A Byzantine Fault Tolerant Distributed Commit Protocol

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

Presented by: Loredana Groza

Today's Agenda

1. Introduction
2. Implementation
3. Performance
4. Conclusions

1. Introduction

The Introduction

Topic

- *Byzantine fault tolerant distributed commit protocol for transactions running over un-trusted networks*

Problem

- Two-phase commit protocol (2PC) does not work if the coordinator is subject to arbitrary faults because a faulty coordinator might send conflicting decisions

Solution

- *The traditional two-phase commit protocol is enhanced by replicating the coordinator and by running a Byzantine agreement algorithm among the coordinator replicas.*

+

About the algorithm

- Is based on Byzantine fault tolerance (BFT) algorithm , which is designed to ensure totally ordered atomic multicast for requests to a replicated stateful server.

What brings new Byzantine fault tolerant distributed commit (BFTDC) :

- is adapted to atomic distributed commit
- in the first phase uses a decision certificate
- is integrated with Kandula, a well-known open source distributed commit framework for Web services.

Kandula is an implementation of the Web Services Atomic Transaction Specification (WS-AT) .

+ Advantages of the new algorithm

- *can tolerate Byzantine faults at the coordinator replicas and a subset of malicious faults at the participants.*

- *a decision certificate, (set of registration records + a set of votes from participants => reach a Byzantine agreement for each transaction.)*

The certificate also limits the ways a faulty replica can use towards non-atomic termination of transactions, or semantically incorrect transaction outcomes.

2. Implementation

+

Implementation

- BFTDC protocol (with the exception of the view change mechanisms)
- integrated it into a distributed commit framework for Web services in
Java programming language.

The extended framework is based on a number of ApacheWeb services projects:
- Kandula (an implementation of WS-AT),
- WSS4J(an implementation of the Web Services Security Specification)
- Apache Axis (SOAP Engine) .

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 a Byzantine agreement on the transaction outcome, and to enable voting.

