

BFT-WS: A Byzantine Fault Tolerance Framework for Web Services

Wenbing Zhao

Cleveland State University
wenbing@ieee.org



Outline

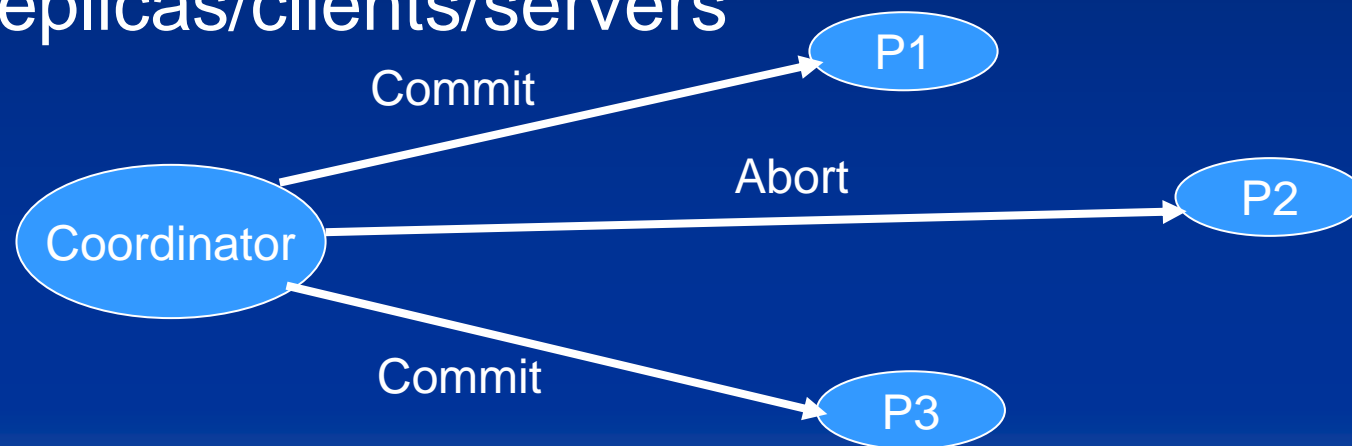
- Motivation
- Background
- BFT-WS Architecture
- Performance Evaluation
- Conclusion

Motivation

- Increasing reliance on services provided over the Web => Strong demand for robust and practical fault tolerance middleware
- Internet is untrusted computing environment => Byzantine fault (arbitrary fault) tolerance is needed to ensure maximum service dependability

Motivation

- Why Byzantine fault tolerance (BFT)?
 - To prevent a faulty server/replica from sending conflicting information to other replicas/clients/servers



Motivation

- Why developing a 100% Web Services (WS) BFT middleware framework?
 - To be consistent with the design principles of the WS platform: interoperability, extensibility, loose coupling, etc.
 - To be (backward) compatible with WS-* standards, in particular, WS-ReliableMessaging (WS-RM)
 - To be more efficient

Related Work

- Closest work is Thema (CMU) (also BFT for Web services)
 - Based on MIT's BFT framework => IP multicast and a proprietary messaging protocol
 - Had to use Java Native Interface
 - Incompatible with WS design principles and WS-* standards
 - It does offer richer functionality than BFT-WS, which we plan to offer in the near future

