SANTA CLARA UNIVERSITY
THE JESUIT UNIVERSITY IN SILICON VALLEY

Analysis of Performance and Data Consistency in Quorum-based Storage with Fault Tolerance

Group 5

Longling Wang
Jing jin
Yuan Song
Zhengyu Chen

Abstract

Quorum-based technique is implemented to ensure transaction atomicity and serializability in the presence of network partitioning. However, there is not too much research focusing on how to achieve the data consistency within the writing quorums while doing the write operation. Especially, when any node in the cluster could crash at any time during the whole process, we need a mechanism to handle these unpredictable situations. Three mechanisms are introduced in this proposal. There are No-Phase commit, Two-phase commit and Three-phase commit. Two-phase commit and Three-phase commit protocols are originally from the implementation of transactions between multiple databases. Potential problems and different solutions of applying these commit protocols are discussed. We expect that the Three-Phase commit model is the best solution with approximate 100% data consistency without blocking encountering for quorum-based protocol in P2P overlay network. In this project, we implement these three kinds of approaches and evaluate them in a quorum-based data store system simulator. The challenge is how to simulate the ad hoc scenario, especially simulate multi nodes randomly crash concurrently and continuously in the quorum-based system, to support the whole evaluation process. we implement an automated evaluation system, with a customized quorum-based data store system, which consists ten quorum nodes and provides data store service, and client node, which starts quorum-based write and read operation and analyze the result automatically. The result basically satisfies our hypotheses, especially when the probability of crash is high. First, inconsistent case occurs under no-phase quorum approach. Second, system blocking problem occurs under two-phase quorum approach, and this problem increase latency greatly. Third, three-Phase approach solves inconsistent and blocking problems efficiently.

Keywords: Quorum, two-phase commit protocol, three-phase commit protocol, fault tolerance, data consistency

Content

1. Introduction

1.1 Objective

- Introduce 2-phase and 3-phase commit mechanism into a strict quorum mechanism to improve availability and keep data consistency in a quorum-based store system with churn
- Evaluate traditional quorum, 2-phase quorum, 3-phase quorum in terms of availability and consistency quantitatively in a simulated quorum-based store system with churn

1.2 What are the problems

Distributed data store system need to be high available. Replication mechanism is the fundamental of high availability. Quorum mechanism is a common method to distribute the replicas through the whole system. But in distributed environment, especially in a cloud-based infrastructure, a situation, which happens so much frequently that cannot be ignored, is that a node in the data store system are not such stable. In practice, even multiple nodes might crash concurrently because of failure of hardware or fatal error of software. This situation might cause two main problems:

1.1.1. Data inconsistency

- Problem description

A traditional quorum-based node write the data into disk immediately, after it receives a write operation request. It is prone to cause data inconsistency in this quorum under two situations: The first one is if a node or multiple nodes in this quorum is/are failed to write this data into disk. That means the data in failed nodes is different with the one in other nodes. The second is if a node or multiple nodes crash after it received the write operation request but before it write the data into disk.

- Approach description

For this problem, we can introduce 2-phase commit into traditional quorum to solve this problem, and we call this method 2-phase quorum. 2-phase quorum means we divide a write operation into two phase. A coordinator will coordinate the two process. Same as 2-phase commit, 2-phase quorum consists of voting and decision phase.

- ➢ Voting phase
  - ◆ Coordinator sends all nodes in quorum a VOTE REQUEST
  - ◆ All nodes in the quorum respond COMMIT or ABORT. If responding COMMIT, the node turn into ready status.
- ➢ Decision phase
  - ◆ Coordinator decides commit or abort: if any participant (or coordinator) voted ABORT, decision must be abort. Otherwise, commit.
  - ◆ Coordinator sends all nodes in this quorum decision
  - ◆ Participants (who have been waiting for decision) commit or abort as instructed and ack.
- ➢ Recover

- When a node restart, it will check if there is pre-apply information in disk. If it found pre-apply information, it will check the operation result from the coordinator. If the result is commit, this node write the data into disk. If the result is abort, this node delete the pre-application information. If the coordinator is not online, this node will try to ask the other nodes in this quorum.

1.1.2. System blocking

As mentioned before, we can use 2-phase quorum mechanism to solve the data inconsistency problem. But 2-phase quorum also has a fatal problem need to be solved. Because a node should follow the instruction of the coordinator, it will be blocked when it can't receive and find out what the next step is, when coordinator crashed and all alive nodes are ready status.

