CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE · INDIA

Master Slave Database Replication in IOT

By

George Rohan Mathew

(2047228)

Under the guidance of

Dr. Rajani Poonia

&

Mr. Anuj Saxena

A Project report submitted in partial fulfilment of the requirements for the award of the degree of Master of Computer Applications of CHRIST (Deemed to be University)

March - 2021

CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE · INDIA

CERTIFICATE

This is to certify that the report titled ***Master Slave Database Replication in IOT*** *is a bonafide record of work done by* ***George Rohan Mathew (2047228)*** *of CHRIST (Deemed to be University), Bengaluru, in partial fulfilment of the requirements of 4th Semester MCA during the year 2021-22.*

Head of the Department **Project Guide**

Valued-by:

	Name	: George Rohan Mathew
1.	Register Number	: 2047228
	Examination Centre	: CHRIST (Deemed to be University)
2.	Date of Exam	:

Trivium eSolutions

PRIVATE & CONFIDENTIAL

April 11th, 2022

Name: GEORGE ROHAN MATHEW
Intern No. : I023

Subject: Internship with Trivium eSolutions Pvt. Ltd.

This is to certify that GEORGE from Christ University is undergoing his internship program with Trivium eSolutions Pvt. Ltd., as an Intern between December 6th, 2022, and June 30th, 2022. His performance and competence during this internship are satisfactory.

Yours sincerely,
for **Trivium eSolutions Pvt. Ltd.**

Surbhi Anand
Manager-HR

Trivium eSolutions Pvt. Ltd. Bangalore

ACKNOWLEDGEMENTS

First of all, I thank God almighty for his immense grace and blessings showered on me at every stage of this work.

I am greatly indebted to Dr Joy Paulose, Head, Department of Computer Science, CHRIST (Deemed to be University) for providing the opportunity to take up this project as part of my curriculum. I am greatly indebted to PG Coordinator, Dr Tulasi B, for providing the opportunity to take up this project and for helping with her valuable suggestions.

I am deeply indebted to our project guide Dr Rajani Poonia, for her assistance and valuable suggestions as a guide. He made this project a reality.

I am deeply indebted to my industrial supervisor Mr Anuj Saxena and my other company mentor Mr Ashwin Baby, CEO Mr Deepak Nakra for rendering support during the project, and all my colleagues for their valuable suggestions and contributions to make this project a reality.

I express my sincere thanks to all faculty members and staff of the Department of Computer Science, CHRIST (Deemed to be University), for their valuable suggestions during the course of this project. Their critical suggestions helped us to improve the project work.

Acknowledging the efforts of everyone, their chivalrous help in the course of the project preparation and their willingness to corroborate with the work, their magnanimity through lucid technical details lead to the successful completion of my project.

I would like to express my sincere thanks to all my friends, colleagues, parents and all those who have directly or indirectly assisted me during this work.

ABSTRACT

Replication basically means to have a copy or to replicate. So in the case of Databases, we have one database which has all the data that we want to save and we will have the exact copy of that data in another database or machine. We will call another database which has the copy of the first one as a replica. So the Database which has the main data or main source of writes and updates becomes the primary database, known as Master and the database which has the copy or replication from the primary database is known as secondary, known as Slave.

In my project "**Master Slave Database Replication in IOT**", for fault tolerance, scalability, and performance, database replication is commonly employed. The failure of one database replica does not prevent the system from functioning since other replicas can take over the failed replica's functions. The load can be distributed over all replicas for scalability, and more replicas can be added if the load grows. Finally, if data copies are stored close to clients, database replication can give rapid local access even if clients are geographically spread.

Despite its benefits, replication is a difficult process to use, with numerous obstacles to overcome. The most important aspect is replica control, which ensures that data copies remain consistent when updates are made. There are numerous options for where updates can take place and when changes are propagated to data copies, as well as how changes are implemented and where the replication tool is stored. Combining replica control with transaction management presents a unique difficulty since it requires numerous activities to be handled as a single logical unit while also ensuring atomicity, consistency, isolation, and durability across the replicated system. The book categorizes replica control mechanisms, goes through numerous replica and concurrency control mechanisms in depth, and covers many of the challenges that arise when such solutions are implemented within or on top of relational database systems.

TABLE OF CONTENTS

INTRODUCTION

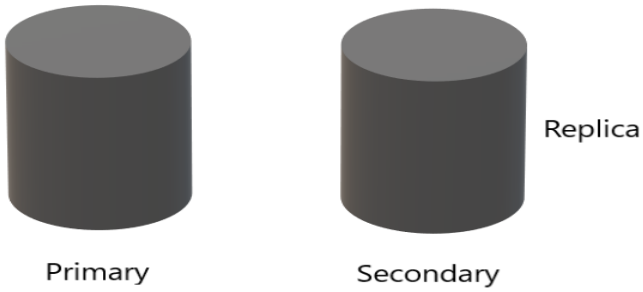

Fig 1

Replication basically means to have a copy or to replicate. So in the case of Databases, we have one database which has all the data that we want to save and we will have the exact copy of that data in another database or machine. We will call another database which has the copy of the first one as a replica. So the Database which has the main data or main source of writes and updates becomes the primary database, known as Master and the database which has the copy or replication from the primary database is known as secondary, known as Slave.

