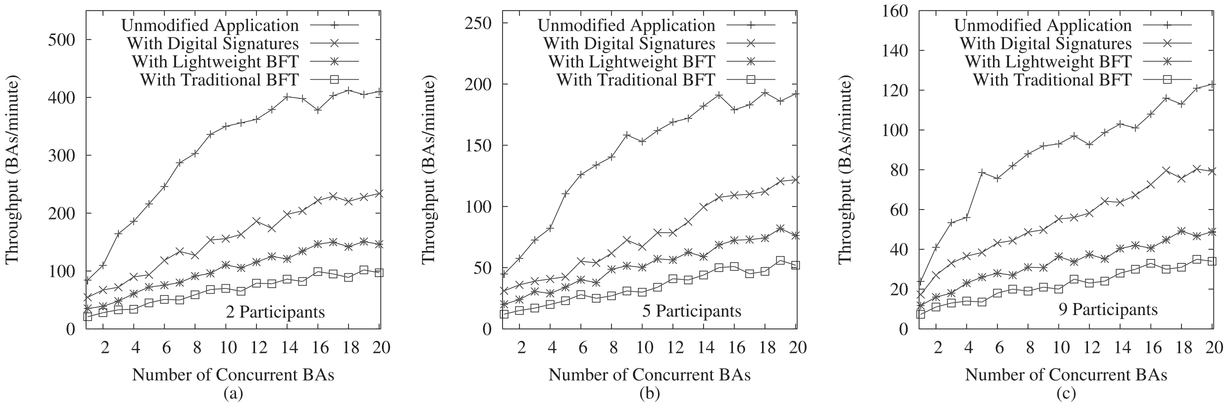

Fig. 8. Throughput of the coordination services in a WAN with different numbers of concurrent business activities.

Byzantine agreement on the outcome of an atomic transaction among all of the nodes in a root cluster. Recently, we revisited this problem and proposed a more efficient solution [33] for atomic transactions by restricting the Byzantine agreement to only the coordinator replicas. Garcia-Molina et al. [15] have applied Byzantine agreement to distributed database systems, in particular, for distributing transactions to processing nodes.

Even though coordination of atomic transactions bears some similarity to coordination of business activities, there are a number of significant differences. In atomic transactions, the coordinator and the participants are tightly coupled. Any participant can unilaterally abort a transaction. Furthermore, the coordinator can decide on the outcome of the transaction, based on the votes collected in the two-phase commit protocol. However, in business activities, the outcome is decided solely by the initiator according to the business logic and, at best, a faulty participant might be able to exert some influence on, but not decide on, the outcome of the business activity. These characteristics necessitate the use of a lightweight BFT solution for the trustworthy coordination of WS-BA, as described in this paper.

The application of BFT techniques to web services has been reported in [24], [30], and [34]. Even though the solutions proposed could be used to protect the coordination services of WS-BA against Byzantine faults, they are unnecessarily expensive. By considering the state model of the WS-BA coordination services, our lightweight BFT algorithm introduces only one additional round of message exchange for each invocation on the coordinator instead of two or more additional rounds of message exchange needed in [24], [30], and [34], and thus, it performs significantly better.

7 Conclusions

We have presented a comprehensive study of the threats to the coordination services of WS-BA. We have carefully analyzed the state model of the WS-BA coordination services, and have argued that it is unnecessary to perform total ordering of requests at the coordinator replicas, to achieve trustworthy coordination of WS-BA. Rather, it suffices to ensure that messages are delivered in source order, i.e., the order in which the sender sent them.

This analysis has enabled us to develop a lightweight BFT algorithm that mitigates the threats and avoids the runtime overhead associated with traditional BFT algorithms. We have implemented the lightweight BFT algorithm and associated mechanisms, and have incorporated them into the open-source Kandula framework that implements the WS-BA specification and the WS-BA-I extension. The performance evaluation results obtained using the prototype implementation show moderate overhead, especially in a wide-area network, where WS-BA are typically deployed.

The research that we have described in this paper fits in well with the recent trend of cloud computing. Many large companies, such as Amazon, Google, Microsoft, and Apple, are now offering cloud services. Some of them might provide coordination services for web services in the cloud. Such coordination services could enable smaller companies to offer composite web services to provide value-added services to their customers without having to invest heavily in computing infrastructures. The coordination service providers might find our lightweight BFT algorithm attractive because of its relatively low overhead. For the best fault isolation, it is desirable to deploy the coordinator replicas across geographically separated data centers, which are readily available for the large companies.

Acknowledgments

This work was supported in part by the US National Science Foundation (NSF) grants CNS 08-21319 and CNS 10-16193 and by a Cleveland State University Scholarship Initiative grant. An earlier version of this work was presented at the 2008 IEEE International Conference on Services Computing [35].

References

[1] Apache Axis, http://ws.apache.org/axis/, 2013.
[2] Apache Kandula, http://ws.apache.org/kandula/, 2013.
[3] Apache WSS4J, http://ws.apache.org/wss4j/, 2013.
[4] C. Attiya, D. Dolev, and J. Gil, “Asynchronous Byzantine Consensus,” *Proc. ACM Symp. Principles of Distributed Computing*, pp. 119-133, 1984.
[5] M. Castro and B. Liskov, “Practical Byzantine Fault Tolerance and Proactive Recovery,” *ACM Trans. Computer Systems*, vol. 20, no. 4, pp. 398-461, Nov. 2002.
[6] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riche, “UpRight Cluster Services,” *Proc. 22nd ACM Symp. Operating Systems Principles*, Oct. 2009.
[7] F. Cristian, H. Aghili, R. Strong, and D. Dolev, “Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement,” *Information and Computation*, vol. 118, no. 1, pp. 158-179, 1995.