1.3 Our approach

- 3-phase quorum

We try to introduce 3-phase commit mechanism into quorum system to solve the blocking problem, and we call this method 3-phase quorum. 3-phase quorum add an additional phase, commit phase, to 2-phase quorum. With this additional phase, it must be possible for nodes in this quorum to decide the next step without waiting for the recovery of the coordinator or dead node

- Voting phase
  - Same as 2-phase quorum
- Decision phase
  - Almost same as 2-phase quorum, EXCEPT change commit to Prepare-to-commit and all nodes don't execute the operation only respond ACK to coordinator
- Commit phase:
  - After getting all ACK, coordinator sends commit command to all nodes.
  - All nodes execute the write operation and response ACK
- Recover
  - Same as 2-phase quorum

1.4 Why this is a project related this class

Nowadays, distributed data store system is widely used in P2P or cloud environment. Quorum technique also be widely used in lot of popular distributed system, which usually deployed in cloud framework, such as Apache Cassandra, Voldemort, Amazon Dynamo, etc.

1.5 Area or scope of investigation

This project focus on distributed data store system and quorum technique and quorum-based system.

2. Theoretical bases and literature review

2.1 Definition of the problem

Replication of data is used to achieve higher availability even if during failures of some storage nodes. However, concurrent updates to multiple replicas may lead to inconsistent situation if there are no control methods while some storage nodes crash and then recovery.

2.2 Theoretical background

2.2.1 Voting-based Quorum

Originally, Thomas (1979) proposed a "majority consensus" algorithm, which based on voting scheme to preserve the mutual consistency of replicated data in the database. [1]

Gifford (1979) proposed a quorum-based voting scheme for replication control. [2] In the algorithm, each copy is assigned a number of votes. And the operation needs to follow the following rules. Every transaction collects a read quorum of r to read a file and a write quorum of w to write a file, such that r + w is greater than the total votes assigned to this file. And w should be greater than half of the total votes. The first rule ensures that there is a non-null intersection between read and write quorum such that a data object cannot be read and written concurrently and read operation at least have a most current copy. The second rule ensures that two write operations cannot happen concurrently.

Huang etl. (1988) introduced a quorum-based commit and termination protocol to maintain high data availability in case of concurrent site failures, lost messages and network partitioning. [3]

Quorum based technique has been widely used to implement various services and applications in distributed systems which helps to achieve consistent operations, especially in distributed storage and replication.

2.2.2 Three phase commit

Three-phase commit is a model for atomic commit protocol in distributed system. It is first introduced by Skeen (1981) as a non-blocking protocol for preserving transactional atomicity. [4] It is then used to form a formal model for atomic commit protocol in distributed systems (Skeen, Stonebraker, 1983). [4]

2.3 Our solution

In this paper, we proposed to implement a writing protocol to preserve the consistency when the storage nodes crash by using the three-phase commit protocol and compare the consistency result with that under two-phase commit protocol and that commit directly without any acknowledgements.

2.4 Why it is different and better?

We combine the three-phase commit protocol with the quorum based distributed storage system in order to avoid the inconsistency which is due to nodes failure. And we compare the result under two-phase commit with that under three-phase commit protocol. Theoretically, by using the two-phase commit protocol, all the nodes inside the write quorums would be blocked if the initiator node crashes. While using three-phase commit protocol, the consistency can be preserved without blocking.

3. Hypothesis

3.1 Hypothesis 1:

For the implementation of quorum-based protocol in P2P overlay networks, if there is no control for data consistency among the writing quorums, and the nodes may crash at any time during writing operations, then the data consistency cannot be guaranteed under this situation, which may cause the wrong reading results.

3.2 Hypothesis 2:

For the implementation of quorum-based protocol in P2P overlay networks, if combining with Two-Phase commit, then the data consistency will be guaranteed no matter when the nodes crash during writing operations. However, if there is any failure of coordinator (coordinator is temporary per request), then potential blocking problem can happen, which will cause low efficiency and long latency of writing operations. Blocking happens when the whole cluster has to wait until the coordinator is recovered to clarify what is the decision made by the coordinator, and then continue the operations.

3.3 Hypothesis 3:

For quorum-based protocol in P2P overlay networks, if Three-Phase commit is implemented for controlling the data consistency in writing quorums, then the data consistency in the whole cluster will be approximately 100% without blocking problem.

4. Methodology

4.1 Input data collection and generation

Each node will randomly generate a value as an input data and write to the quorum. And there is a monitor program to read the data from each node and analyze the data.

4.2 How to solve the problem?

Because we cannot guarantee every node in the quorum can finish the writing task. So we use 2-phase commit to ensure all node in quorum has consistent data. However, 2-phase commit has a potential blocking problem when nodes crash, so that we using 3-phase commit instead of 2-phase commit to keep data consistency under any circumstances without blocking.

4.3 Algorithm Design

In this project, we assume each node knows the topology of the network. Each node will randomly invoke and write data to quorum. The node which is doing this operation would be the leader of this operation.

Before write, the coordinator has to start a quorum voting. The coordinator randomly picks nodes add to the quorum until the size of quorum satisfies the requirement. These nodes will ack yes or no to the coordinator. Node who responds yes cannot respond yes to any other nodes. If the coordinator cannot get enough yes, it has to abort and notify nodes who has responded yes in order to unlock them. After randomly times, start this operation again.

Writing without commit

The coordinator send write request to all node in the quorum. Regardless of these nodes complete the write or not, the operation complete.

