INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

UNIVERSITY OF WESTMINSTER (UOW)

MSc Advanced Software Engineering

7SENG007C.3 - Concurrent and Distributed Systems

Coursework 02

By

Name	Student ID	UOW Number	UOW Name
Kajendran Alagaratnam	20200630	18292844/1	w1829284

TABLE OF CONTENTS

1. Introduction

Inventory management systems are systems that allow organizations to track their merchandise or stock throughout the complete supply chain, from purchasing production to end sales. These systems control the way that organizations approach the inventory management of their business. Each organization will handle their merchandise in their own unique way, based on the nature and size of their business. A distributed system will be needed for organizations with large scale business as these organizations will have to handle a large number of stocks distributed across warehouses and factories that are distributed geographically. Such systems are expected to be scalable, highly available and fault tolerant in order to ensure that orders placed by factories are processed precisely.

2. Requirements

2.1 Inventory management system

2.1.1 Functional requirements

FR ID	**Requirement**
FR1	Inventory clerk should be able to update new arrival of stock items
FR2	Workshop managers from multiple factories should be able to make orders for different stock items
FR3	Orders should be placed on a first come first served basis
FR4	The system should allow managers to place orders simultaneously and should not allocate the same items for multiple orders
FR5	Clerk and workshop managers should be able to check the current inventory and order status

Table 1: Functional requirements

2.1.2 Non-Functional requirements

NFR ID	Requirement
NFR1	The system should be highly available
NFR2	The system should be scalable
NFR3	The system should maintain high accuracy and consistency

Table 2: Non-functional requirements

2.2 Use case diagram

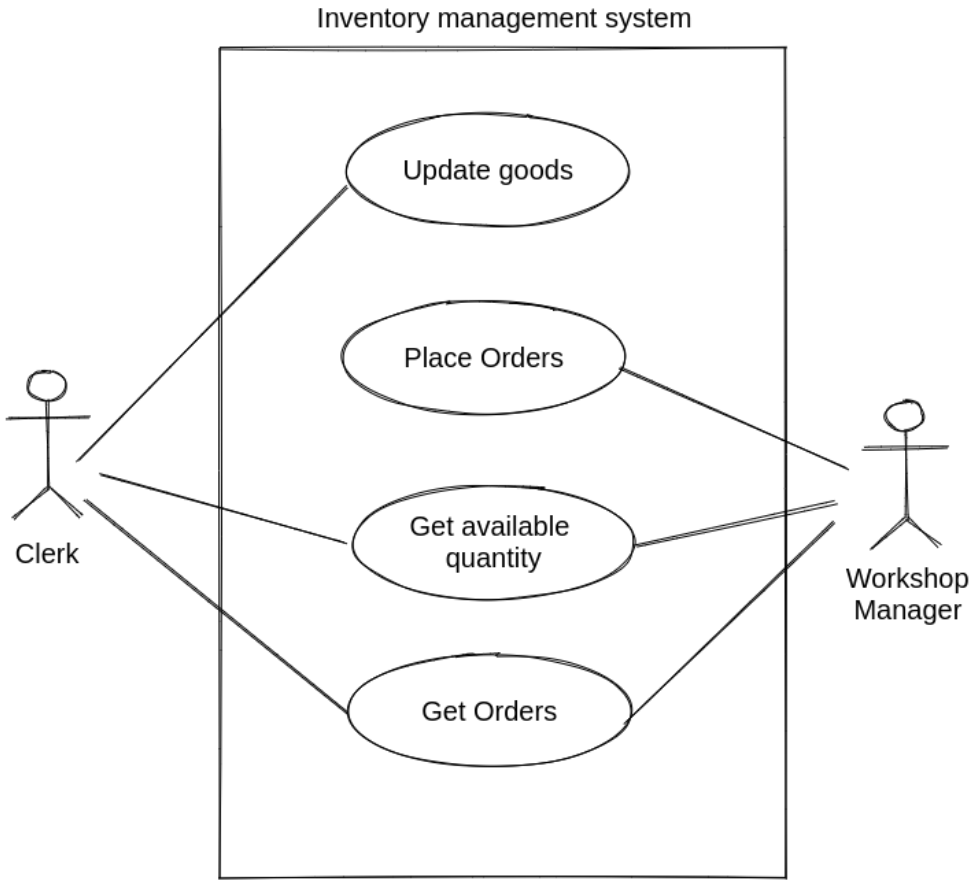

Figure 1: Use case diagram

2.3 Use case description

Use Case Name	Update goods
ID	UC-001
Description	Clerks should be able to update the available quantity of goods
Participating Actors	Clerk
Pre-conditions	Update quantity must be greater than zero
Main flow	1. Clerk selects the server that he/she wants to update 2. Clerk provides the updated quantity and makes the request
Alternative flow	None
Exceptional flows	None