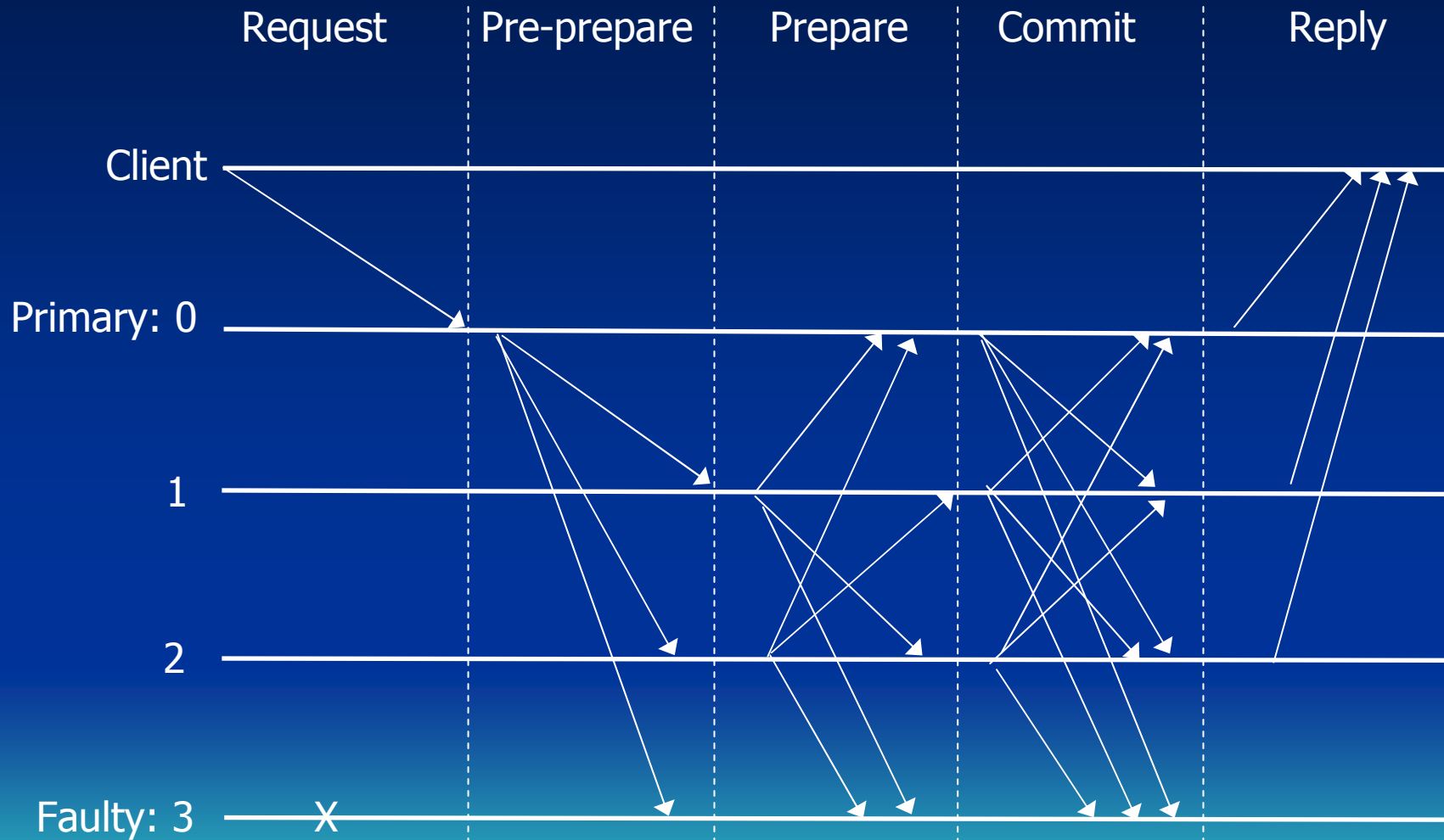
Our Approach and Contributions

- BFT-WS extends the WS-RM standard
 - Backward compatible with WS-RM
 - Can easily upgrade a non-replicated WS-RM server for high availability and dependability
- BFT-WS is 100% based on the WS platform (SOAP and WSDL)
 - Interoperable (of course the same BFT algorithm must be used)
 - Extensible, in particular, upgradable
- BFT-WS uses MIT's BFT algorithm for efficiency

Background

- Byzantine Fault Tolerance
- Web Services Reliable Messaging

Byzantine Fault Tolerance



Byzantine Fault Tolerance

- It takes three phases for all correct replicas to reach a Byzantine agreement on the total ordering of requests **in normal operation mode**
- The first two phases ensure if two correct replicas decide on an ordering, they agree on the same ordering
- The third phase is needed to ensure replicas that decide on different views to agree on the same ordering
- A new view is installed whenever the primary fails

