Fault tolerance in distributed systems

1. 1. Fault Tolerance in Distributed Systems Submitted by Sumit Jain Distributed Systems(CSE-510)
2. [2.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-2-638.jpg?cb=1404288499)Topics 1.Introduction 2.Basics Terminologies 3.Phases in the fault Tolerance. 4.Fault Tolerance Techniques 5.Limitations
3. [3.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-3-638.jpg?cb=1404288499)1.Introduction • In the early days of computing, Centralized systems were in use. • Due to the gradual development in the field of networks and micro-electronics, centralized systems lost their ways to Interconnected-multiprocessor systems. • Interconnected multiprocessor systems 1.Parallel Processing systems-Single system wide memory. 2.Distributed Systems-No shared memory.
4. [4.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-4-638.jpg?cb=1404288499)1.1 What is a Distributed System • Collection of independent computers that appear to its users as a single coherent system. • Every system has its own memory and its own set of resources and they can share some common peripheral devices such as a printer. • Systems are organized in such a manner so as to hide their existence from the end user. • Transparency is to be maintained-ISO(8 kind of transperanacy) • Message passing or RPC technique through communication technology such as TCP/IP.
5. [5.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-5-638.jpg?cb=1404288499)1.2 • Designing a distributed system is a complex task because of the presence of a large number of components which can be located at distance from each other. some of the major challenges that designers have to face are listed below • Fault tolerance • Communication primitives • Flexibility • Transparency
6. [6.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-6-638.jpg?cb=1404288499)1.3Block Diagram of Distributed Systems
7. [7.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-7-638.jpg?cb=1404288499)2.Faults, Errors and Failures. • In any distributed system, three kinds of problems can occur. 1) Faults 2)Errors(System enters into an unexpected state) 3)Failures • All these are inter related. • It is quite fair to say that fault is the root cause, where a problems starts, error is the result of fault and failure is the final out come.
8. [8.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-8-638.jpg?cb=1404288499)2.1Types of Faults Transient Faults Permanent Faults Occur for a very short duration. Permanent Hard to locate Easy to be identified Do not affect the system to a great extent. Can cause severe damage to the entire system Network fault, processor fault, Media Fault are some of the examples Node Level Faults-when an Entire node is unavailable.
9. [9.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-9-638.jpg?cb=1404288499)2.2 Types of Failures Crash Failure Occurs when a server crashes or any other hardware related problem occurs. Omission Failure Occurs when a server does not receive incoming requests from client or fails to send messages in response to clients request. Timing Failure Occurs when a server fails to respond with in a particular amount of time. Response failure Occurs when a server sends incorrect message in response to the client’s message. Arbitrary Failure Occurs when a server sends any arbitrary message.
10. [10.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-10-638.jpg?cb=1404288499)2.3 What is Fault Tolerance? • Ability of a system to continue functioning in the event of a partial failure. • Though the system continues to function but overall performance may get affected. • Distributed systems are made up of a large number of components, developing a system which is hundred percent fault tolerant is practically very challenging. • Two main reasons for the occurrence of a fault 1)Node failure -Hardware or software failure. 2)Malicious Error-Caused by unauthorized Access.
11. [11.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-11-638.jpg?cb=1404288499)2.4 Why do we need fault tolerance? • Fault Tolerance is needed in order to provide 3 main feature to distributed systems. 1)Reliability-Focuses on a continuous service with out any interruptions. 2)Availability - Concerned with read readiness of the system. 3)Security-Prevents any unauthorized access. • examples-Patient Monitoring systems, flight control systems, Banking Services etc.
12. [12.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-12-638.jpg?cb=1404288499)3.Phases In The Fault Tolerance • Implementation of a fault tolerance technique depends on the design , configuration and application of a distributed system. • In general designers have suggested some general principles which have been followed. 1)Fault Detection 2)Fault Diagnosis 3)Evidence Generation 4)Assessment 5)Recovery