Table 3: Update goods use case description

Use Case Name	Place orders
ID	UC-002
Description	Workshop manager should be able to place a new order or update an existing order
Participating Actors	Workshop manager
Pre-conditions	Order quantity must be less than available quantity
Main flow	1. Workshop manager selects the server that he/she wants to update 2. Workshop manager provides the quantity and the id of the order and makes the the request
Alternative flow	None
Exceptional flows	None

Table 4: Place orders use case description

Use Case Name	Get available quantity
ID	UC-003
Description	Clerk or workshop manager should be able to check the available quantity of stocks
Participating Actors	Clerk, Workshop manager
Pre-conditions	-
Main flow	1. Workshop manager or clerk selects the server that he/she wants to request from and makes the request
Alternative flow	None
Exceptional flows	None

Table 5: Get available quantity use case description

Use Case Name	Get Orders
ID	UC-004
Description	Clerk or workshop manager should be able to the list of current orders
Participating Actors	Clerk, Workshop manager
Pre-conditions	-
Main flow	1. Workshop manager or clerk selects the server that he/she wants to request from and makes the request
Alternative flow	None
Exceptional flows	None

Table 6: Get Orders use case description

3. Implementation

3.1 Architecture diagram

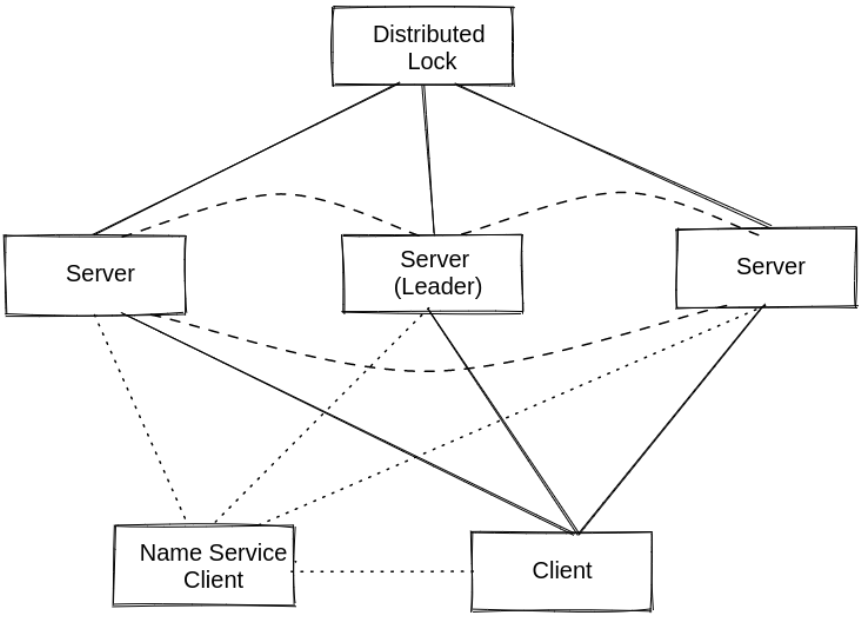

Figure 2: Architecture diagram

3.2 Components

Following are the components of the implemented distributed system and all of these components were written in Java.

- Distributed lock: Helps to coordinate the processes within the distributed system using Apache Zookeeper
- Name service client: Keeps track of the ip address and ports of the servers within the distributed system using Etcd key value pair storage
- Server: Stores the data, participates in two phase commits, ensures data consistency and responds to client requests. One of the server would be a primary node and the others would be secondary nodes
- Client: Communicates with a selected server using gRPC protocol inorder to perform the actions such as updating inventory, placing orders and checking current status of inventory and orders

3.3 Assumptions

Following assumptions were made when implementing the system

- There will be just three nodes in the implemented system
- There will be just one type of item
- Ignore the scenario in which all of the nodes end up crashing

3.4 Distributed architecture

A distributed system is a group of independent systems that seem to the users of the system as a single coherent system. Individual computers within this system will have diverse characteristics but will be working together in order to achieve a common goal.

Following are some of the benefits of opting for a distributed system

- Distributed systems are ideal when the system is meant to be highly scalable as this can be achieved by increasing the number of nodes in the system.
- Eradicate the single point of failure by allowing other nodes to take over when one node fails.
- Distributed architecture allows allows us to be economical by starting small and gradually grow
- Same resources can be accessed concurrently by many clients

3.5 Communication protocol

Within a distributed system, the servers and clients will need to communicate with each other in order to collaboratively work to achieve a single goal. The processes running on different nodes will have to communicate with each other and for this purpose, the author has chosen to use gRPC which is a cross platform open source high performance remote procedure call framework. gRPC allows us to build language agnostic remote procedure call services. The interface definition language that allows us to perform these communications is known as protobuf.