Web Services Reliable Messaging

- WS-RM describes a reliable messaging (RM) protocol between two endpoints, termed as RM source (RMS) and RM destination (RMD)



A *sequence* is a unidirectional reliable channel between an RMS and an RMD

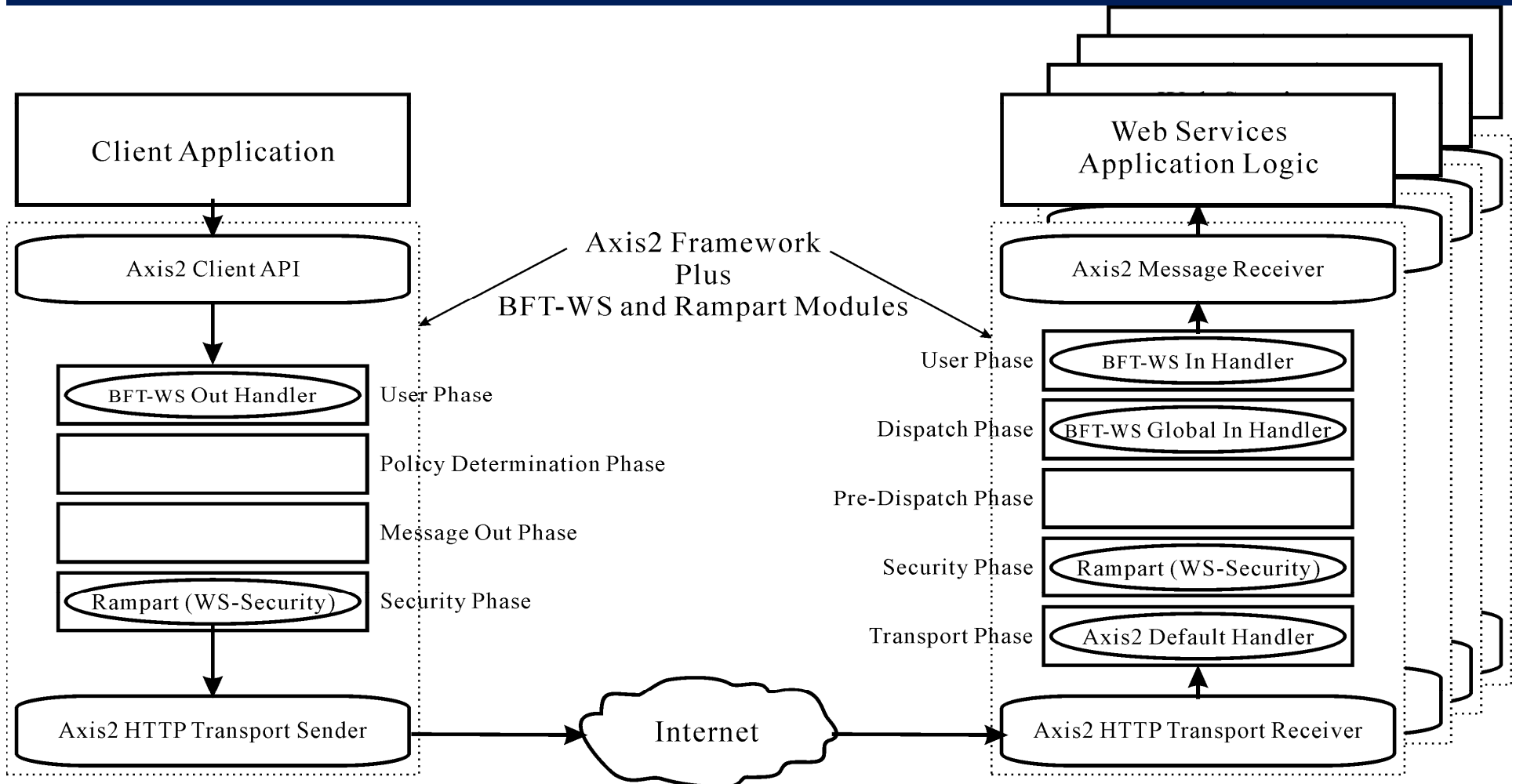
Web Services Reliable Messaging

- Message reliability: achieved by retransmission and positive acknowledgement mechanisms
- Quality of services provided:
 - AtMostOnce
 - AtLeastOnce
 - Exactly-Once
 - InOrder: any of the above three can be combined with InOrder delivery
- WS-RM is widely implemented. We use Sandesha2 (<http://ws.apache.org/sandesha/sandesha2/>)

