Failure Detection and Fault-Tolerance for Key-Value store in Distributed Systems

Project report submitted in partial fulfilment of the requirement for the degree of`

Bachelor of Technology

In

Computer Science and Engineering/Information Technology

By

Ujwal Gupta (171231)

Under the supervision of

Dr. Pradeep Kumar Gupta

to

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

Candidate's Declaration

I hereby declare that the work presented in this report entitled "**Failure Detection and Fault Tolerance for Key-Value Store in Distributed Systems"** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Information Technology** submitted in the Department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from January 2021 to May 2021 under the supervision of **Dr. Pradeep Kumar Gupta.**

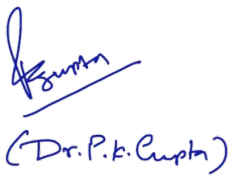
The work done embodied in the report has not been appeased for the award of any other degree or diploma.

Student signature

Ujwal Gupta

(171231)

This is to certify that the above affirmation made by the candidate is true to the best of my knowledge.



Dr. Pradeep Kumar Gupta

Associate Professor

Department of Computer Science and Engineering and Information Technology

Dated:

Acknowledgement

I would like to take the opportunity to thank and express my deep sense of gratitude to my mentor and project guide **Dr. Pradeep Kumar Gupta for** his immense support and valuable guidance without which it would not have been possible to reach at this stage of our final year project.

I am grateful to Prof. **Dr. Samir Dev Gupta**, **Head** of Dept. of CSE and IT for his excellent support during our work. We would also like to thank all the professors & other supporting members of the department of Computer Science & Engineering and Information Technology for their generous help in various ways for the completion of this work.

I am also obliged to all my faculty members for their valuable support in their respective fields which helped me in reaching at this stage of my project.

Date:

Table of Content

1. Introduction:

1.1 Introduction
1.2 Problem Statement
1.3 Objectives
1.4 Methodology

2. Literature Survey:

2.1 Fault Tolerance in Real Time Distributed Systems
2.2 Fault Resolution Mechanisms
2.3 Key-Value Store

3. System Development:

3.1 Building a distributed key-value store
3.2 System Architecture
3.3 Classes/Entities used
3.4 Storage replication and stabilization

4. Performance Analysis

Conclusion

References

List of Figures:

Figure 1. SWIM Membership Protocol

Figure 2. Trade-off between Consistency, Availability and Partition Tolerance

Figure 3. Types of Faults

Figure 4. Types of Check Pointing

Figure 5. Distributed System

Figure 6. Replica Management

Figure 7. Connectivity between clients and replicas in key-value store

Figure 8. System Layers

List of Tables:

Table 1: Key-Value Store example.

Table 2: Membership protocol test results

Abstract

With organisations going global and people becoming global citizens, services running seamlessly everytime and available everywhere on the globe has become crucial. Organisations need to access data at high speeds even if anything goes down at the service centers. This is achieved by services that run not on centralised but distributed systems which are fault tolerant. Non-availability and/or inconsistency of resources can result in losses to businesses, delay in hospitality, healthcare, education etc.

A distributed system is made up of many independent processing components that communicate with each other through an interlinked communication link system composed of communication components. Distributed computation refers to the algorithmic controlling of the distributed system's computing components by use of a distributed program in order to achieve a common objective, that is, to provide a specified service. Unfortunately, virtually all system's components are inherently flawed and therefore vulnerable to errors that can make the system unable to deliver the service.

KEYWORDS:

Key-Value Store, failure detection, fault-tolerant systems, distributed systems, consistency, availability.

Chapter 1 Introduction

1.1 Introduction

A distributed system is made up of several nodes that are physically distinct but are connected via a network. All of the nodes in this system interact with one another and work together to complete tasks. A tiny portion of the distributed operating system software is stored on each of these nodes.

Fault tolerance means a system's capacity ie: computer, network, cloud cluster, etc to continue to run smoothly when one or more of its components happen to fail without interruption. The aim of designing a fault-tolerant system is to avoid failures resulting from a single point of malfunction, to ensure that mission-critical software or services remain highly accessible and business continuity.

Availability is generally taken as the percentage of time some application and its services are available, given a specific time interval. High efficiency refers to the capacity of a system to eliminate service interruption by minimizing downtime. It's expressed as a percentage of overall operating time in terms of the uptime of a machine. Five nines is called the "holy grail" of availability, or 99.999 percent uptime. In most situations, a business continuity approach would require both high availability and error tolerance to ensure that during minor faults, and in the event of a catastrophe, your company retains critical operations.

While both fault tolerance and high availability refer to a system’s functionality over time, there are differences that highlight their individual importance in your business continuity planning.

Some important considerations when creating fault tolerant and high availability systems in an organizational setting include:

- Downtime – There is a minimum permissible degree of service disruption for a highly accessible device. For example, for approximately 5 minutes each year a machine of "five nines" availability is down. It is assumed that a fault-tolerant device can run continuously without suitable service interruptions.

- Scope – High usability draws on a similar collection of tools used together to handle failures and reduce downtime. Fault tolerance depends on backups of power sources, as well as hardware or applications that can detect faults and turn to redundant parts immediately.

- Cost – As it needs the continual operation and repair of additional redundant parts, a fault tolerant system may be expensive. High availability usually arrives from a service provider as part of an overall bundle (e.g., load balancer provider).