Stubs are pieces of code that can convert parameters transferred between the client and server during a remote product call. Client stube transforms the local procedure call to a network call and server stub intercepting the request transforms it inversely. An interface definition language such as protobuf can be used to describe the communication and stubs are generated by tools by looking at the interface definition.

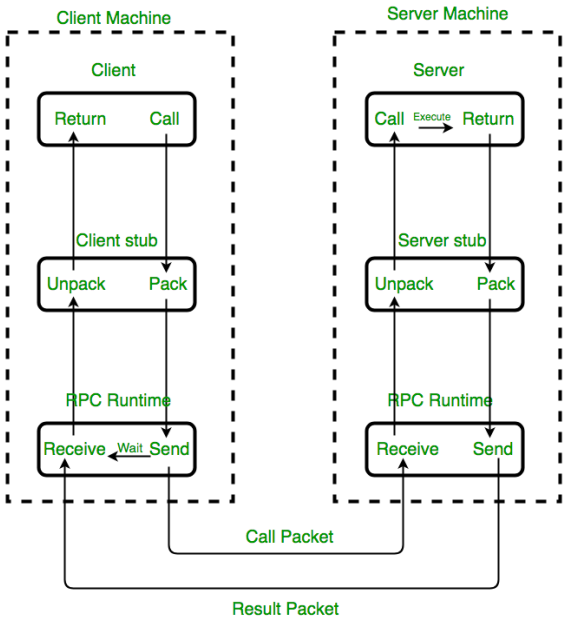

Figure 3: Implementation of RPC mechanism

3.6 Synchronization

Distributed systems consist of multiple nodes running independently and these nodes will use various approaches to communicate with each other and in order to achieve a single goal there needs to be an approach to coordinate with each other and this is achieved through approaches such as mutual exclusions, elections and coming to agreements on time.

Leader election is the process of selecting a node that will coordinate other processes. There are several algorithms such as the bully algorithm and the ring algorithm to select the leader process. In most cases the oldest process is selected as the leader within a distributed system

Apache ZooKeeper is an open source tool for coordinating applications within a distributed system by managing the state of the application. It supports features such as replicated databases, broadcast changes to other nodes and request processor that will allow the leader node to handle write requests

The implemented solution includes a distributed lock that was built with the help of znodes in Apache ZooKeeper. This lock will allow two processes to access a shared resource with mutual exclusion. A root znode will be created to represent a lock. If a process aims to get the lock, it will generate a child znode under the root node. When a process attempts to acquire the lock it will verify whether the requesting process has the lowest sequential number and grant the lock to that process and if not, the process will wait until it becomes the lowest. After accessing the resource, the process lets go of the lock by removing the znode it created.

3.7 Name service

Name services allow processes within a distributed system to discover the location of other services. Etcd is a highly available, distributed key value store which provides a reliable approach to store data that needs to be accessed by a distributed system. The implemented solution utilizes Etcd to implement a simple name service that allows services to register their address details and clients to discover the location of these services. This will ensure that the ip address of the services do not have to be hardcoded anywhere within the distributed system.

3.8 Data management

In a distributed system, data will be replicated by maintaining multiple copies of data. This ensures that high availability is achieved and that if one replicate gets corrupted, another copy can be used to continue the operations. This also helps with improving the performance within the distributed system as multiple processes can handle the request instead of allowing a single process to become a bottleneck. This also allows data to be placed closer to the client which in turn would reduce latency.

The issue with replicating the data is that it would be expensive to maintain the consistency of the replicas and in order to maintain consistency, all conflicting operations would be required to

be executed in the same order in all replicas. Logical clocks can be used when synchronizing operations globally.

3.9 Consistency Protocols

Consistency protocols are protocols that can be followed in order to ensure the consistency between the data stored in a distributed manner.

Following are the two kinds of consistency implementations

- Primary based approach where each data item has a primary copy which coordinates the writes. Data reads can be served using a local copy
- Replicated writes where writes are done on multiple copies simultaneously.

A primary based remote write protocol has been implemented in the provided solution and the following diagram illustrates how the remote write is performed. The solution will be using the distributed lock that was discussed in the previous sections in order to keep track of the primary node