We'll take the example of an internet store selling puppets to better illustrate our point. The store's website allows customers to search the inventory of puppets available and purchase the ones they choose. The corporation maintains a database behind the web server that contains all product information, including photos, stock levels for each product, and pricing information. It also keeps track of clients that are currently or were previously involved in a transaction. So, why would our firm want to duplicate the information?

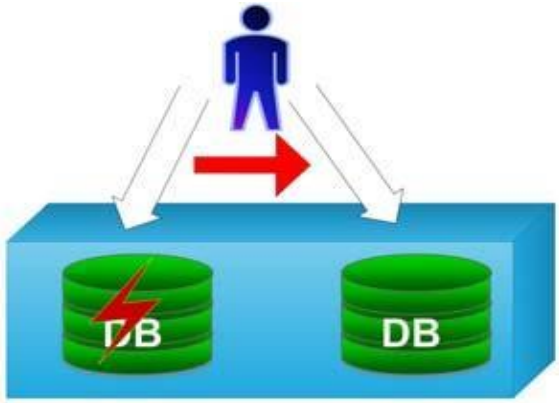

Fig 2.1

The first factor is tolerance for mistakes. Regardless of failures, our puppet firm wants to ensure that its customers have access to their system 24 hours a day, seven days a week. A failure can occur when the database software process fails, the physical machine crashes (for example, due to hardware failure), or the link between the client and server systems is (temporarily) disrupted due to a network failure. Our organization install two copies of the database on different nodes to address these failure scenarios. A node is also referred to as a replica in cases where the complete database is copied.

If one of the replicas fails, there is still one replica that is operational. The system can withstand the loss of one of the copies. Because the service stays available despite failures, this is also known as high availability. The replicas in most high-availability solutions are connected to the same local area network, as shown in Figure 2.1, to allow for quick communication between them. Any failure is detected via a failure detection mechanism, and clients connecting to the failed node are rejoined to the accessible node, where request execution resumes.

However, such a solution cannot handle network failures between clients and databases because the client will be unable to connect to any of the replicas.

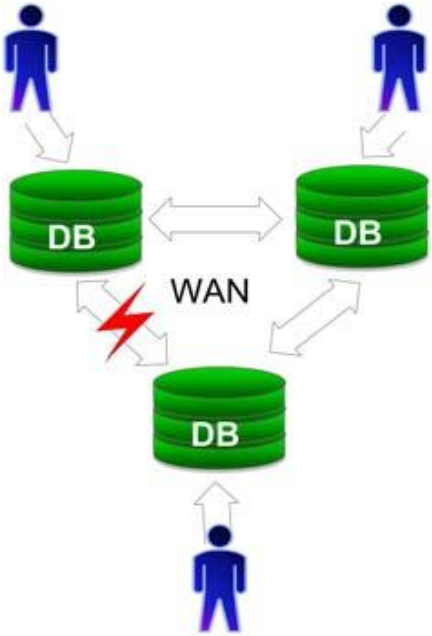

Fig 2.2

Wide area replication can be utilized to solve this problem (see Figure 2.2). The database replicas are geographically distributed in wide area replication, and each client connects to the closest replica. Even if a remote replica is unavailable, the local replica is generally accessible. However, because of the likelihood of network partitions, the latency of the wide area network poses fascinating issues, and failover becomes more complicated.

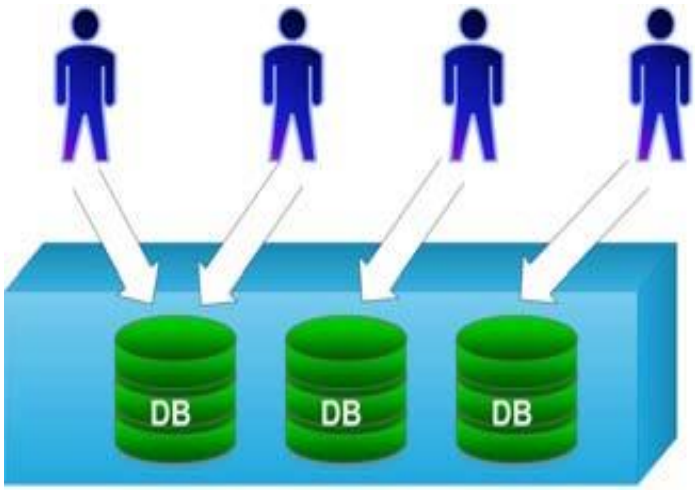

Fig 2.3

Performance is a second key application for replication, as it can help boost throughput and minimize response time. Let's return to our puppet shop. The organization can build numerous replicas within a local area network as a first option, resulting in a cluster (see Figure 2.3).

To the outside world, the cluster seems to be one unit, and requests are distributed across the copies as they arrive. The system can scale up to meet rising demand by adding new replicas. Replication fulfils the objective of scalability in this scenario because it can deliver increased throughput.

Eager Database Replication

Since Database Replication mostly faces performance issues, eager database replication was introduced. It is all about consistency and is being implemented using three key parameters. Since it is being used for both fault tolerance and performance purposes mostly there would be a compromise between the efficiency of it and the consistency. A great number of research databases are based on asynchronous

replication which is also called as lazy update model. But a lot of consistency are being compromised. And consistency is a very significant aspect as well. The consistency issues in the lazy replication are difficult to be solved.

So mostly this eager approach is also being used in synchronous replication as well. A change that is being made synchronises with all the copies that are they before the commit happens.