Writing using 2-phase commit

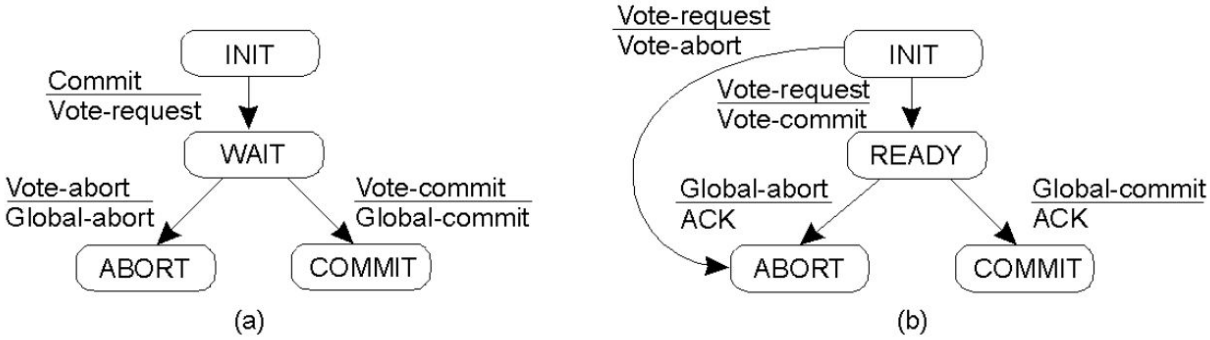


Fig. 1

a) 2-PC finite state machine for the coordinator

b) 2-PC finite state machine for a node

Each node has four stages, INIT, READY, ABORT, COMMIT, and starts with INIT stage. The coordinator first send vote-request to all node in quorum. Node will log its decision and ack commit or abort back. When coordinator receives all acks, and all nodes voted commit, it send commit to all nodes and all nodes commit the write. Otherwise, it send abort and all nodes

abort.