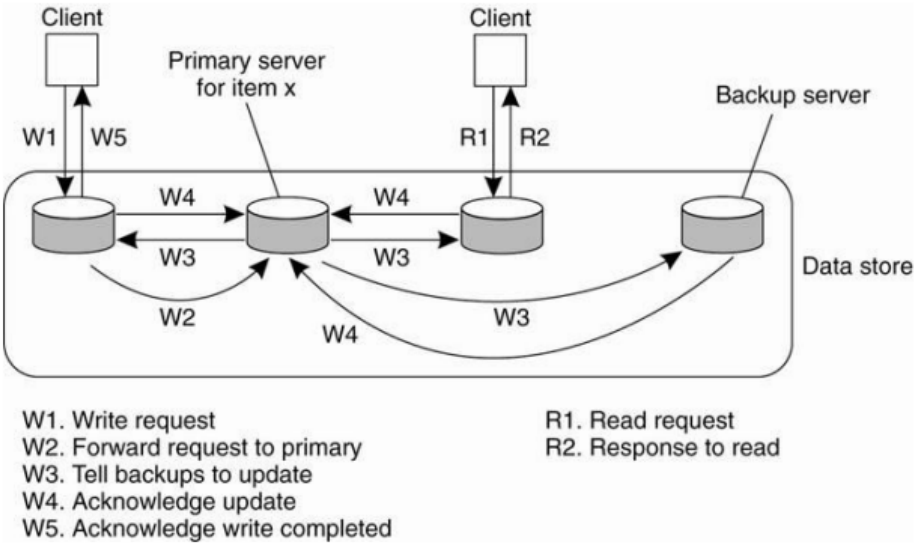

Figure 4: Primary based remote write protocol

According to the implemented solution, all nodes will be competing to acquire the lock and become the leader. The node that acquires the lock will act as the primary and other nodes will act as backup servers. A placeOrder and updateQuantity service has been added in order to manage the inventory and these update operations will follow the remote write protocol. The distributed datastore will be stored as in-memory variables. When a node acts as the primary node, it will require to call all of the secondary nodes in order to propagate the changes and when a node acts as a secondary node, it will require to inform the primary node about the update calls that it receives and the primary node will once again inform all the secondary nodes about the update.

3.10 Distributed Commit

Distributed commits ensure that an operation is carried out by all nodes in the group or none at all. Two phase commit has been implemented in the distributed inventory management system to ensure that modification to the data will only be carried out when all nodes agree to commit

The following diagram illustrates how a two phase commit is performed.

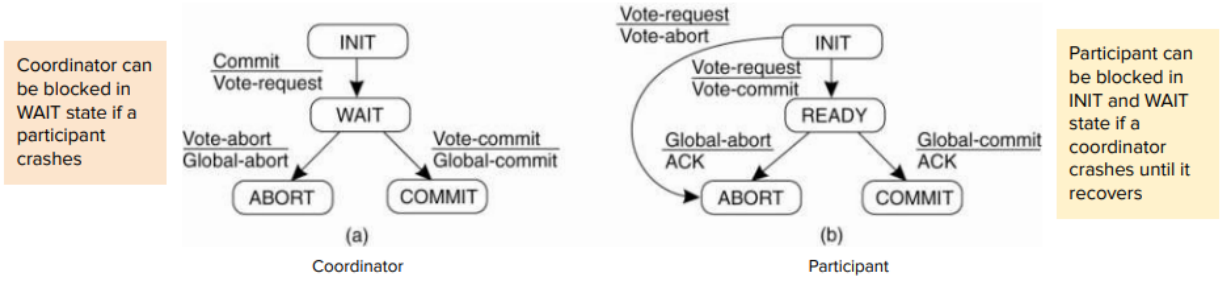

Figure 5: Two phase commit

The primary node will initiate the voting process requesting from each secondary node, whether they vote for the commit or to abort the commit. If all nodes have agreed to commit, then a global commit message will be be sent to all the participants informing that they can proceed and commit the change locally and if not a global abort message will be sent to the participants informing that they should abort the transaction

4. Sequence Diagrams

4.1 Inventory item addition - From primary server

The following sequence diagram illustrates the sequence followed when the clerk adds or updates the inventory item quantity by calling the primary server and how all the servers will be informed about the updated quantity.