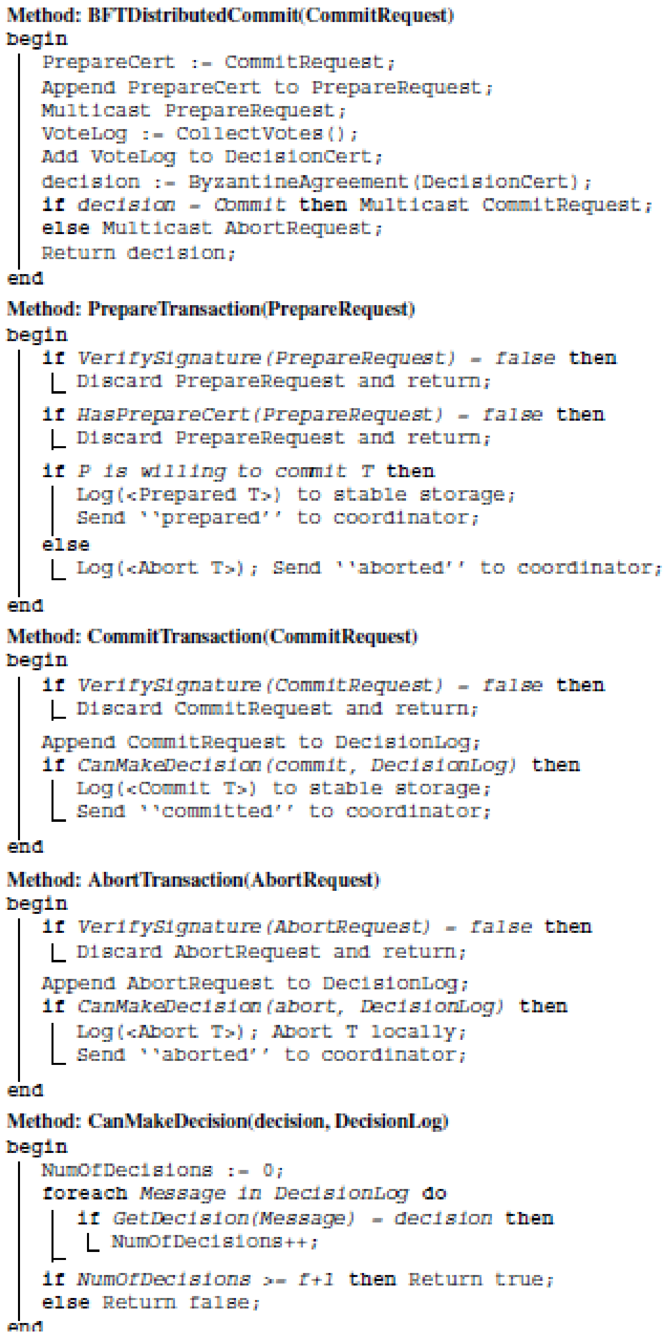
Implementation

Comparing with the 2PC protocol, there are two main differences:

– At the coordinator side, an additional phase of Byzantine agreement is needed for the coordinator replicas to reach a consensus on the outcome of the transaction, before they notify the participants.

– At the participant side, a decision (commit or abort request) from a coordinator replica is queued until at least f+1 identical decision messages have been received, unless the participant unilaterally aborts the transaction.
This is to make sure that at least one of the decision messages come from a correct coordinator replica.

The pseudo-code for the proposed Byzantine fault tolerant distributed commit protocol



```
Method: BFTDistributedCommit(CommitRequest)
begin
    PrepareCert := CommitRequest;
    Append PrepareCert to PrepareRequest;
    Multicast PrepareRequest;
    VoteLog := CollectVotes();
    Add VoteLog to DecisionCert;
    decision := ByzantineAgreement(DecisionCert);
    if decision = Commit then Multicast CommitRequest;
    else Multicast AbortRequest;
    Return decision;
end
Method: PrepareTransaction(PrepareRequest)
begin
    if VerifySignature(PrepareRequest) = false then
        Discard PrepareRequest and return;
    if HasPrepareCert(PrepareRequest) = false then
        Discard PrepareRequest and return;
    if P is willing to commit T then
        Log(<Prepared T>) to stable storage;
        Send ''prepared'' to coordinator;
    else
        Log(<Abort T>); Send ''aborted'' to coordinator;
end
Method: CommitTransaction(CommitRequest)
begin
    if VerifySignature(CommitRequest) = false then
        Discard CommitRequest and return;
    Append CommitRequest to DecisionLog;
    if CanMakeDecision(commit, DecisionLog) then
        Log(<Commit T>) to stable storage;
        Send ''committed'' to coordinator;
end
Method: AbortTransaction(AbortRequest)
begin
    if VerifySignature(AbortRequest) = false then
        Discard AbortRequest and return;
    Append AbortRequest to DecisionLog;
    if CanMakeDecision(abort, DecisionLog) then
        Log(<Abort T>); Abort T locally;
        Send ''aborted'' to coordinator;
end
Method: CanMakeDecision(decision, DecisionLog)
begin
    NumOfDecisions := 0;
    foreach Message in DecisionLog do
        if GetDecision(Message) = decision then
            NumOfDecisions++;
    if NumOfDecisions >= f+1 then Return true;
    else Return false;
end
```