Eager DataBase Replication : System & Architecture

Let there be set of clients for database and $S=\{s1,s2,s3,\ldots,s_n\}$, S being a set.Full Replication of data is there on all the servers that is each server has copy of the whole of the database.The client can connect to any of the servers, s_i to execute a query. Queries basically can do any read operation. But when a write operation is taking place, it is called update transactions. The transactions that are to be done can be done with a single message or operation by operation. Say the client gets connected to the server S_i. Operation by operation basically means interactive transaction. If a client wants to do a read request as well as a write operation, then the client basically sends a request for read first and wait for the server to respond. As soon as client gets the data, they sent the write request next. Stored Procedures is basically as group of statements in sql being considered as a logical unit. Such stored procedures are called Service Request. A transaction outcome also has the corresponding results. If a transaction is not done if S_i fails, then it is upto the client to see if they should retry it or not. They can try connecting to any other server and raise the same service request.

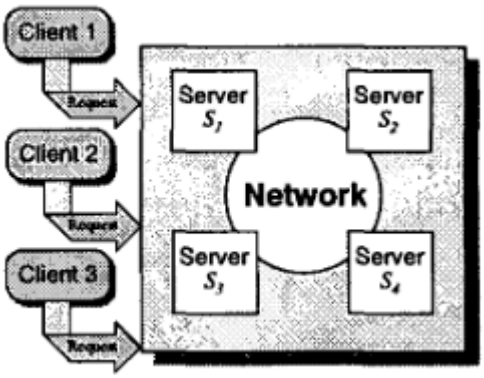

Fig 3

The servers of the database communicate with each other through 1 to n or point to point communication. Group communication primitives have two different parameters: Ordering and Reliability of message delivery.

Regarding the reliability of message delivery Reliable Broadcast and Uniform reliable broadcast takes care of it. A message which is being send the right database server or the message that is being delivered by the right database is eventually been delivered to all the correct databases by the Reliable Broadcast. Similarly, a message that is being delivered by any server is eventually send to all the correct databases (This messages can be right or failed right after the delivery) by the Uniform Reliable Broadcast.

The Total Order Broadcast takes care of the order of the messages. It can either be delivered in the same order or in the arbitrary order.

Classification Criteria

Eager replication techniques can be classified using three characteristics that characterise the protocol's nature and properties.

The server architecture (primary copy or update everywhere), how modifications or operations are propagated across servers (per operation or per transaction), and the transaction termination protocol are all aspects to consider (voting or non voting).

Classification Criteria : 1) Server Architecture

A specific site is required for the primary copy replication. Whatever changes or updates that are being made should initially go to the primary copy wherein an analysis would be done to fix the serialization order. If primary crashes by any chance, one of the other servers would take up the role of the primary. For avoiding bottlenecks, the database doesn’t usually make only one primary. It would have multiple as in multiple primary for a group of subsets belonging to a particular category.

Updates to a data item can be made from anywhere in the system with update everywhere replication. That is, updates can arrive at two separate copies of the same data item at the same time (which cannot happen with primary copy). Update everywhere techniques are more gentle when dealing with failures as a result of this attribute, as no election protocol is required to continue processing. In the same way, updating everything does not, in theory, cause performance bottlenecks. Update everything, on the other hand, may necessitate that, rather than one site doing the work (the primary copy), all sites execute the same task. If the design isn't thorough, update everywhere can have a considerably bigger impact on performance than primary copy alternatives.

Classification Criteria : 2)Server Interaction

The degree of communication among database servers during the execution of a transaction is the second criterion to examine. The quantity of network traffic created by the replication mechanism, as well as the overall overhead of processing transactions, are determined by this. This parameter is a function of the number of messages required to complete a transaction's operations (but not its termination). Furthermore, the type of primitive used to exchange these messages will influence the protocol's serialisation features.

We'll look at two scenarios:

Constant interaction refers to systems that use a certain amount of messages to synchronise the servers for a specific transaction, regardless of the number of operations in the transaction.

Typically, protocols of this category send only one message every transaction, containing all of the transaction's operations in one message. Linear interaction is a term that refers to techniques in which a database server propagates each operation of a transaction one at a time. The operations can be delivered as SQL statements or as log records that include the results of doing the operation on a certain server.

Classification Criteria : 3)Transaction Termination

The final parameter to consider is how transactions are managed, or how atomicity is ensured.

We discriminate between two scenarios:

Voting Termination: To coordinate the various copies, an additional round of messages is required at the end of the voting process. This round can be as complicated as an atomic commitment protocol (for example, the two-phase commitment protocol or as simple as a single confirmation message sent by a single site.

Non-voting termination means that sites can determine whether to commit or abort a transaction on their own. Non-voting approaches necessitate deterministic behavior from replicas. This is not, however, as limiting as it appears at first appearance. Because determinism only impacts transactions that are serialized in relation to one another, Transactions or processes that do not clash can be carried out in different orders at various locations. Because determinism only impacts transactions that are serialized in relation to one another, Transactions or processes that do not clash can be carried out in different orders at various locations.

THE FAULT TOLERANT MASTER SLAVE PROTOCOL

The Master-Slave Replication Strategy

As previously stated, the master-slave replication approach employs three distinct entities: master, slave, and client. There will be only one master at whatever given moment. There may, however, be a large number of slaves and clients. The replicated data is held by the master and slaves. Clients, on the other hand, only serve as front-ends for processes that rely on duplicated data. As a result, the processes that consume the data, or users, do not need to know where the data replicas are located.
Because entities are frequently represented as processes in implementations, we will now refer to entities as processes. On duplicated data, a user can make two types of queries: read requests and read/write requests. Only the duplicated data is read in read requests. The replicated data may be updated as a result of read/write requests. Both types of requests must be sent to a client by a user. A client must pass a request to a slave when it receives one. After then, two things can happen when the slave receives the request. In the instance of a read request, the slave fulfils the request and responds to the client who made the request.