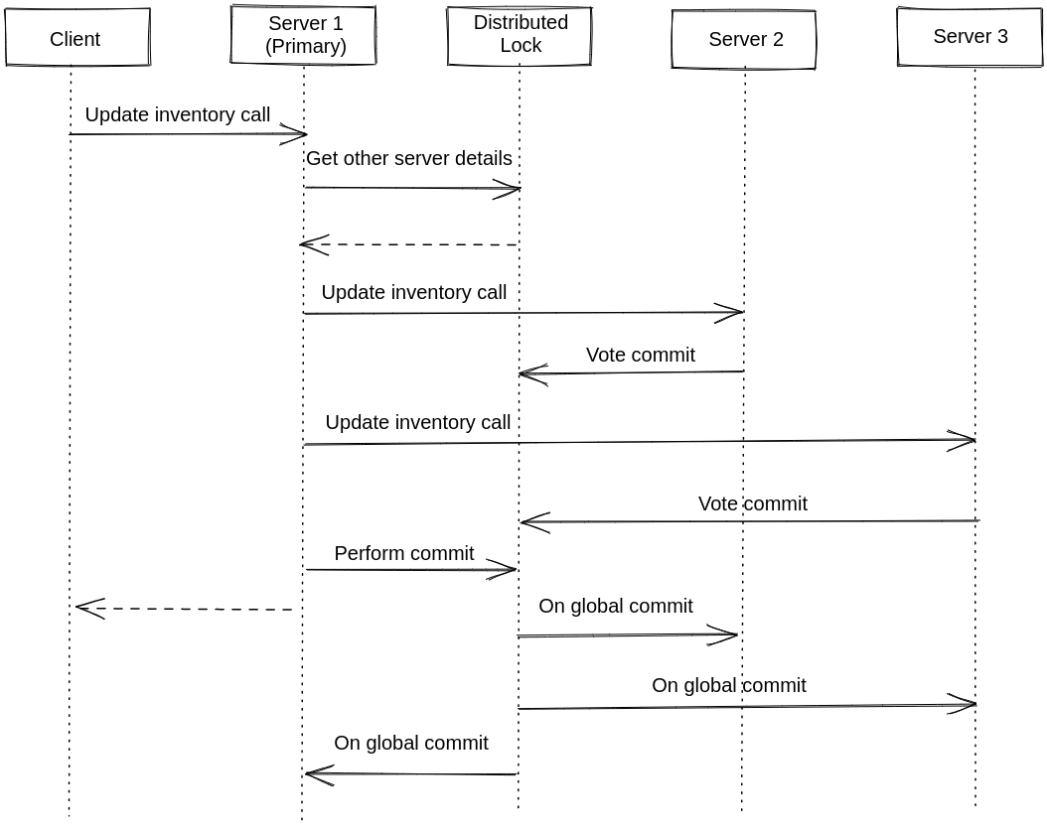

Figure 6: Inventory item addition flow - From primary server

- A primary based remote write protocol has been used within the implemented distributed system where each server maintains a copy of the available inventory item quantity
- A two phase commit approach has been used to initiate a voting process and coordinate the change among all the available servers.
- When the clerk calls the primary server through a client, the primary server will fetch the details of the secondary servers from the distributed lock

- The primary server will make the same exact gRPC call that the client made, to all the secondary servers.
- If a request was received by the secondary servers from the primary server, then the secondary servers will check whether they can fulfill the request and then vote and agree for the change to take place. If any of the secondary server cannot fulfill the request, it will send an abort message to the distributed lock
- The primary server will check whether it can fulfill the request and also whether all the secondary servers have voted towards the change. If so, the primary server will send out a global commit message notifying all the secondary servers to perform the change and if any of the secondary servers have aborted the request, then a global abort message will be sent notifying everyone to abort the request

4.2 Order placing - From Secondary server

The following sequence diagram illustrates the sequence followed when a workshop manager adds or updates the order by calling one of the secondary servers and how all the servers will be informed about the updated quantity.