13. [13.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-13-638.jpg?cb=1404288499)3.1 Phases In The Fault Tolerance Fault Detection • Constantly monitoring the performance and comparing it with expected outcome. •Fault is reported if there is a deviation from expected outcome. Fault Diagnosis •Done to understand the nature of the fault and possible root cause. Evidence Generation •Report generated based on the outcome of the fault diagnosis. Assessment • Understanding the extent of the damage caused by the faulty component. •Done by examining the flow of information that has passed out from the faulty component to the rest of the system. •A virtual Boundary is created. Recovery Making the system fault free and restoring it to a consistent state- Forward recovery and Backward recovery.
14. [14.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-14-638.jpg?cb=1404288499)4.Fault Tolerance Techniques Replication • Creating multiple copies or replica of data items and storing them at different sites • Main idea is to increase the availability so that if a node fails at one site, so data can be accessed from a different site. • Has its limitation too such as data consistency and degree of replica. Check Pointing • Saving the state of a system when they are in a consistent state and storing it on a stable storage. • Each such instance when a system is in the stable state is called a check point. • In case of a failure, system is restored to its previous consistent state. • Saves useful computation.
15. [15.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-15-638.jpg?cb=1404288499)4.1 Replication • Can be used only for deterministic processes. • Main idea is to create multiple copies of data and storing them at multiple locations. • If one site goes down, data is accessed from other so that performance in not affected. • An effective approach but have limitations also 1.Data consistency(Data should match) 2.Degree of Replica(Exact Numbers of Replica).
16. [16.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-16-638.jpg?cb=1404288499)4.2 Types of Replication Active Replication • Can be used only for deterministic processes. • Client’s request is processed by all the servers. • Requires an atomic broadcast protocol ,to forward the requests to all the servers in the same order.
17. [17.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-17-638.jpg?cb=1404288499)Passive Replication • Can be used for non deterministic processes also. • There is only one server that processes client’s request known as primary server . • Other servers act as back up servers. • Response time is high as there is only one server which process many client’s request.
18. [18.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-18-638.jpg?cb=1404288499)4.3 Check Pointing • Every system has some information associated with it which defines its state at a particular moment. • This information include process state, environment, value of the active registers and variable. • All this information are collected and stored and each such instance is called a check point. • In the event of a failure, system is restored to a previously stored check point rather than starting it from the beginning. • Check pointing is useful but time consuming.
19. [19.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-19-638.jpg?cb=1404288499)4.4 Types of Check Pointing User triggered •Requires User Intervention. •Can be useful only when users have understanding of the computation. •It is not easy to identify when a check point should be created Uncoordinated Check pointing • Also known as Independent check pointing. •Processes do not communicate with each other and creates their own check points. •No communication overhead Coordinated Check points •Processes communicate with each other. •firstly, temporary check points are created and then made permanent. •Recovery time is high, due to communication. Message Based Check pointing Suitable when communication is through message passing only. The state of the processes is stored in the form of message. if one process goes down , other takes its place and acquire its state with the help of messages.
20. [20.](https://image.slidesharecdn.com/faulttoleranceindistributedsystems-140702080630-phpapp02/95/fault-tolerance-in-distributed-systems-20-638.jpg?cb=1404288499)4.5 Limitations Replication • Difficult to manage as the no. replica or copies increases. • Consistency and degree of replica is a major issue. • Active replication can be used only for deterministic processes. • Passive replication can be used for non deterministic but its recovery time is high. Check Pointing • Lost of computation • Check point length and check point frequency and storage is a major issue. • User level check pointing requires users involvement, but not every user is capable of doing it. • Coordinated check pointing includes communication over head work.