After that, the client sends the user the response. If the slave receives a read/write request, it sends it to the master. After that, the master processes the request and transmits an update of the replicated data to each slave. This ensures that the slaves have the most recent copy of the duplicated data. The master not only sends the updates, but also responds to the slave who forwarded the request. After getting the response, the slave forwards it to the client who made the request. Finally, the client sends the user the response. There are two conditions that must be followed while handling read requests and read/write requests. The first criterion is that all updates must be applied in the same order by the master and slaves. This ensures that there are no anomalies between copies as a result of differing update orders. The second criterion is that after a user submits a read/write request, the user must send another request that operates on a version of the replicated data that incorporates the read/write request's changes.

When a user's request is causally related to another user's request, this criterion ensures that the user's request is executed on the right data. A different form of condition is that the master can only send updates to each slave if it knows where each slave is. Every newly activated slave could send an announcement message to the master to inform him of the slaves' locations. Another benefit of sending this message is that it allows the master to send a copy of the replicated data to the activated slave. Unfortunately, there is a difficulty in that the slave must somehow discover the master's address. However, we may simply accomplish this by utilising a directory service that contains the necessary information.

The Non Fault Tolerant Master Slave Protocol

Following that, we'll go over a non-fault tolerant master-slave protocol that serves as a warm-up for the fault tolerant protocol. The state diagrams in Figures 2.1, 2.2, and 2.3 are used throughout the explanation. Every sort of procedure has its own state diagram. The diagrams are essentially directed graphs with states at the vertices and state transitions at the directed edges. A start state is a state transition that does not begin in any other state. When the process is not handling any events, it enters the state named after the process in question.

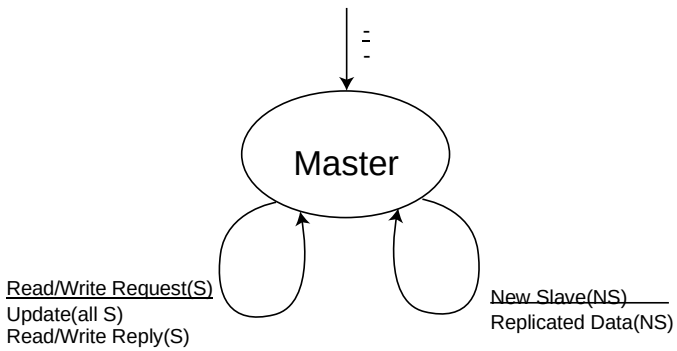

Fig 4.1

Every state and every state transition has a label in the state diagrams. A state's label makes it simple to refer to that state. A state transition's label can be used to describe two things. The label first describes the transition's reason. This could be a message from a process or a request from a user. Second, the label also specifies the process's externally evident reaction. This response could be a message sent to one or more

processes, or a user response. Both portions of a transition label are separated by a horizontal line. The cause is always on the top of the line, and the response is always on the bottom. The internal changes induced by state transitions are not reflected in the transition labels. These modifications are only mentioned in the text.

Both the text of the cause and the text of the reaction may be a single dash in the labels associated with state transitions. No external event is required for the transition to occur if the cause is a dash. A dash indicates that no external reply is seen in the case of a response. We identify one or more letters between parentheses behind every cause and response that isn't a dash. The letters above the line in a state transition label tell us who caused the occurrence. The letters below the line indicate to whom a reaction is meant. Table 2.1 shows the many letter combinations that can be used.

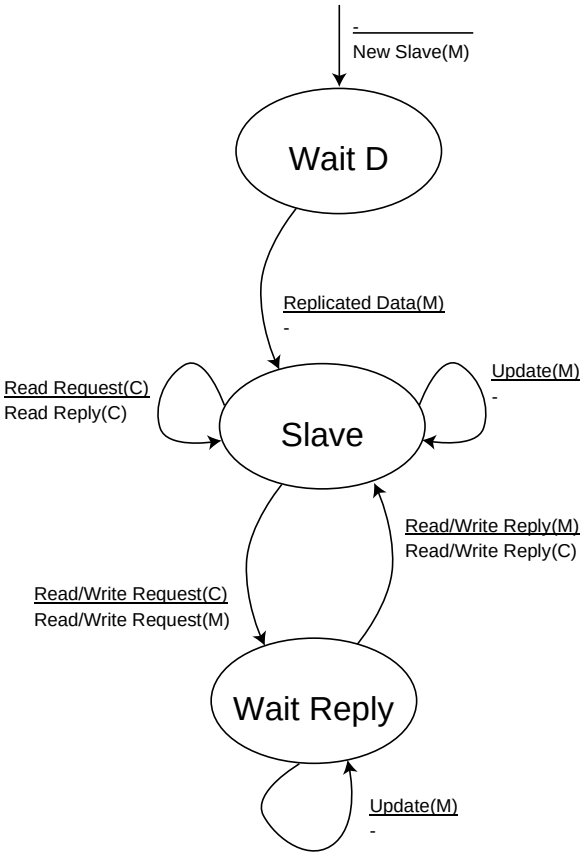

Fig 4.2

We now explain the non-fault tolerant protocol by means of scenarios that refer to the state diagrams. The scenarios revolve around the events given in the

general explanation of the master-slave replication strategy. Only two types of events are possible: activation events and requests sent by users. There are three activation events: master activation, slave activation, and client activation. The sending of requests has two possibilities: sending of a read request and sending of a read/write request.