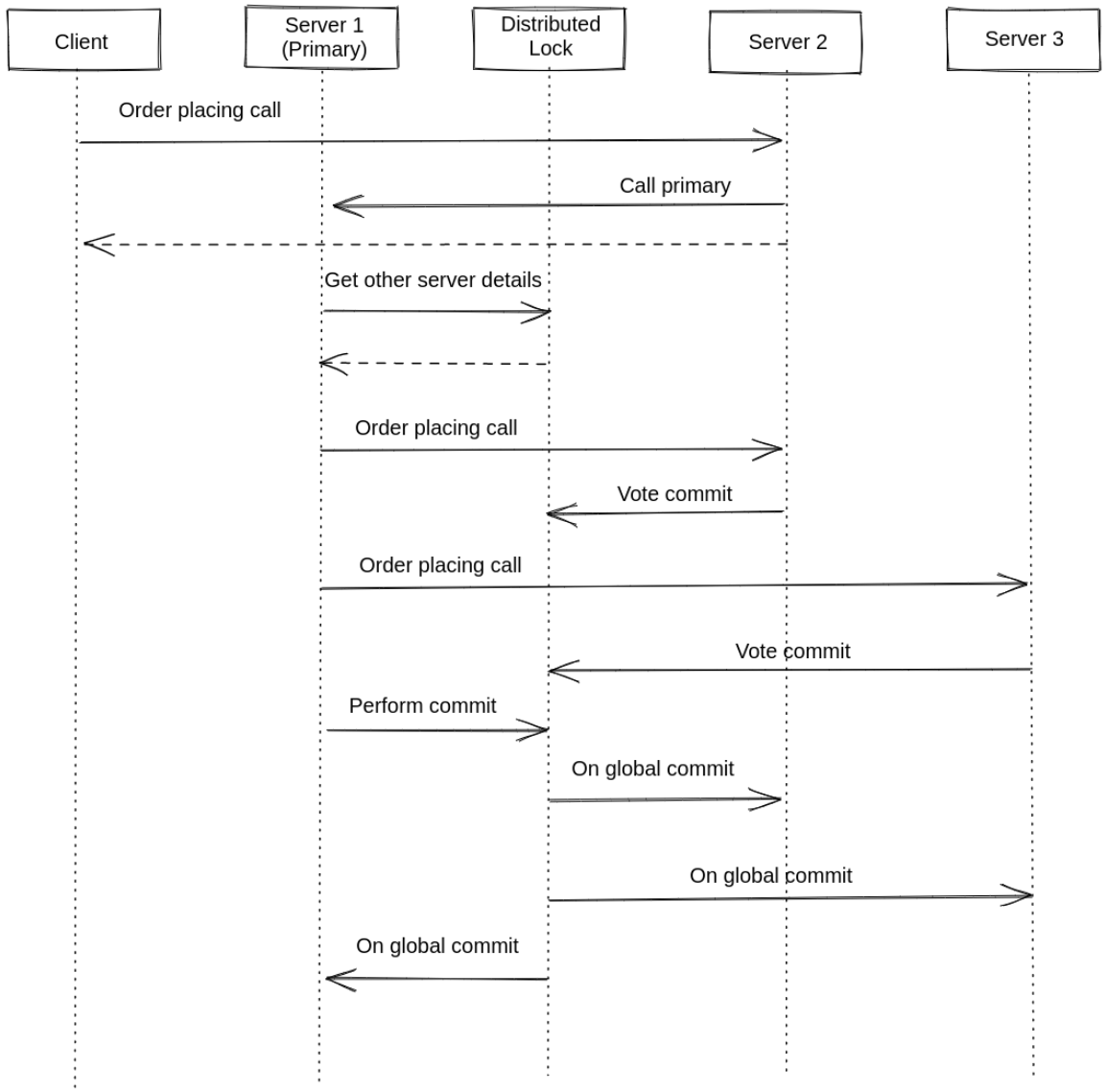

Figure 7: Order placing flow - From Secondary server

- The sequence is identical to flow of inventory addition (Refer section 4.1) but the noticeable difference being the server that gets called by the client is not the primary server.
- When a secondary server receives an order placing request from a client, it would make the same call to the primary server where the primary server will initiate a two phase commit flow and decide whether or not to proceed with the order request and notify all the servers in the distributed system

4.3 Simultaneous order placement with insufficient stock

The following sequence diagram illustrates the sequence followed when a two different order placement calls were made simultaneously and there is sufficient stock to fulfill only one of these orders

- While the primary node is processing the initial request, it will initially update itself before sending out a global commit message
- This would mean that even if there is a second order requests made to a different server, the primary node will not have sufficient stock to fulfill the order
- This would result in the primary node sending out a global abort message through the distributed lock to all servers notifying them that they should abort the request.