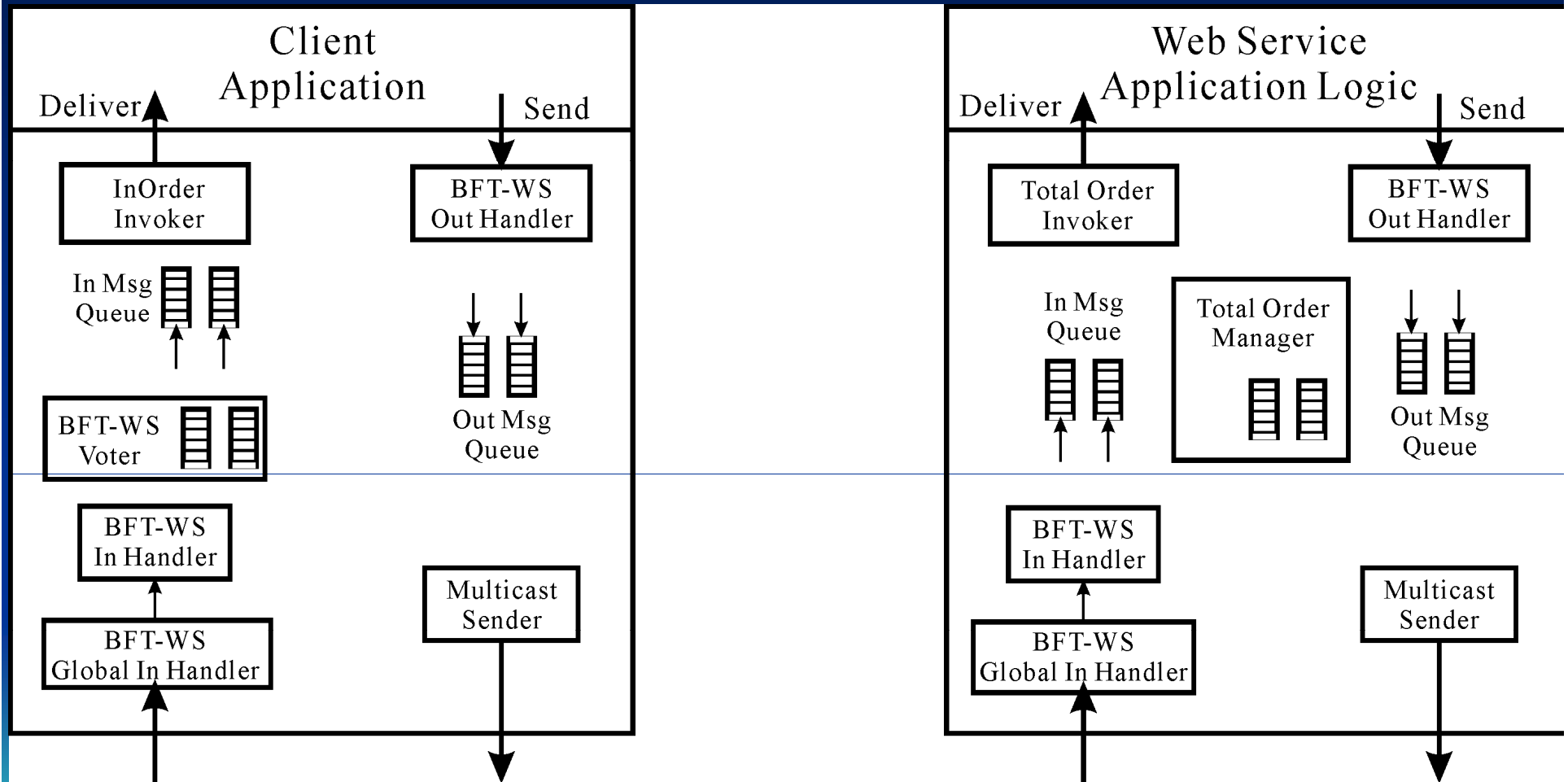
BFT-WS System Architecture

- Software Used
 - Apache Axis2 (SOAP Engine)
 - Sandesha2 (WS-RM)
 - Rampart (WS-Security)
 - Tomcat (Java Container)
- Architecture Overview
- Components

BFT-WS Architecture Overview



BFT-WS Main Components



BFT-WS Out Handler

- BFT-WS Out Handler: performs out-flow processing for reliable messaging
 - generates create-sequence/terminate-sequence request
- To enable duplicate detection, include a UUID string in the create-sequence request
- To ensure consistent Sequence ID among the replicas: use client supplied UUID

BFT-WS Multicast Sender

- Maintains the mapping between the virtual group sequence and the actual sequence used for each replica
- Performs garbage collection only when acks received from all destinations
- Client always multicast its requests to all replicas => prevent selective attack on the primary

BFT-WS Global In Handler

- The Sandesha Global In Handler performs duplicate filtering on application messages
- BFT-WS Global In Handler disable duplicate filtering on the client side to enable voting

BFT-WS In Handler

- Axis2 dispatches all application messages targeted to the BFT-WS-enabled services and the BFT-WS control messages to this handler
- On client side: all msgs passed to BFT-WS Voter
- On server side:
 - All app msgs passed to In Message Queue
 - All BFT control msgs passed to Total Order Manager

BFT-WS Voter

- BFT-WS Voter only exists on client side
 - Verifies authenticity of app msgs received
 - Collects responses
 - Performs voting: deliver a response only when $f+1$ identical response messages are received

BFT-WS Storage Manager

- Storage Manager consists of
 - In Message Queue
 - Out Message Queue
 - Other data structures for sequence, ack, retransmission management and in-order delivery

BFT-WS Total Order Invoker

- This invoker is responsible to deliver requests in the total order
 - Runs on a separate thread and polls msgs in In Message Queue for the next eligible msg for ordering and delivery
 - Inform Total Order Manager if a msg available for ordering
 - Obtain info from Total Order Manager for the next msg to be delivered