Activation of the Master

The master's activation is the first situation we'll look at. The master enters the only state of Figure 2.1 when it becomes active. There are no acts that are apparent from the outside. When slaves use a directory service to find the master, the time of activation is, of course, a good time to register with the directory service.

Activation of Slave

Slave activation is more difficult than master activation. The first thing a slave must do is determine the master's address. It might be able to accomplish this by calling a directory service.

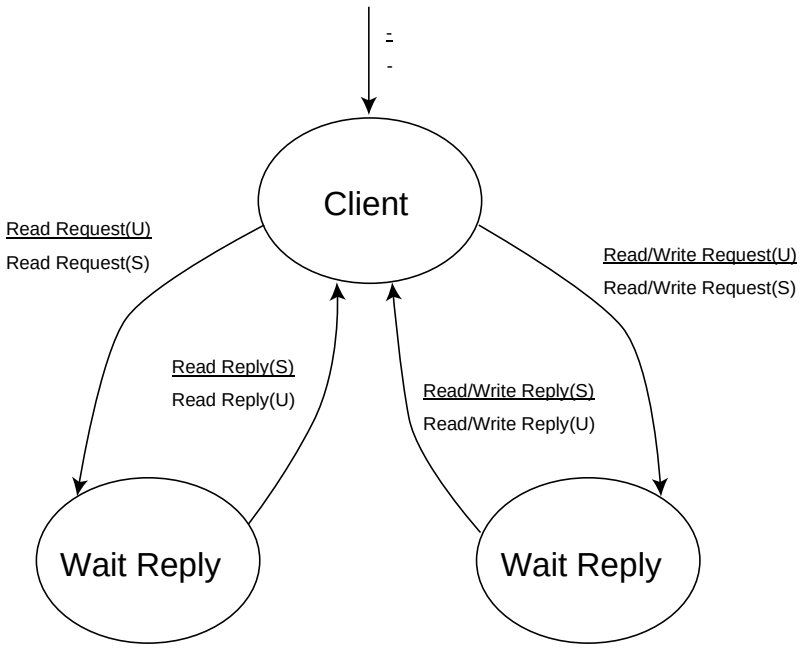

Fig 4.3

Meaning of all possible letter combinations in Figure 2.1, 2.2 and 2.3

Letter combination	Meaning
M	Master
S	Slave
NS	New slave (an activating slave)
All S	All known slaves
C	Client
U	User

After that, The slave must inform the master that it has arrived. The slave accomplishes this by allowing the slave to send the master a message containing address information. Figure 2.2 shows the New Slave notification at the top. The slave enters the Wait D state after transmitting the message and waits for a response. The master eventually receives the New Slave message. As seen in the master state diagram, the master responds by providing a copy of the duplicated data. The slave's information is likewise saved by the master so that it can send updates later. A transition to the Slave state occurs when the slave receives the replicated data. This completes the slave's activation. The slave, like the master, may need to register with a directory service. This is a good time to do it right after receiving the replicated data, because the slave is only completely functional from that point on.

Activation of Customers

During activation, a client, like the master, does not need to do any externally observable actions. A client, on the other hand, requires the address of a slave in order to function. The moment of activation is an ideal time to get such an address. Alternatively, the client can wait until a user submits a request, as it is only then that the client requires the address. When a client is activated, it enters the Client state (see Figure 2.3).

Sending a Request to Read

When a user makes a read request to a client, the client passes it on to a slave. This is only done by the client when it is in the Client state. It is busy handling a request when it is in any of the previous stages, and the newly sent request must wait. Regardless, the client sends the request to one of the slaves, who gets it. When this occurs, the slave processes the request and responds to the client. The slave, like the client, only handles requests when it is in the Slave state. After forwarding the request, the client enters the Wait Reply state on the left side of Figure 2.3 to await the response. When the client receives the response, it returns it to the user, and the transition back to the Client state occurs.

Requesting a Read/Write Access

A client transmits a sent read/write request to a slave in the same way that a read request is forwarded to a slave. Unlike a read request, however, the client waits in the right Wait Reply state in the state diagram. Furthermore, the slave passes the request on to the master. When the master receives the request, he fulfils it. Following execution, the master gives each slave an update of the replicated data. In both their Slave and Wait Reply states, the slaves can handle this update. In addition to sending updates, the master responds to the slave who sent the request. This slave is in the Wait Reply state, waiting for a response. The slave returns to the Slave state after receiving the response. In addition, the slave relays the response to the right client. That client sends a response to the user who made the request.

We've now covered every feasible possibility. We didn't go into detail about how the protocol meets the requirements for ordering updates and executing requests with the most up-to-date version of replicated data. Fortunately, the protocol can meet both needs by relying on the FIFO network assumption described in Section 2.1, allowing only one network connection between each pair of processes, enabling each user to use a unique client, and allowing each slave to use a unique client. Let's start with how we can ensure that all updates are applied in the same order on both the master and slave machines. We know that the master is in charge of all read/write requests,

which are the source of all updates. As a result, all slaves must apply all updates received in the same sequence as the master conducts the read/write requests. By providing all updates in the order of request execution, the master may quickly inform the slaves of the order. Because each slave has just one network connection and all network connections are FIFO, the network transmits all updates in the order that the master sends them.