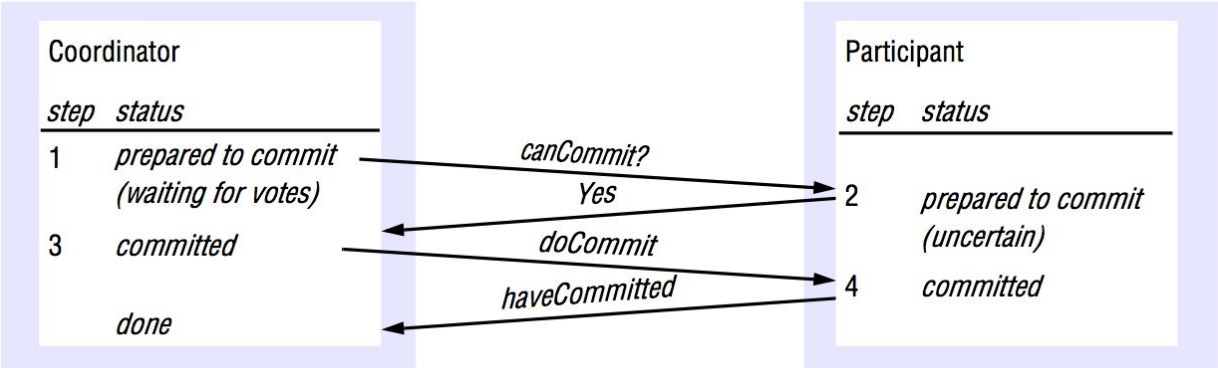


Fig. 2 Communications in 2-phase commit

Writing using 3-phase commit

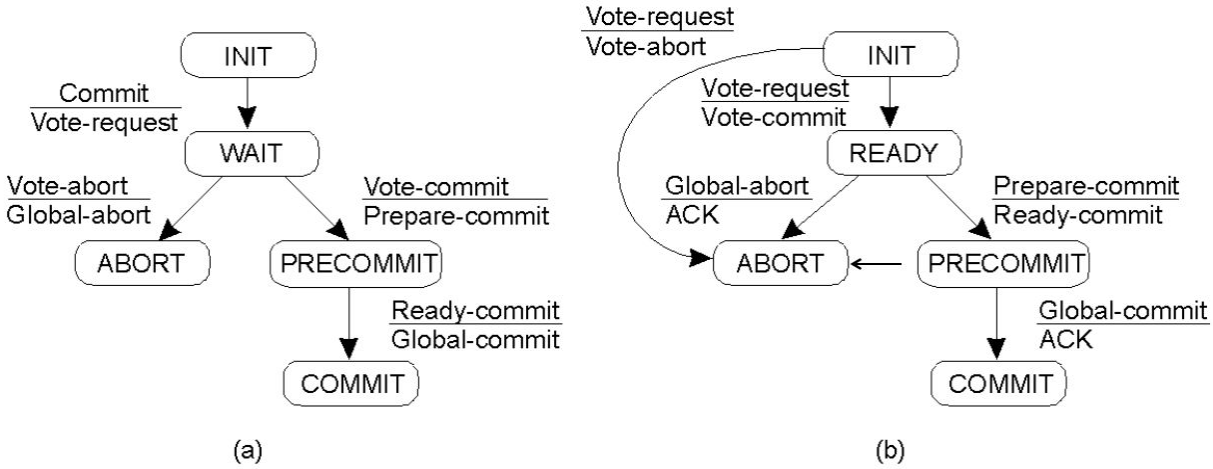


Fig. 3

a) 3-PC finite state machine for the coordinator

b) 3-PC finite state machine for a node

Three phases: Vote, Decision, and Commit. Coordinator will start the vote request, all nodes will ack commit or abort. The coordinator has to wait for all nodes ack and make a decision. If all nodes ack commit, then it will send pre-commit to all nodes. Otherwise, send abort to all nodes. Nodes will change its state to pre-commit or abort correspondingly and ack to the coordinator. When the coordinator collects all acks from nodes, it will sends commit to all nodes, so that all nodes will eventually commit the write and ack to the coordinator. The operation complete.

Test

We will randomly crash nodes in specific failure rates in order to test the system can tolerate any failures and maintain data consistency. During the interval between transactions, a monitor program will read values from every node to verify the data consistency.

We are going to repeat this test 50 to 100 times, and generate a table to show the consistent rates between this three different protocols.

Language used
Java programming language to implement this project.

Tools used
Linux server in Design Center.

4.4 How to generate output

○ There will be a monitor setting outside the whole cluster. It is designed for checking the data consistency under different commit protocols (such as no-phase, two-phase and three-phase). The nodes belonging to the cluster will only perform write operations and simulate random time failures, while the monitor consistently reads the value of data saving in all nodes in order to calculate the consistency rate.

○ Since the monitor will read the value of data in all nodes for each read operation and the number of nodes is N, if the writing quorum is equal to NW, then the number of same values we get out from N should be NW. If this is the case, we claim that the data for this read operation is consistent. The read operation will be executed by monitor for many times (approximately 50 – 100 times), and the consistency rate will be calculated from the times of consistent reading / total times of reading * 100% (table 1).

Data from node 1	Data from node 2	…	Data from node 10	Number of same value	If consistent
67	67		90	7	Yes
55	67		90	5	No
78	67		78	7	Yes
…	…	…	…	…	…
167	223		233	7	Yes
Times of total reading		Times of consistent reading		Consistency rate	

1000	987	98.7%

Table 1. The output for data consistency rate while number of nodes is 10 and the number of writing quorums is 7.

- The read operation will only be performed while all nodes in the cluster are alive so that we will be able to read the values of data in all nodes and calculate the consistency rate.

- Since the read operation is performed in this quorum based protocol P2P overlay networks, the number of reading quorums should follow the rule as: NW + NR > Total and NW > Total / 2, even the monitor does not belong to the cluster. This is designed for guaranteeing the serializability in order to avoid reading and writing happens at the same time. Therefore, for every reading request, the monitor needs to get enough reading quorums and then lock these nodes from other operations, and eventually get the values of data from all nodes (the reading can be performed even not all nodes are locked).

- Table 1 shows the output for data consistency rate while number of nodes is 10 and the number of writing quorums is 7. This similar table will be created for each commit model to get the data consistency rate of those three commit models.

- At the end of each commit model simulation, every node will notify the monitor for the completion of their program execution. Then, the monitor will gather the information regarding on the times of blocking problem encountering. In other word, at the end of each commit model, every node will report how many times they encountered the blocking situation.

- Monitor will present a table at the end of whole project for a comprehensive comparison of data consistency rate and times of blocking encountering, which is shown in table 2 as an example.

Commit model	Data consistency rate	Times of blocking encountering	Total number of write operations
No-Phase commit	87.5%	0	156
Two-Phase commit	99.6%	43	149
Three-Phase commit	99.8%	0	160