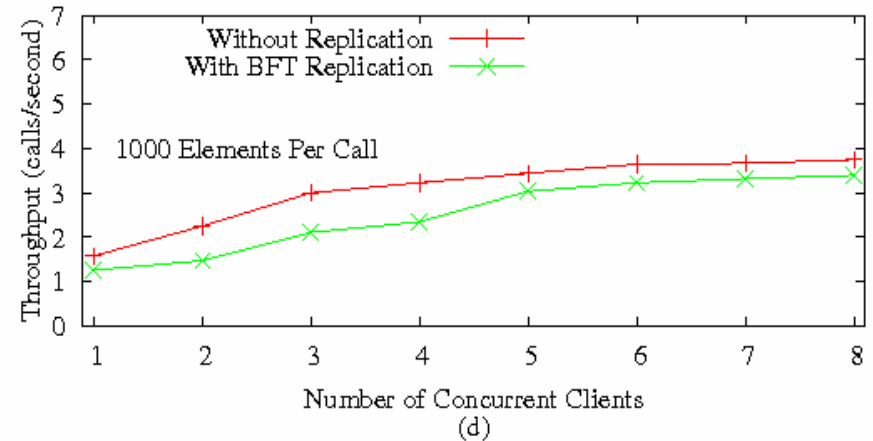
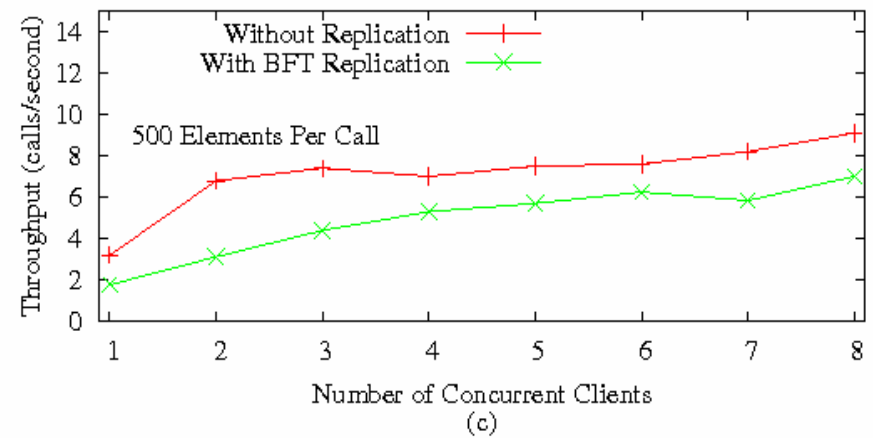
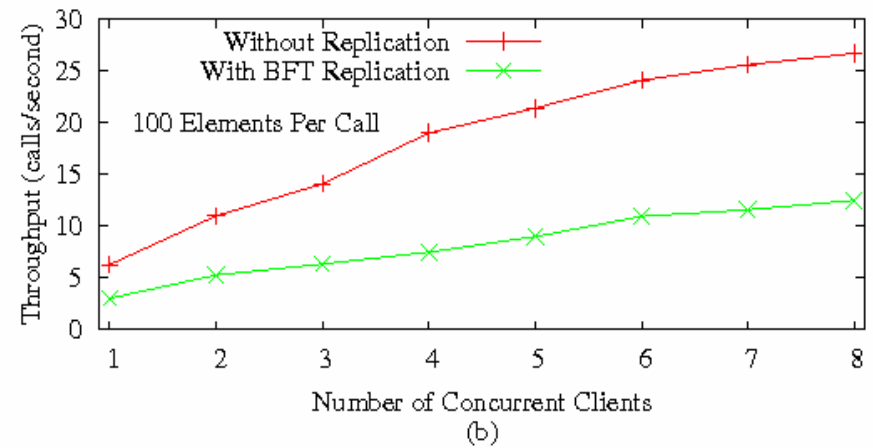
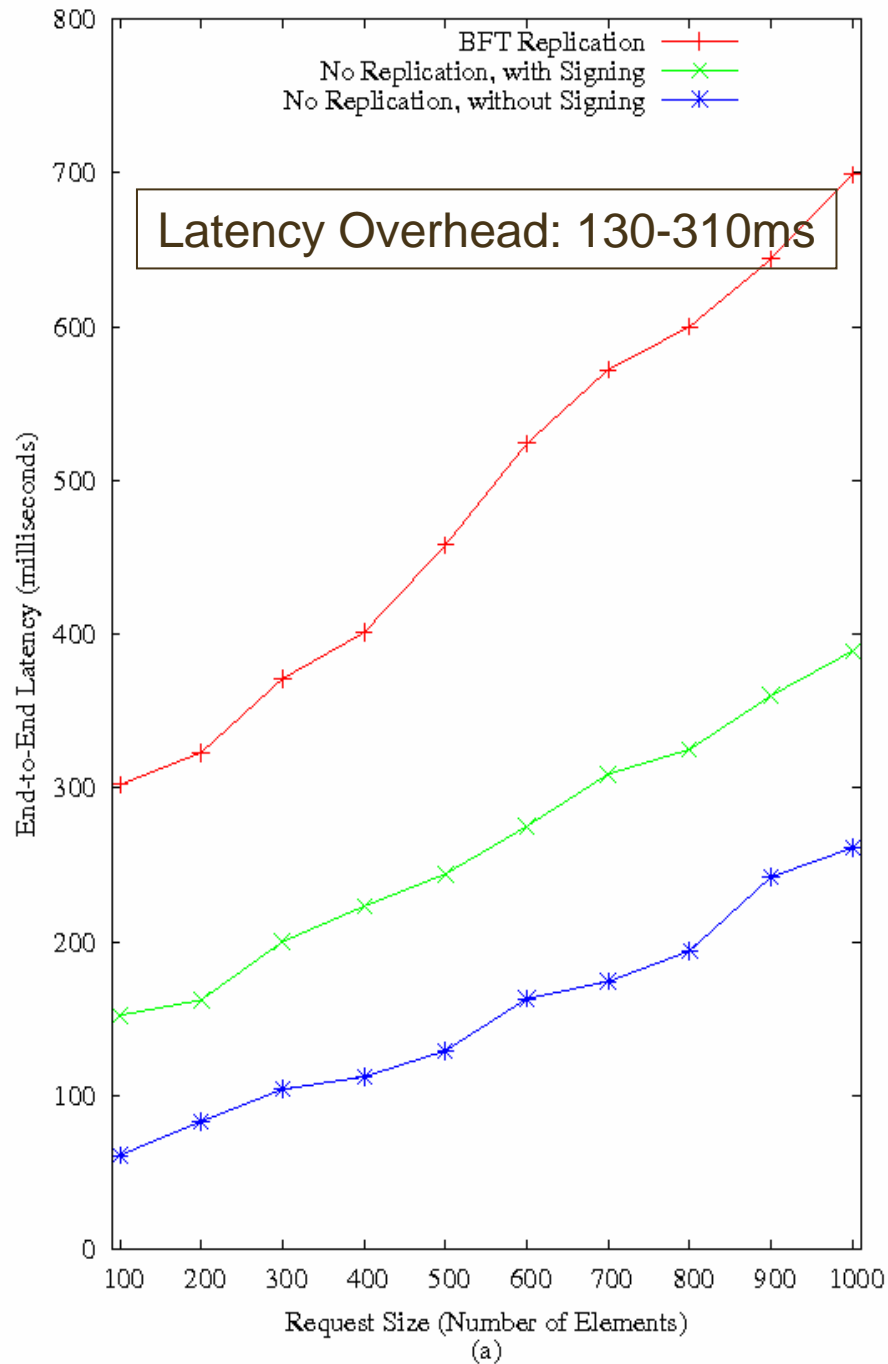
BFT-WS Total Order Manager

- Responsible for imposing a total order on all app requests
- Drives the BFT algorithm
- Internally use WS-RM sequences to communicate with other replicas

Performance Evaluation

- Testbed consisting of 12 Dell SC440 servers connected by a 100Mbps Ethernet (2.8 GHz CPU, 1GB RAM, SuSE 10.2 Linux)
- An echo Web service is used as the micro-benchmark application





Detailed latency measurement for a particular run with a single client and 4 replicas

Processing Step	Latency (ms)
Request out-processing	9.7
Request multicast	127.5
Request in-processing	10.6
Request ordering and delivery	306.9
Request processing at application	43.6
Reply out-processing	5.4
Reply send	78.5
Reply voting	11.2
Reply in-processing	8.7
Reply delivery	16.3
Message signing & verification (derived)	130.0
Total	748.4

Conclusion

- We described the design & impl. of BFT-WS
- BFT-WS uses standard WS technology to build the BFT framework for better interoperability and extensibility
- We documented in detail the architecture and the major components of our framework
- BFT-WS is carefully tuned to exhibit optimal performance

Future Work

- Future work will focus on the expansion of the feature set of BFT-WS
 - Support for multi-tiered Web services
 - Support for transactional Web services
- Incorporating speculative BFT for further performance improvement