The necessity that a request issued by a user after a read/write request sent by that same user execute with an up-to-date version of the replicated data is now discussed. Because there are two types of requests, we must evaluate two scenarios. However, in both circumstances, the request that comes after the read/write request is not treated until the read/write request's response is received. This is the result of users utilizing the same client all of the time. The client only processes one request at a time, and the FIFO assumption ensures that the requests are received in the correct sequence. We can see that in the case where both requests are read/write requests, the master must process both. The master is aware of the changes to the duplicated data made by the first request since the client sends the second read/write request only after receiving a response to the first. As a result, it can carry out the second request with the most recent version of the duplicated data.

When we consider the case when the second request is a read request, we can see that we have a situation where both the master and the slave must handle the request. What we know is that the master communicates with the slave via a single connection, which sends the read/write request. The FIFO assumption ensures that the slave knows which update is connected with the request since the master transmits the update associated with the request just before sending the reply associated with the request. Because a client always uses the same slave, the read/write and read requests must be seen by that slave. As a result, the slave knows which changes relate to the read/write request before it receives the read request since the client ensures the slave does not receive the read request until after receiving the reply to the read/write request. As a result, the slave can carry out the read request using the most recent version of the duplicated data.

It's worth noting that the procedure described above lacks the ability to shut down processes. A shutdown, on the other hand, is quite similar to a crash failure in that both stop the generation of output. We chose to postpone the addition of shutdowns to the next part because we will only address crash failures in that area.

The Fault Tolerant Master-Slave Protocol

We enhance the non-fault tolerant protocol by allowing processes to stop in addition to crash failure robustness. Every process is expendable, which is the underlying notion behind our fault tolerant protocol. What this means varies depending on the process. Being disposable in the case of the master process means that another process must be able to take over the master's job. Every slave process must be able to take over the role, we decided. Slave processes are appropriate since they have replicated data. To make the takeover totally conceivable, we decided that every slave, like the master, must have knowledge about the other slaves present.

Expendability has a different meaning for slaves. It's likely that there are numerous slaves because we're replicating for performance. This means that if a slave crashes or shuts down, other slaves can easily take over its clients. There are two issues to consider here. To stop providing updates, the master must first notice the slave that has shut down or crashed. Second, in order to achieve the greatest performance, we may need to activate a new slave. The protocol addresses the first issue by allowing the master to run a crash detection system to detect slave crashes, as well as allowing slaves to send a message to the master when they shut down. The second issue can be addressed by allowing the protocol to create new slaves dynamically. This was not mentioned in the protocol. This capability, however, should be simple to implement. When it comes to clients, expendability refers to the ability for a user to start a new client in the event of a shutdown or crash.

As might be deduced from the above description, we treat shutdowns similarly to crashes. We can accomplish this since shutdowns are extremely similar to crash failures in that they both stop output production. Because the crash detection system we suggest may be delayed, we don't treat shutdowns and crashes in the same way, as we'll see later in this section. During shutdowns, we take several steps to avoid the crash detection method.

The protocol is once again described using state diagrams and scenarios. The state diagrams are shown in Figures 2.4, 2.5, and 2.6, one for each type of process. The elements in the diagrams are the same as in the preceding section. There are, however, five new elements. To begin with, several of the state transition labels now incorporate italicized text. These italics briefly define a requirement that must be met before a process can proceed to the labelled transition. Second, an end state is now included in all state diagrams. When a process shuts down, it has reached its final state. End states are indicated by vertices with a double border. Third, a state transition can now be caused by a variety of factors. Backslashes separate the numerous causes in the labels associated with state changes. Fourth, the cause of some transitions does not include letters between parenthesis. The letters are missing because identifying a process or user from whence the event comes is impossible. More information can be found in the scenarios below. Between parenthesis, three new letter combinations appear: NM, all S - NS, and all C. The meanings of these novel letter combinations, as well as those that have already occurred, are listed in Table 2.2.