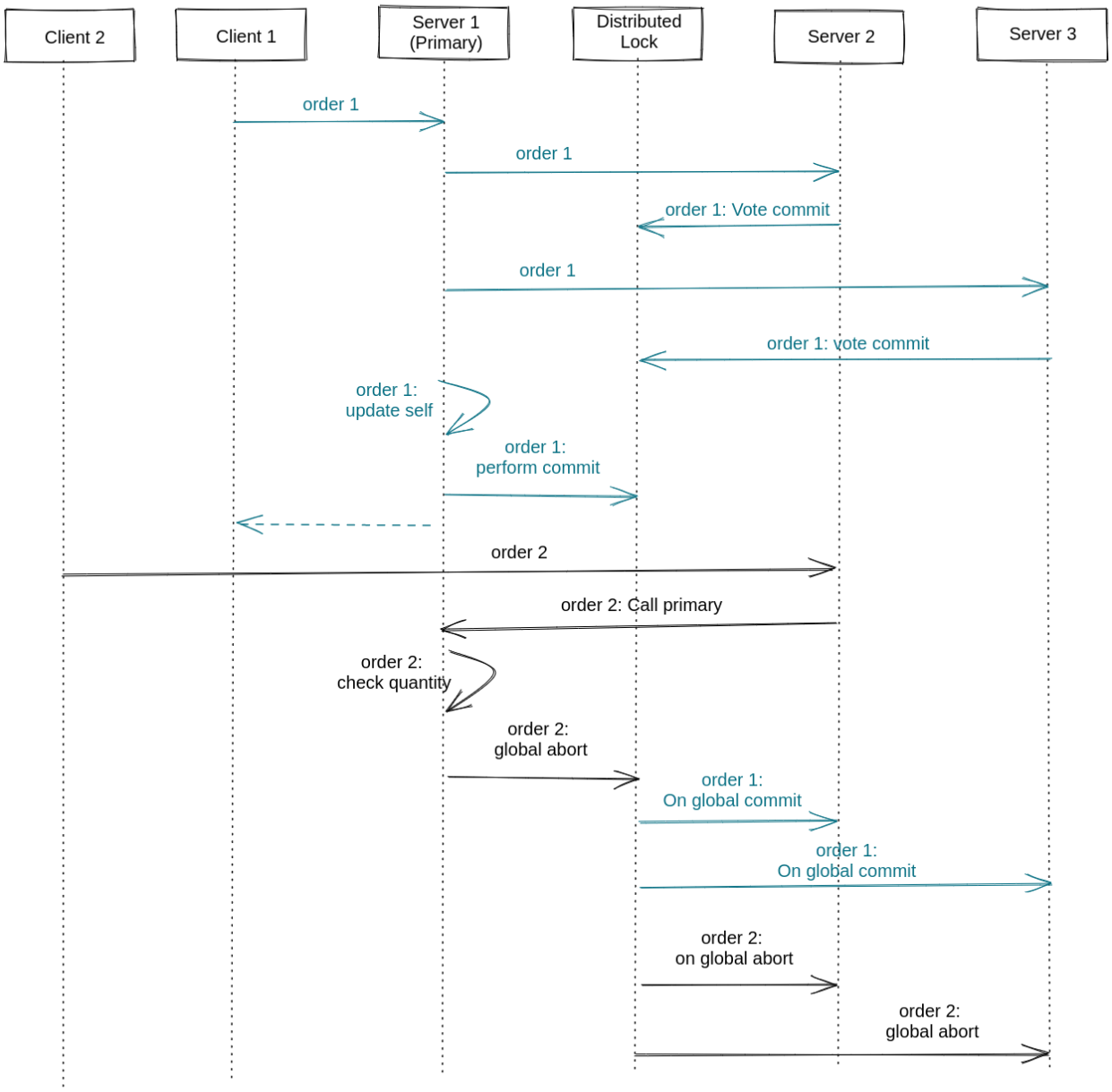

Figure 8: Simultaneous order placement flow

4.4 Node exit flow

The following sequence diagrams illustrate the sequence followed when either a primary or a secondary node exits the distributed system.

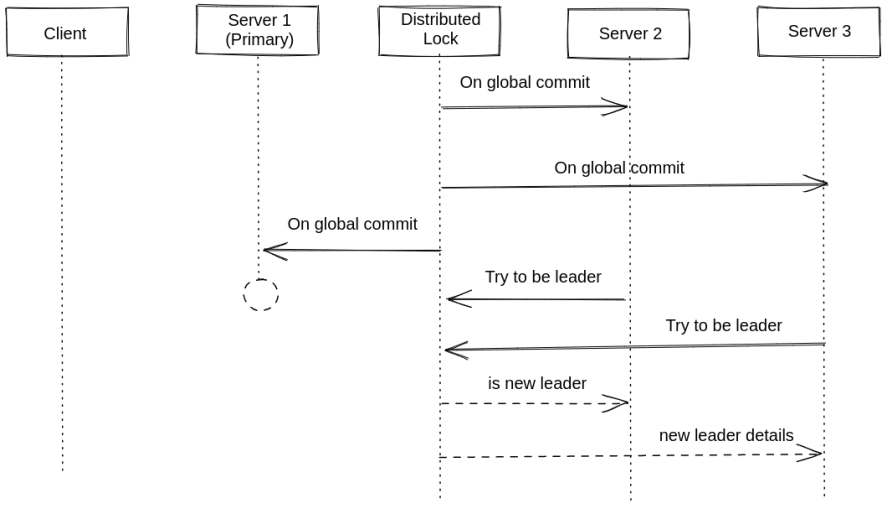

Figure 9: Primary node exit flow

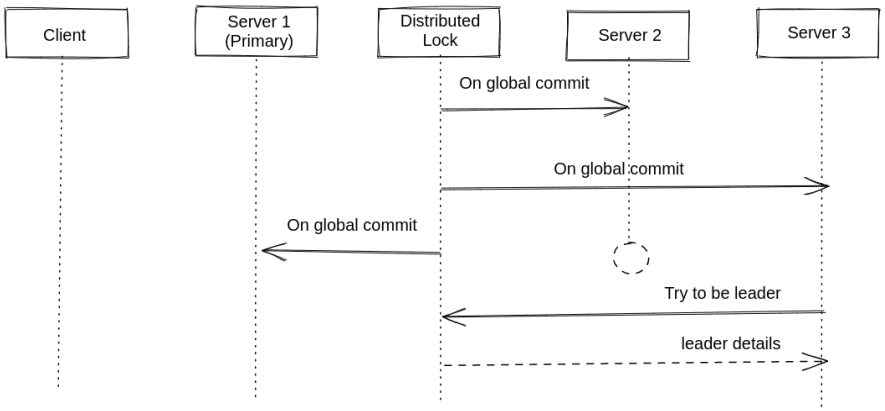

Figure 10: Secondary node exit flow

- According to section 3.3, the system was built assuming that there will always be at least one node running all the time. This is due to the fact that data stored as in-memory data will not be recoverable if all the nodes in the system end up crashing.
- The system was designed with high accuracy in mind as the system follows primary based remote write protocol and ensure that the data in every single node is accurate and even if any of the nodes end up crashing, the correct data will be maintained in the other nodes.