At the heart of every distributed system is a consensus algorithm. Consensus is an act wherein a system of processes agrees upon a value or decision. In this post let’s look at two famous consensus protocol namely two phase and three phase commits widely in use with the database servers.

The processes propose values for others and then agrees upon a value if it’s confident that every other process also has agreed upon the same value. The consensus has three characteristics

Agreement — all nodes in N decide on the same value.

Validity — The value that’s decided upon should have been proposed by some process

Termination — A decision should be reached !!

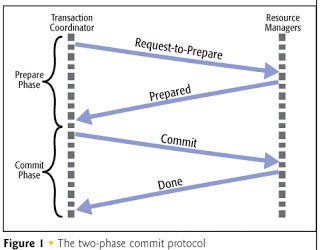
#### Two phase commit

This protocol requires a coordinator. The client contacts the coordinator and proposes a value. The coordinator then tries to establish the consensus among a set of processes (a.k.a Participants) in two phases, hence the name.

1. In the first phase, coordinator contacts all the participants suggests value proposed by the client and solicit their response.

2. After receiving all the responses, the coordinator makes a decision to commit if all participants agreed upon the value or abort if someone disagrees.

3. In the second phase, coordinator contacts all participants again and communicates the commit or abort decision.



Img Courtesy, <https://docs.particular.net/nservicebus/azure/understanding-transactionality-in-azure>

We can see that all the above-mentioned conditions are met. The agreement is there because the participants only make a yes or no decision on the value proposed by the coordinator and don’t propose values. Validity is there because the same decision to commit or abort is enforced by the coordinator on all participants. Termination is guaranteed only if all participants communicate the responses to the coordinator. However, this is prone to failures.

When speaking about failures what are the types of failures of a node?

Fail-Stop Model, Nodes just crash and don’t recover at all.

Fail Recover Model, Nodes crash, and recover after a certain time and continue executing.

Byzantine failure, Nodes start behaving arbitrarily trying to interrupt the regular behavior.

Now coming to how failures affect 2 Phase commit,

1. Coordinator fails even before initiating phase 1. This literally means the consensus isn’t started at all and theoretically the protocol works correctly.

2. Coordinator fails after initiating phase 1. Some nodes have received the message from coordinator initiating a fresh round of 2PC. These nodes might have sent their responses and are blocked waiting for the 2nd phase of 2PC to start. This also means that no future consensus rounds of 2PC can start. One way out of this issue is to have time outs when waiting for responses. So when a node times out waiting for a response from the coordinator it can assume that coordinator is dead and take over the role as coordinator. It can reinitiate phase 1 and contact all other nodes asking them for the consensus based on the value for which this node voted as a participant before the actual coordinator crashed. However if another node crashes before recovery node gathers all messages of phase 1, then the protocol can’t proceed. This is because recovery node doesn’t know what’s the intended decision of the crashed node. If all other participant nodes have agreed to commit but the newly crashed node might have intended to abort. So the recovery node can’t call the decision as a commit. This argument applies vice versa also.

3. Similarly, if a participant fails during phase 1 before the coordinator receives a response from the participant, the protocol comes to a grinding halt. The reasoning is similar as point 2, because coordinator doesn’t know the result of failed node and hence can’t proceed to commit or abort the consensus.

4. Similarly, if coordinator fails during phase 2 we would want a node to take over and shepherd the protocol to completion. Another big issue is that if a participant node fails during commit phase the system is left to lurch in the dark because the coordinator doesn’t know whether the participant failed after committing or before committing. Hence coordinator can’t proactively decide whether the transaction is committed.

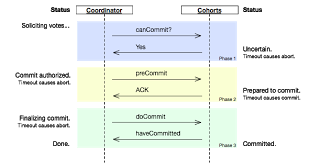
Summarizing one of the biggest disadvantages of two phase commit is that it’s blocking protocol. If a cohort sends an agreement message to the coordinator it holds the resources associated with the consensus till it receives the commit or abort message from the coordinator. The failure of coordinator will prevent the cohorts from recovering from a failure. The same reasoning applies to failure of a participant.

#### Three phase commit

This is an extension of two-phase commit wherein the commit phase is split into two phases as follows.

a. Prepare to commit: After unanimously receiving yes in the first phase of 2PC the coordinator asks all participants to prepare to commit. During this phase, all participants acquire locks etc, but they don’t actually commit.

b. If the coordinator receives yes from all participants during the prepare to commit phase then it asks all participants to commit.



Img Courtesy, CC BY 3.0, <https://en.wikipedia.org/w/index.php?curid=16452453>

The pre-commit phase introduced above helps us to recover from the case when a participant failure or both coordinator and participant node failure during commit phase. The recovery coordinator when it takes over after coordinator failure during phase2 of previous 2 pc the new pre-commit comes handy as follows. On querying participants, if it learns that some nodes are in commit phase then it assumes that previous coordinator before crashing has made the decision to commit. Hence it can shepherd the protocol to commit. Similarly, if a participant says that it doesn’t receive prepare to commit, then the new coordinator can assume that previous coordinator failed even before it started the prepare to commit phase. Hence it can safely assume no other participant would have committed the changes and hence safely abort the transaction.

s