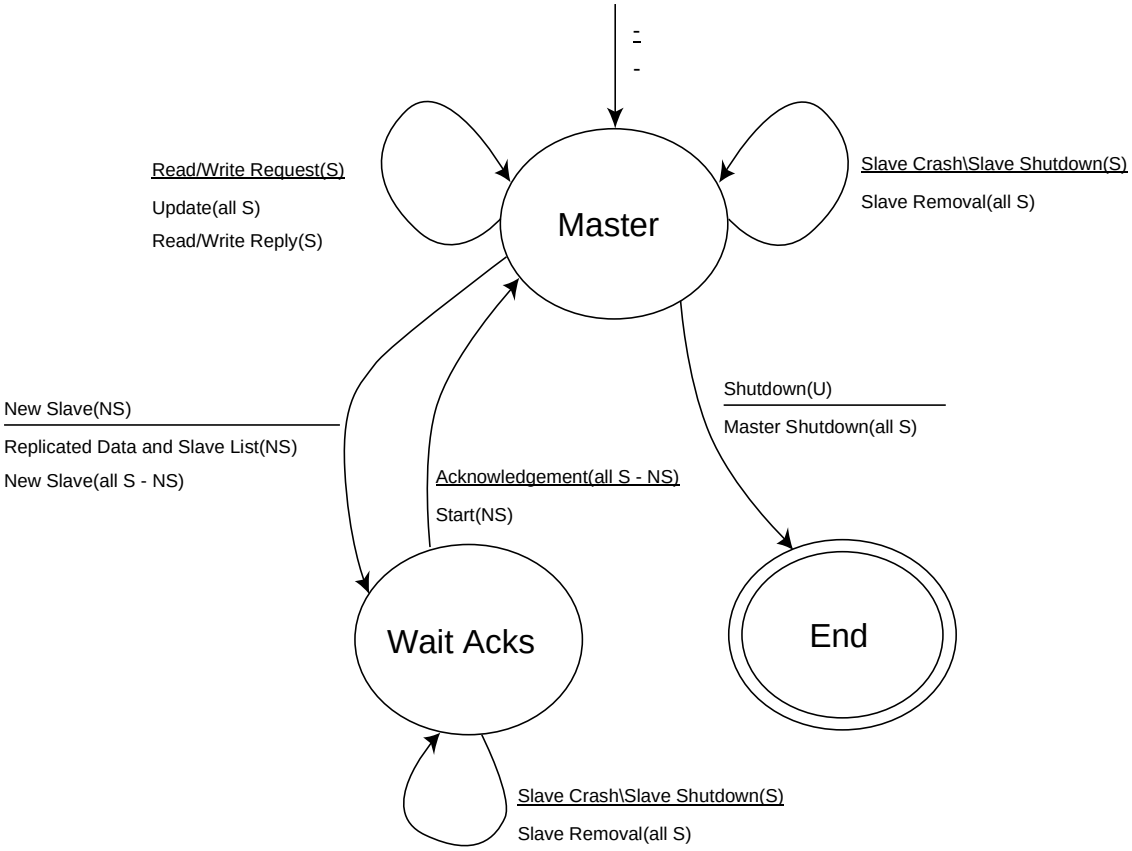

Fig 4.4

Meaning of all possible letter combinations in Figure 2.4, 2.5 and 2.6

Letter combination	**Meaning**
M	Master
NM	New master after master crash
S	Slave
NS	New slave (an activating slave)
All S	All known slaves
All S - NS	All known slaves except a new slave
C	Client
All C	All clients associated with a specific slave
U	User

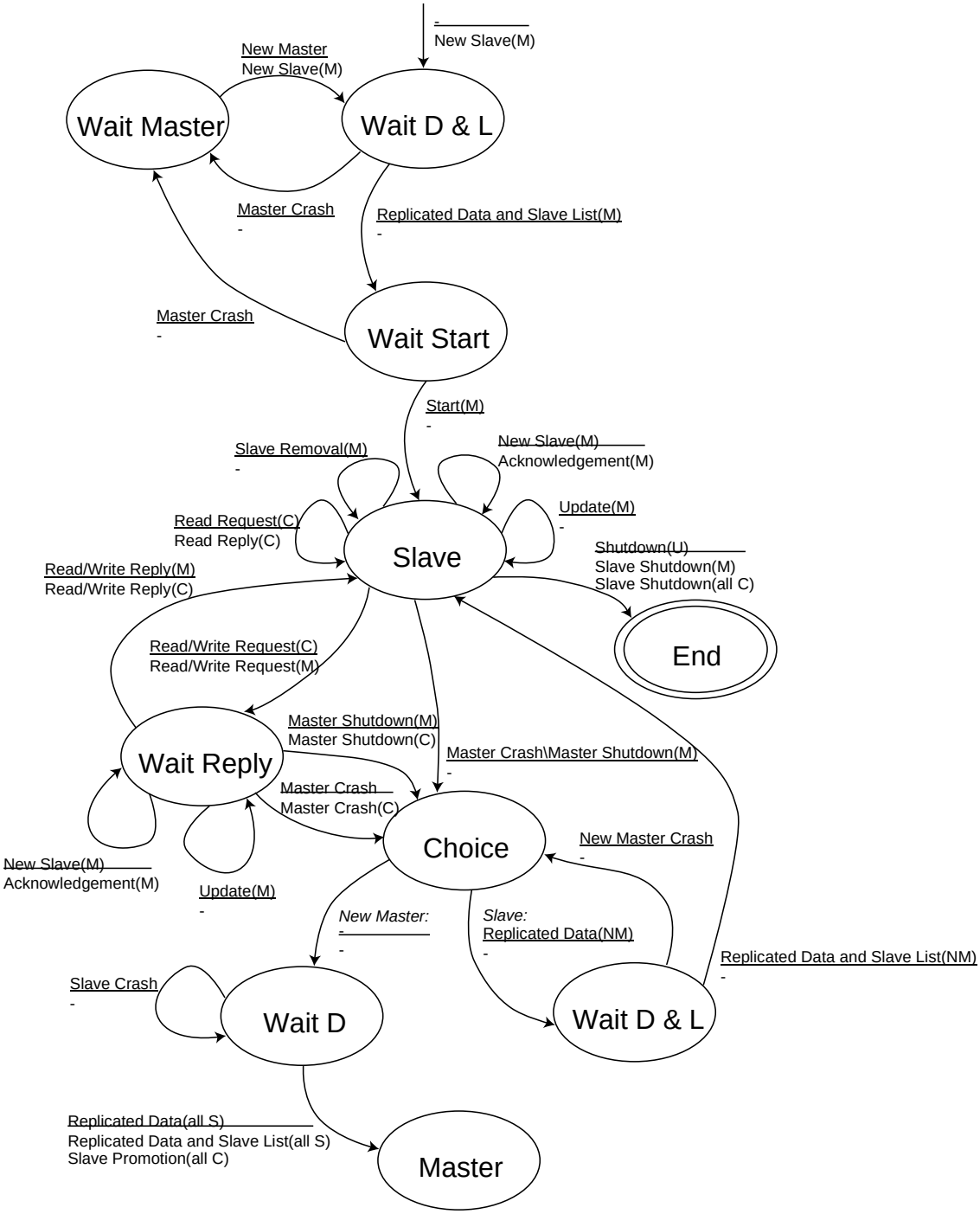
-
New Slave(M)
New Master
New Slave(M)
Wait Master
Wait D & L
Master Crash
-
Replicated Data and Slave List(M)
-
Master Crash
-
Wait Start
Start(M)
-
Slave Removal(M)
-
New Slave(M)
Acknowledgement(M)
Update(M)
-
Read Request(C)
Read Reply(C)
Slave
Shutdown(U)
Slave Shutdown(M)
Slave Shutdown(all C)
Read/Write Reply(M)
Read/Write Reply(C)
End
Read/Write Request(C)
Read/Write Request(M)
Master Shutdown(M)
Master Shutdown(C)
Wait Reply
Master Crash\Master Shutdown(M)
-
Master Crash
Master Crash(C)
Choice
New Master Crash
-
New Slave(M)
Acknowledgement(M)
Update(M)
-
New Master:
-
-
Slave:
Replicated Data(NM)
-
Replicated Data and Slave List(NM)
-
Slave Crash
-
Wait D
Wait D & L
Replicated Data(all S)
Replicated Data and Slave List(all S)
Slave Promotion(all C)
Master