Table 2. Conclusive table of comprehensive comparison between all commit models

4.5 How to test against hypothesis

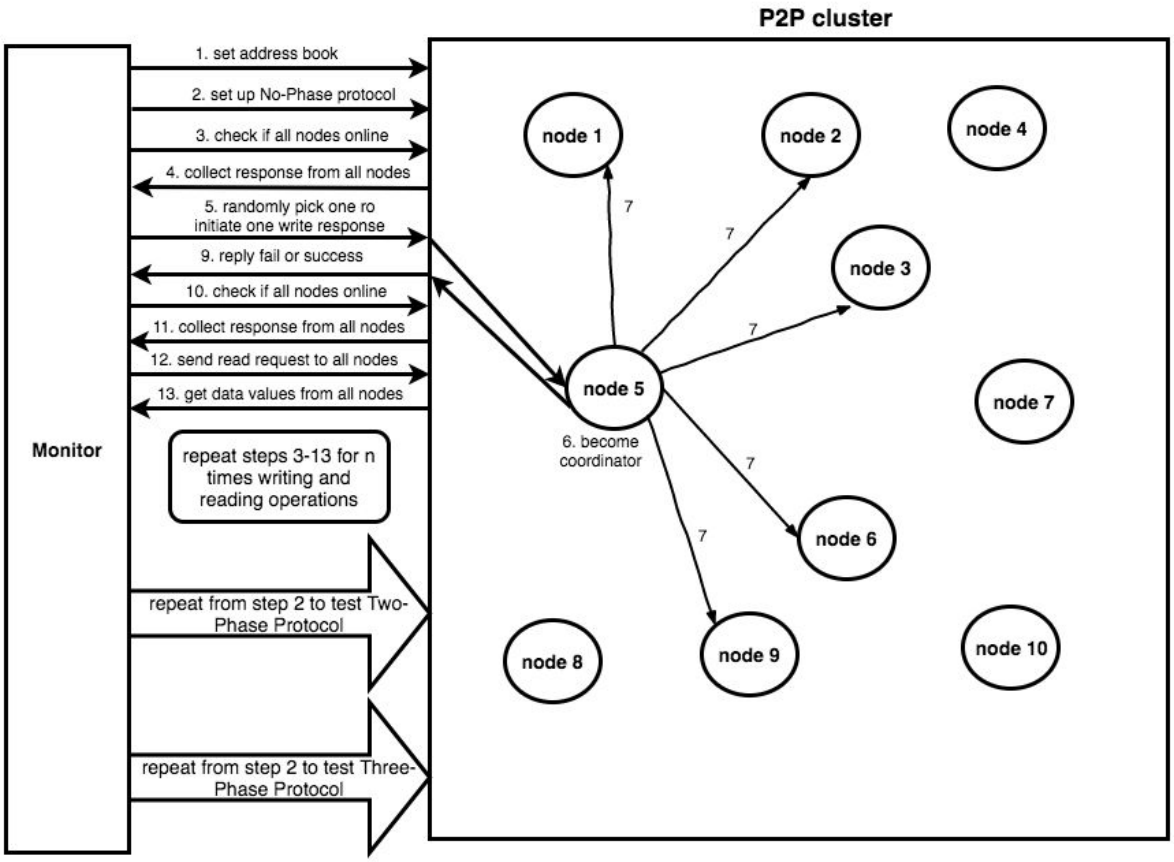
As introduced previously, 5-10 nodes will form a cluster in P2P overlay network. And a monitor will be another machine outside this cluster to consistently read the values from all nodes. There will be an interface named “CommitModel” in order to switch from different commit models. No-Phase commit, Two-Phase commit and Three-Phase commit will implement this “CommitModel” interface. There is only one abstract method in “CommitModel” interface named “execute (args)”. This method will realize the following functions:

1. Each node repetitively sends a write request at random time following the quorum base protocol. Every time before sending the request, the current node needs to make sure every other node is alive. Otherwise, draw back for a random period before resend the request.
2. Therefore, before the execution of this write operation, the requesting node needs to get enough writing quorums and lock these nodes whoever sends back “approval” vote for this writing request. Otherwise, the requesting node must release all votes and unlock these nodes if not enough votes are gathered.
3. After getting enough votes, the requesting node becomes temporary coordinator and release the actually writing operation to the writing quorums. Therefore, it is coordinator per request model.
4. The followed operations and logics are differentiated with different commit models.
5. All nodes may crash at random moments, no matter if current node is a general node or a coordinator node. And different commit models will have different mechanisms to handle different types of failures at different stages.
6. All the nodes should be able to recover after certain time period to continue the execution.
7. Blocking situation is defined as whenever the coordinator is failed, and all the nodes cannot get a certain decision by asking around their peers. Blocking problem will be detected by alive nodes in cluster and the counter of blocking times in local will be incremented. The counter will be finally reported to monitor.

For testing against Hypothesis 1, 2 and 3, all the nodes may crash at anytime as mentioned previously, and the values of data will be tracked by monitor through consistent reading. Therefore, the inconsistent data should be caught by monitor. Besides, the number of blocking encountering will be tracked by nodes and reported to monitor eventually. Therefore, if we simulate those three commit model and gathered the data as we expected, then we may conclude that our hypotheses are solid.

5. Implementation

5.1 Design document and flowchart



- 7: randomly chooses nodes and sends replicating request
- 8: not showing on graph, handle the request depending on different protocol

The flowchart of project design is shown as above. There are two sets of systems that are communicating between each other. The Monitor is acting as the client which will persistently send writing requests to update the data saving in this P2P cluster, and also read the data from all nodes to check consistency. Nodes system has more than one nodes to act as a P2P cloud network.