- Replicating data in each server will result in high availability as every one of the servers could be placed close to the client in a distributed manner, which will reduce the latency when making the network calls
- In the scenario that the primary node exits the system, one of the remaining secondary nodes will be assigned as the leader node
- The client can choose to call a different node, if a particular node has crashed and the end results will not differ.

4.5 New node join or restart flow

The following sequence diagram illustrates the sequence followed when new server node is added to the cluster

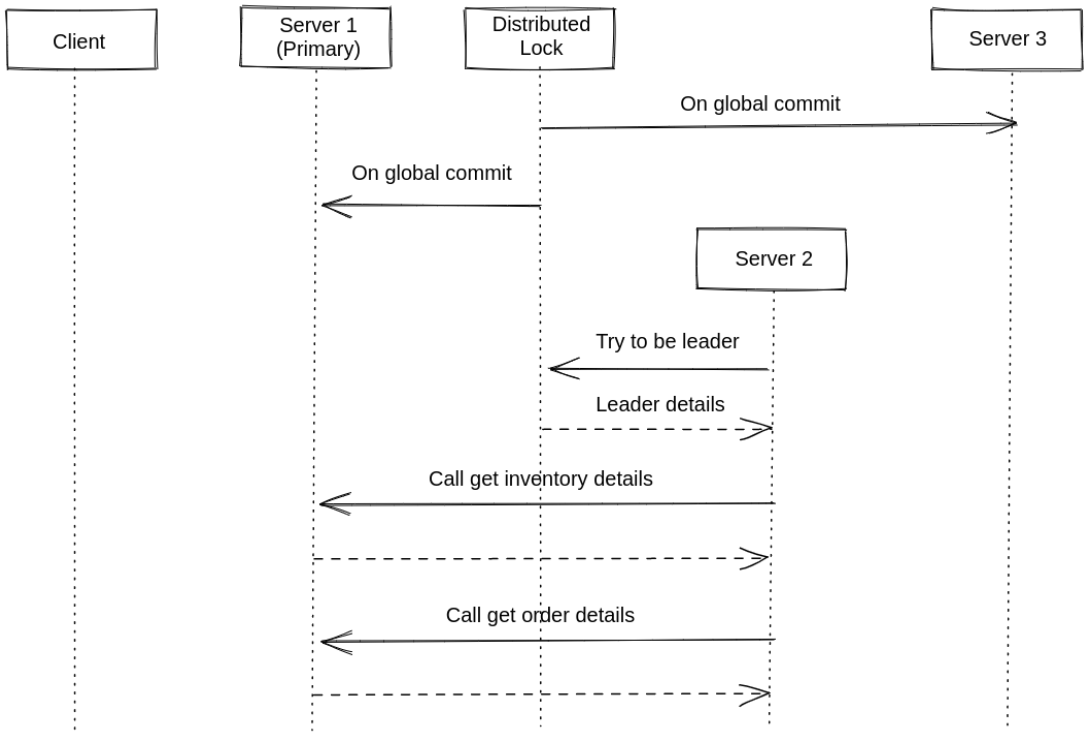

Figure 11: New node join flow

- The newly created node will try to acquire the leader lock and the distributed lock will return the details of the current leader
- The node will call the get inventory and get order details gRPC calls and update itself to have the same state as the leader node

5. Future Enhancements

- **Cloud Deployment:** The provided implementation was tested out by running multiple instances of the server within the same machine, on different ports but ideally these servers need to be deployed in different regions to handle the requests from different warehouses.
- **Edge Computing:** Many cloud service providers provide several services where the data is replicated across numerous edge locations in order to process the data within the edge location, in the hope of reducing network latency.
- **Three phase commit:** The implemented two phase committing mechanism could be further improved in the future by introducing a three phase commit process where a pre commit stage will be added in order to further improve the resiliency of distributed commits
- **Distributed Databases:** The implemented solution stores replicated data as in-memory variables in multiple servers but a better solution would be to use distributed databases such as Apache Cassandra or Apache HBase as they have robust inbuilt functionality to easily manage data across multiple database instances