Fig 4.5

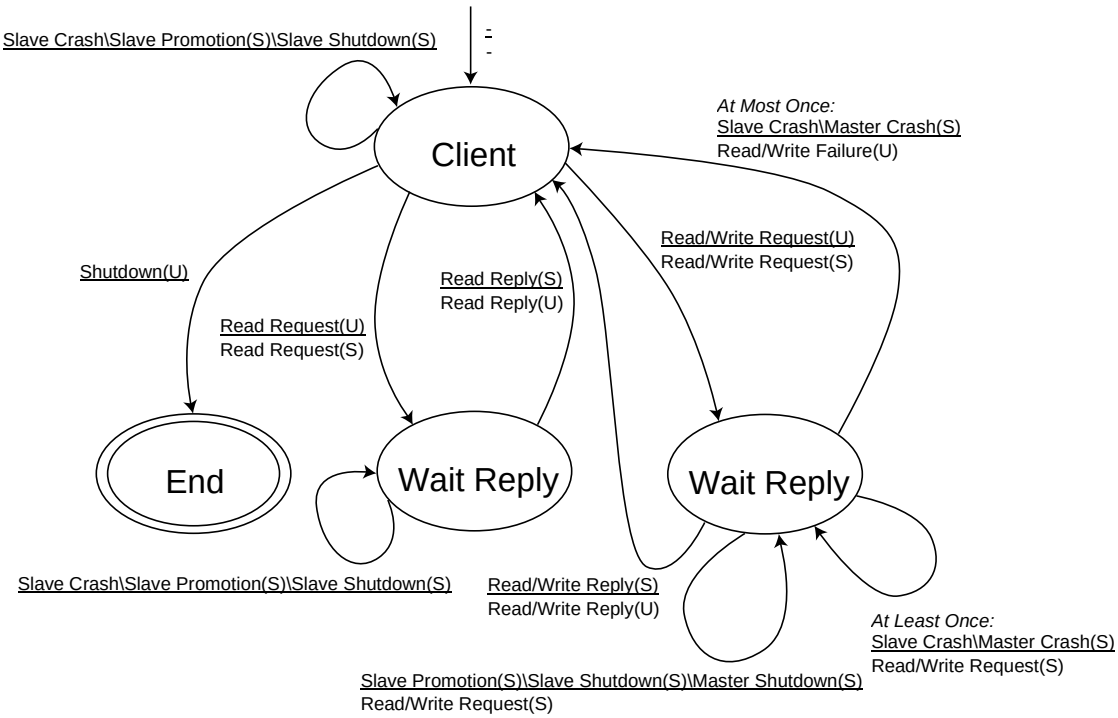

Fig 4.6

Slave Activation (Without Crashes During Activation)

In contrast to master and client activation, slave activation in the fault tolerant protocol varies from slave activation in the non-fault tolerant protocol. Slave activation is different because we want each slave to be capable of taking over the master's job. As previously stated, we do this by forcing each slave to be aware of the presence of other slaves. When a new slave is activated, this means that every slave must learn about the new slave. It also means that the newly activated slave must learn about all of the other slaves. In a data structure known as the slave list, both the master and the slaves save address information about the slaves they believe are active.

Slave activation starts with the new slave sending the master a New Slave message. This message contains information on the sender's address. The information is added to the slave list by the master when it gets the message. After that, the master sends the new slave a copy of the replicated data as well as a copy of its slave list. The slave list ensures that the new slave is aware of all other slaves. The master also informs the

other slaves about the new slave by forwarding the New Slave message to them. In the Wait D & L stage at the top of Figure 2.5, the new slave waits for replicated data and the slave list.