Initially, monitor will collect all IP and Port information from nodes, and send the address book (step 1), which contains all IP and Port information to the nodes in cluster so that the nodes can communicate with other nodes. After that, monitor will test three protocol one by one. Starting from No-phase protocol, monitor will send message to all nodes and set No-Phase protocol

environment for testing(step 2). After setting is done, monitor will ping all nodes to make sure every node is alive without crashing (step 3 and 4), and initiate a writing request with a value randomly created. The writing request will send to a random node in cluster (step 5). After that node received this request, it becomes a temporary coordinator that is in charge of this writing request(step 6). Coordinator will randomly choose available nodes as replicas for this writing basing on the number of writing quorums(step 7). The number of writing quorums can be customised regarding on different testing purpose. Then, the process for handling the writing request is different according to different protocols (step 8). The detailed process of three protocols refer to part 4.3. Coordinator will decide if this request is successfully proceeded or failed due to crash encountering among the nodes or coordinator itself, and return this result to monitor (step 9). The execution time of every write request will be recorded. The starting time is right after monitor sends out the request, and the end time is whenever the monitor receives a fail or success response.
After monitor gets the response from coordinator, it will send a reading request to all nodes in the cluster(step 12). It will also check to see if every node is alive before sending out this reading request (step 10 and 11). After collecting the value from all nodes, monitor compared the value with last writing request to see if the data gets updated to the right value. If it is, then we record this writing as consistent writing operation. then, step 3-13 will be repeated n times depending on how many writing and reading operation we want to test. After the testing on No-Phase protocol, a conclusive table (similar to table 1) will be shown in terminal.
Two-Phase protocol and Three-Phase protocol will be tested one by one after No-Phase protocol with same number of writing/reading operation times.

6. data analysis and discussion

6.1 Output generation

There are 10 participating nodes in our writing quorum system and the number of write quorum is set to be 6. For each experiment, the monitor will initiate 10 write operations under each protocol and certain crash rate. And we will collect the data including write operation fail rate, total and average execution time in cases of success writing and failed writing, consistent rate and blocking times, etc. The crash rate means each node will randomly crash at this rate before sending messages to other nodes. Then the crashed node will recover in the crash duration.

The output data is generated under different crash rate of 1/3, 1/20, 1/50, 1/100, 1/1000, 1/3000 respectively and the crash duration of 1 second.

6.2 output analysis

Under crash rate of 1/50, the average data output analysis.

- No Phase

In No Phase writing protocol, failure rate is 30% and the total execution time is 343 milliseconds. The average execution time of failed is 29 milliseconds and the average execution time of successful writing is 36 milliseconds. The time differentiate between those two times are very small, which is intuitive since no matter success or not, the no phase protocol will be terminated.

In our test, the average consistent rate is 85.71%, because coordinator or nodes maybe fail during the writing operation. If the coordinator failed during the operation, the data is inconsistent obviously, but there is another situation that the coordinator finished the operation, but the node in the writing quorum failed, the data will still inconsistent. Thus, no matter the operation is success or not, the data still can be inconsistent.

- Two-Phase Protocol

In two-phase protocol, failure rate is 10%, and the total execution time is 3236 milliseconds, the average execution time of failed writing is 109 milliseconds, and the average execution time of successful writing is 347 milliseconds and blocking 3 times. The execution time is much longer than no phase protocol, since we encountered average 3 times during the operation, nodes have to wait for nodes which has the commit information to recover. However, two-phase protocol can guarantee data consistency if the operation is success. Thus, the data consistent rate is 100%.

- Three-Phase Protocol

In three-phase protocol, failure rate is 10%, and total execution time is 1314 milliseconds, the average execution time of failed writing is 60 milliseconds, and the average execution time of successful writing is 139 milliseconds.

Three-phase protocol not only can guarantee 100% data consistent but also guarantee that no blocking problem during the operation.

6.3 Compare output against hypothesis

Performance

Protocol	Total Execution Time	Average Failed Execution Time	Average Success Execution Time
No-Phase	343	29	36
Two-Phase	3236	109	347
Three-Phase	1314	60	139

We compare the time of each operation to analysis performance of these three protocols. The total execution of No-Phase protocol is more than five of times faster than

the other two protocols. The reason is that No-Phase protocol does not guarantee data has been successfully wrote to the whole quorum, which means less messages communications between nodes and less network overhead. However, in Two-Phase and Three-Phase protocol, there are many messages between coordinator and nodes to ensure the data consistency, overhead slows down the whole process.

Let us look at the average time of failed execution, the Three-Phase Protocol takes shorter time than Two-Phase Protocol, however, both of them are much longer than No-Phase Protocol. We Discussed this situation and found out the reason as the following: first, there is a only way a Three-Phase Protocol can fail, operation abortion. Nodes crash would not lead to abortion in Three-Phase Protocol. So the Three-Phase Protocol do not need to wait crashed nodes recover to make the final decision. On the other hand, No-Phase Protocol would not wait for recover as Two and Three-Phase Protocols. Second, the reason of Two-Phase Protocol is slower than Three-Phase is that Two-Phase may encounter blocking problem. We test another time, in the most cases, the times differentiation is relatively closed to each other. The reason is that based on our PC's performance, we cannot do a large amount of writes, the result may not reflect the actual situation. Additionally, network congestion may affect the performance between Two-Phase and Three-Phase Protocols. Three-Phase needs more rounds of message communications, so the effect of network latency will be significant, which may cause worse performance.