[8] D. Davis, A. Karmarkar, G. Pilz, S. Winkler, and U. Yalcinalp, *Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.1,* OASIS Standard, Jan. 2008.

[9] D. Dolev, "The Byzantine Generals Strike Again," *J. Algorithms,* vol. 3, no. 1, pp. 14-30, 1982.

[10] D. Dolev, "An Efficient Algorithm for Byzantine Agreement without Authentication," *Information and Control,* vol. 52, no. 3, pp. 257-274, Mar. 1982.

[11] D. Dolev and H.R. Strong, "Authenticated Algorithms for Byzantine Agreement," *SIAM J. Computing,* vol. 12, pp. 656-666, 1983.

[12] 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," *Proc. IEEE Int'l Conf. Web Services,* pp. 216-224, July 2007.

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

[14] T. Freund and M. Little, *Web Services Business Activity, Version 1.1,* OASIS Standard, Apr. 2007.

[15] H. Garcia-Molina, F. Pittelli, and S. Davidson, "Applications of Byzantine Agreement in Database Systems," *ACM Trans. Database Systems,* vol. 11, no. 1, pp. 27-47, 1986.

[16] Y. Gil and D. Artz, "A Survey of Trust in Computer Science and the Semantic Web," *J. Web Semantics,* vol. 5, pp. 58-71, 2007.

[17] T. Grandison and M. Sloman, "A Survey of Trust in Internet Applications," *IEEE Comm. Survey Tutorials,* vol. 4, no. 4, pp. 2-16, Oct.-Dec. 2000.

[18] K. Iwasa, J. Durand, T. Rutt, M. Peel, S. Kunisetty, and D. Bunting, *WS-Reliability 1.1,* OASIS Standard, Nov. 2004.

[19] K.P. Kihlstrom, L.E. Moser, and P.M. Melliar-Smith, "The SecureRing Group Communication System," *ACM Trans. Information and System Security,* vol. 4, no. 4, pp. 371-406, Nov. 2001.

[20] K.P. Kihlstrom, L.E. Moser, and P.M. Melliar-Smith, "Byzantine Fault Detectors for Solving Consensus," *Computer J.,* vol. 46, no. 1, pp. 16-35, 2003.

[21] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Trans. Programming Languages and Systems,* vol. 4, no. 3, pp. 382-401, July 1982.

[22] M. Little and A. Wilkinson, *Web Services Atomic Transactions, Version 1.1,* OASIS Standard, Apr. 2007.

[23] D. Malkhi and M. Reiter, "Secure and Scalable Replication in Phalanx," *Proc. IEEE 17th Symp. Reliable Distributed Systems,* pp. 51-58, 1998.

[24] M. Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou, and P. Narasimhan, "Thema: Byzantine-Fault-Tolerant Middleware for Web Services Applications," *Proc. IEEE 24th Symp. Reliable Distributed Systems,* pp. 131-142, Oct. 2005.

[25] C. Mohan, R. Strong, and S. Finkelstein, "Method for Distributed Transaction Commit and Recovery Using Byzantine Agreement within Clusters of Processors," *Proc. ACM Symp. Principles of Distributed Computing,* pp. 89-103, 1983.

[26] L.E. Moser and P.M. Melliar-Smith, "Byzantine-Resistant Total Ordering Algorithms," *J. Information and Computation,* vol. 150, pp. 75-111, 1999.

[27] L.E. Moser, P.M. Melliar-Smith, and N. Narasimhan, "The SecureGroup Group Communication System," *Proc. IEEE Information Survivability Conf.,* vol. 2, pp. 256-279, Jan. 2000.

[28] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker, *Web Services Security: SOAP Message Security 1.1,* OASIS Standard, Nov. 2006.

[29] A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, and H. Grangvist, *WS-Trust 1.4,* OASIS Standard, Feb. 2009.

[30] S.L. Pallemulle, H.D. Thorvaldsson, and K.J. Goldman, "Byzantine Fault-Tolerant Web Services for N-Tier and Service Oriented Architectures," *Proc. IEEE 28th Int'l Conf. Distributed Computing Systems,* pp. 260-268, June 2008.

[31] M. Reiter, "The Rampart Toolkit for Building High-Integrity Services," *Proc. Int'l Conf. Theory and Practice in Distributed Systems,* pp. 99-110, 1995.

[32] W. Zhao, "Byzantine Fault Tolerance for Non-Deterministic Applications," *Proc. IEEE Int'l Symp. Dependable, Autonomous and Secure Computing,* pp. 108-115, Sept. 2007.

[33] W. Zhao, "A Byzantine Fault Tolerant Distributed Commit Protocol," *Proc. IEEE Int'l Symp. Dependable, Autonomous and Secure Computing,* pp. 37-44, Sept. 2007.

[34] W. Zhao, "BFT-WS: A Byzantine Fault Tolerant Framework for Web Services," *Proc. Middleware for Web Services Workshop,* Oct. 2007.

[35] W. Zhao and H. Zhang, "Byzantine Fault Tolerant Coordination for Web Services Business Activities," *Proc. IEEE Int'l Conf. Services Computing,* pp. 407-414, July 2008.