3. Performance

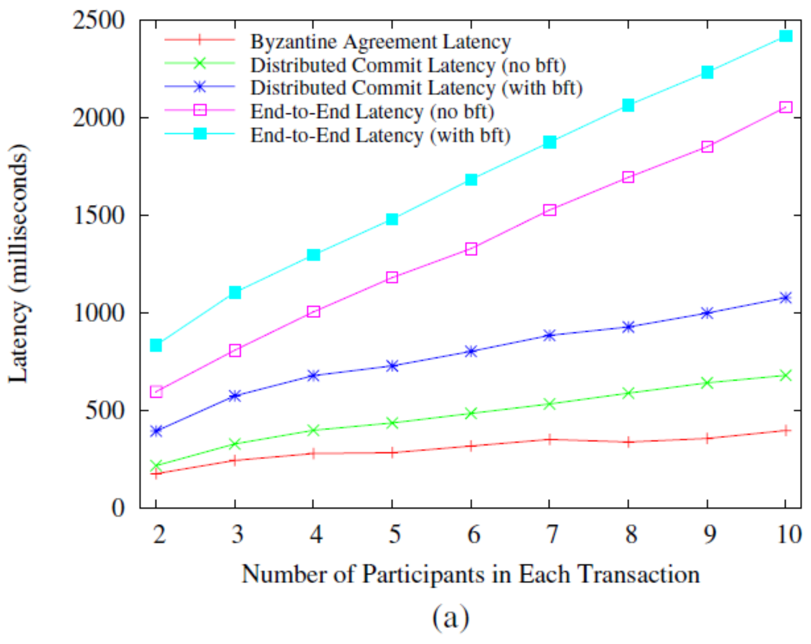
+ Performance

- 20 Dell SC1420 servers connected by a 100Mbps Ethernet
- each server -2 Intel Xeon 2.8GHz processors and 1GB memory
running SuSE 10.2 Linux.

- simple banking Web services application :
 - A bank manager (*i.e.,* initiator) transfers funds among the participants within the
scope of a distributed transaction for each request received from a client.
 - The coordinator-side services are replicated on 4 nodes to
tolerate a single Byzantine faulty replica.
 - The initiator and other participants are not replicated, and run on
distinct nodes.
 - The clients are distributed evenly (whenever possible) among the remaining nodes.
 - Each client invokes a fund transfer operation on the banking Web service within
a loop without any “think” time between two consecutive calls.

- each run, 1000 samples
- end to-end latency for the fund transfer operation is measured
at the client
- latency for the distributed commit and the Byzantine agreement is measured at the
 coordinator replicas
- throughput of the distributed commit framework is measured at the initiator for various number
of participants and concurrent clients.

Performance



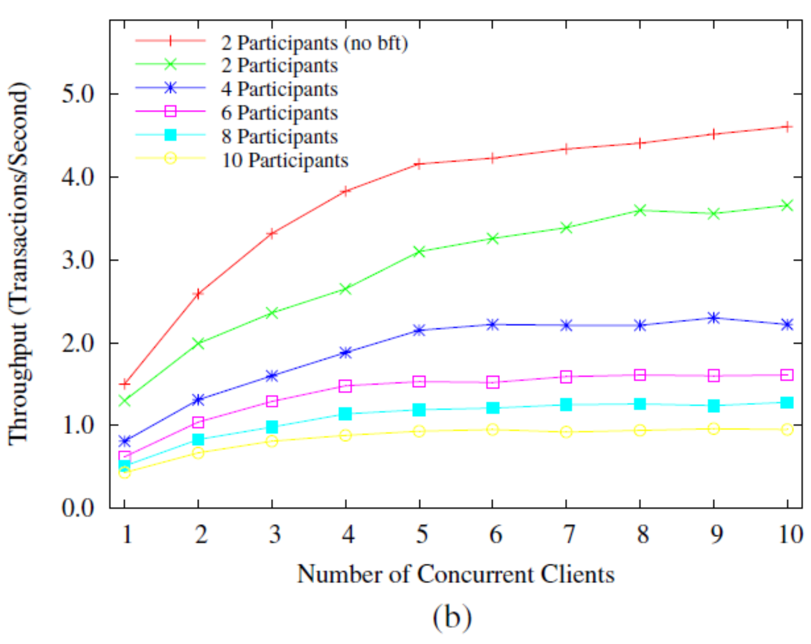


Figure 2. (a) Various latency measurements for transactions with different number of participants under normal operations (with a single client). (b) Throughput of the distributed commit service in terms of transactions per second for transactions with different number of participants under different load.

Moderate runtime overhead during normal operations.

4. Conclusions

+

Conclusions

- adapted Castro and Liskov’s BFT algorithm to ensure Byzantine agreement on the outcome of transactions.

- If a correct coordinator replica ba-commits a transaction t with a commit decision, the registration records of all correct participants must have been included in the decision certificate, and all such participants must have voted to commit the transaction.

- ensures that all correct coordinator replicas agree on the same decision regarding the outcome of a transaction.

- The BFTDC protocol guarantees atomic termination of transactions at all correct participants.

- Moderate runtime overhead during normal operations.

Take home message

Trust must be built !