The average of success execution time of No-Phase is still ten times faster the other two due to low overhead. The most important is that the execution time of Three-Phase is about three times faster than Two-Phase one, which proves our hypothesis that Two-Phase Protocol may encounter blocking problems if nodes crash. The blocking problem will cause all nodes pause and wait for informed nodes recover, which costs plenty of time.

Thus, operation time performance, No-Phase is the best and Two-Phase is the worst, Three-Phase is in the middle In our test cases.

Data Consistency

Protocol	Fail Rate	Consistent Rate	Blocking Times
No-Phase	30%	85.71%	0
Two-Phase	10%	100%	3
Three-Phase	10%	100%	0

Data consistency is another important things we concern. In hypothesis 1, we assume that No-Phase protocol has data inconsistency limitations. And the result also proves the hypothesis. Two-Phase and Three-Phase Protocols are in order to overcome the limitations. From the result, they did guarantee 100% data consistent rates, but difference between these two protocols are Two-Phase may encounter blocking problems, in our test, 3 times, which also proves our hypothesis 2 and hypothesis 3.

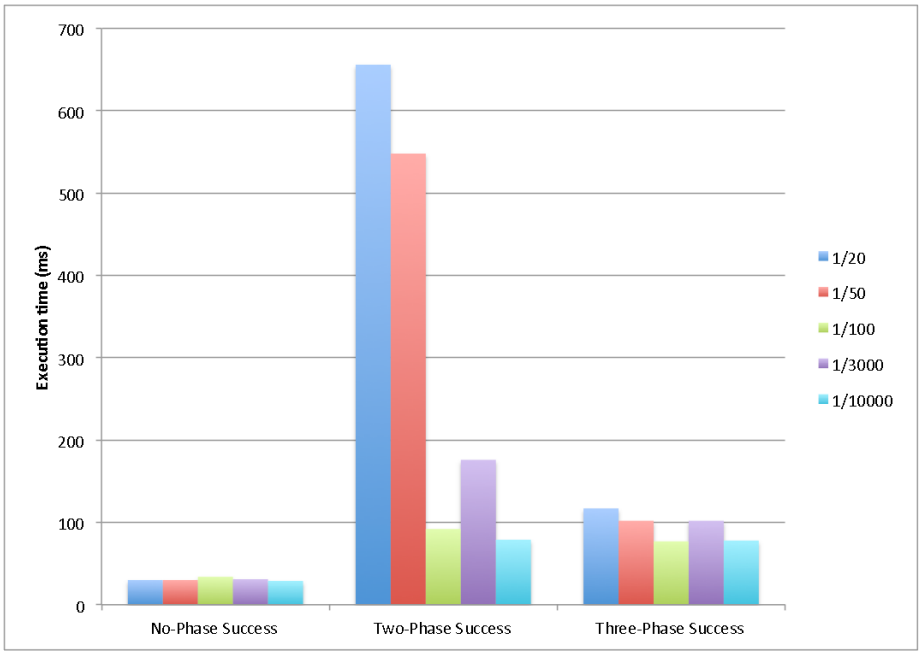


Figure 6.1 Execution time under different crash rate

Data Consistency in Different Crash Rate

Protocols	3	20	50	100	3000	10000
No-Phase	0	50	66.67	80	100	100
Two-Phase	0	100	100	100	100	100
Three-Phase	0	100	100	100	100	100

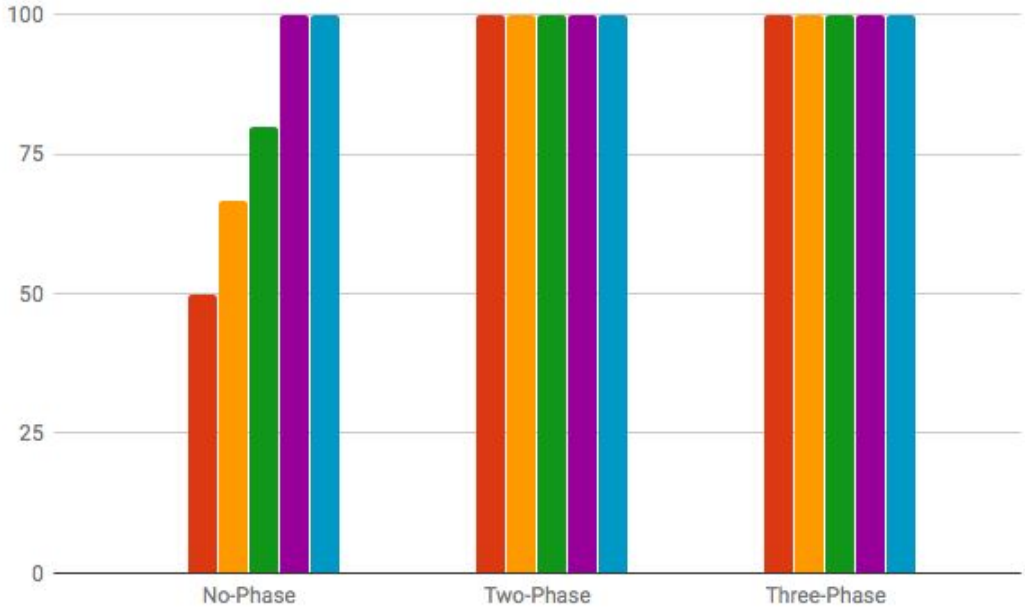


Figure 6.2 Consistent Rate in Different Crash Rate

In high crash rate, No-Phase protocol may have low data consistency rate, however, both Two-Phase and Three-Phase Protocol can guarantee 100% data consistent rate.

In real practice, we have to do some tradeoff to select the right protocol based on performance and data consistent rate.

6.4 Abnormal case explanation

We designed to run our program in Design Center's Linux machine. So we intended to do 100 times of writing in ten machines. But the Design Center's server has SSH connection limitations so that we simulate our program in our own laptop instead of servers in Design Center, which comes to a performance issue. We cannot do 100 times of writing, which will cause the memory leaking of the JVM, so we only can simulate 10 times of writing of each protocol. This limitation may cause the result we collected a little bit different than the actual situation. We ran multiple times of test to minimize this impact.

As we mentioned above, in some cases, the performance time fluctuates too much between different tests. We try to use the average data in all our tests and try to give a better output data.

Theoretically, the latency under Three-phase protocol would be longer than that under Two-phase protocol because Three-phase requires one more round trips of

communication. However, in our simulation, we tested the system in our local machines and the latency is not significant enough to show the network delay.

6.5 Discussion

In the industry practice, most of the companies are using eventually consistency model instead of strict consistency model due to performance issue. Under our test, we can easily find that No-Phase Protocol, which is no consistency control, has a much faster performance than other two protocols. If the crash rate is low, No-Phase protocol will have the best performance with a reasonable consistent rate. If the services is not sensitive to data consistency, relax the data consistency control is an option to improve performance.

Two-Phase and Three-Phase Protocols can guarantee data consistency if the operation succeed. But because of potential blocking problem in Two-Phase Protocol, Two-Phase Protocol may encounter longer execution time than Three-Phase Protocol if crash rate is relatively high. However, we have to consider the message communication overhead of Three-Phase Protocol, and trade off the total performance and choose a better performance protocol from these two.

7. conclusions and recommendations

7.1 Summary and conclusions

In this project, we simulated the writing operation in a quorum writing system under No-phase, Two-phase and Three-phase protocol and generated the performance result in terms of different metrics including operation fail rate, average execution time, consistent rate and blocking times. We execute 10 times of write operation under these three protocols and compared the performance under different crash rate of nodes.

7.2 Recommendations for future studies

Due to the SSH connection limitation of the Design Center, we can only test the system in our local machine. As a result, the network delay is too short. The latency of Three-phase protocol due to one more round trip of communication is not that longer than that of Two-phase protocol as we expected. However, this is not the case in reality and the execution time we get. In future study, we should test the result in real distributed environment.

In our experiments, we focused more on the effect of different crash rates on three different protocols. However, in the real life, the crash duration can vary a lot. And it may cause significant difference of performance between different protocols. This can be more emphasized

in future study in order to test how much impact of different crash durations on these three protocols.

In our design, node crash is simulated in each stage of sending messages to other nodes, which means the node will randomly crash each time it sends a message to other nodes. However, the crash rate we control is only for each stage and the general crash rate under different protocol is not well controlled. As a result, under different protocols, the situation of crash will not happen in exactly same basis. In the future research, we should better handle the simulation of crash in a comparable basis. Then we could generate more representative data to analyze.

Reference

[1] Thomas, Robert H. "A majority consensus approach to concurrency control for multiple copy databases." *ACM Transactions on Database Systems (TODS)* 4.2 (1979): 180-209.
[2] Gifford, David K. "Weighted voting for replicated data." *Proceedings of the seventh ACM symposium on Operating systems principles*. ACM, 1979.
[3] Ching-Liang, Victor OK, and A. Li. "Quorum-based Commit and Termination Protocol for Distributed Database Systems, Foirth Int." *Conference on Data Eng, Los Angeles, California*. 1988.
[4] Skeen, Dale. "Nonblocking commit protocols." *Proceedings of the 1981 ACM SIGMOD international conference on Management of data*. ACM, 1981.
[5] Skeen, Dale, and Michael Stonebraker. "A formal model of crash recovery in a distributed system." *IEEE Transactions on Software Engineering* 3 (1983): 219-228.