A key-value store, sometimes known as a key-value database, is a basic database that employs an associative array as its core data model, with each key corresponding to one and only one item in a collection. A key-value pair is the name for this type of connection. The key in each key-value pair is a string, such as a filename, URI, or hash. Any type of data, such as a picture, a user preference file, or a document, can be used as the value. The value is saved as a blob, which eliminates the need for any data modelling or schema design up front. In general, there is no query language for keyvalue databases. They let you to save, retrieve, and update data using simple get, put, and delete operations. Retrieving data is as simple as sending a direct request to the object in memory or on disc. The simplicity of this model makes a key-value store fast, easy to use, scalable, portable and flexible.

1.2 Problem Statement

Data today is consumed globally simultaneously, accessed by people on different continents from different replicas of the data. It is necessary that any changes made to the data on the master server is replicated across its replicas as soon as possible. Also, availability of data at all times in spite of any failures in the system is needed.

We, here will be trying to provide solution to the problems at a small scale by creating a simulation of interconnected, independent, asynchronous nodes.

1.3 Objectives

This project is about implementing a membership protocol. Because running a thousand cluster nodes (peers) over a real network is impossible, the membership protocol will be implemented above EmulNet in a peer-to-peer (P2P) layer, but below an App layer. Consider it a three-layer protocol stack, with Application, P2P, and EmulNet serving as the three levels (from top to bottom). More details are below.

Your protocol must satisfy: i) Completeness at all the time: every process that is non-faulty must detect every node that joins, fails, and leaves, and ii) When there are no communication losses and message delays are minimal, failure detection accuracy is high. When there are communication losses, completeness and accuracy are both required. It must do all of these even if several failures occur at the same time.

You can use any of the membership protocols, including all-to-all heartbeating, gossip-style heartbeating, and SWIM-style membership. Either gossip or SWIM is a good choice since it is the most effective method to learn. Further, we will be implementing:

1. A key-value store supporting **CRUD operations** (Create, Read, Update, Delete).
2. **Load-balancing** (via a consistent hashing ring to hash both servers and keys).
3. **Fault-tolerance** up to two failures (by replicating each key three times to three successive nodes in the ring, starting from the first node at or to the clockwise of the hashed key).
4. **Quorum consistency level** for both reads and writes (at least two replicas).
5. **Stabilization** after failure (recreate three replicas after failure).

1.4 Methodology:

Membership Protocols:

A group membership protocol allows processes in a distributed system to decide on a set of presently operating processes. Most distributed systems use membership protocols, which have shown to be essential for ensuring availability as well as consistency in distributed applications.

The SWIM process group stands for Scalable Weakly-consistent Infection-style Process. In a distributed system, the membership protocol can be used to keep track of which processes are members. A membership protocol provides each process with in group with a locally stored list of all other non-faulty processes in the group, known as a membership list.

The protocol, hence, carries out two important tasks -

- detecting failures, means which process have failed and
- disseminating information, meaning. how to notify all other processes in the system

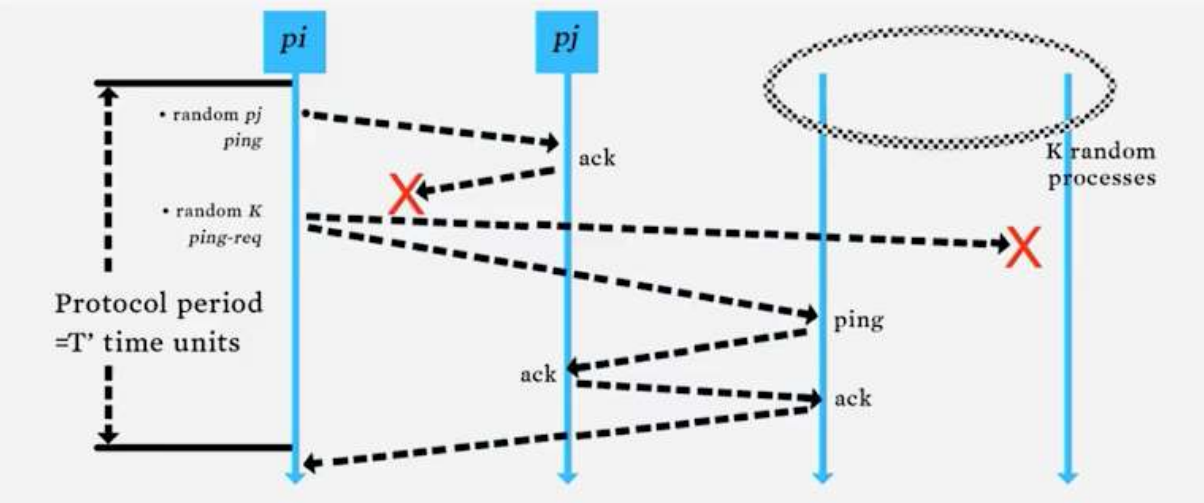


Figure 1. SWIM Membership Protocol

about the occuring failures.

Earlier to SWIM, most membership protocols utilised a technique known as heartbeats, in which each node sends a HEARTBEAT message to all other nodes in the network on a regular basis, signalling that it is alive. The peer who should have delivered the heartbeat is believed to be dead if a node does not get a heartbeat for a predetermined period of time. This approach ensures that any defective node will be identified by any of the other nodes.

While heartbeats work good enough for small networks, they have difficulty scaling due to network load: Since each node sends a clear message to each and every node, the total number of messages sent in each interval for a network of size N is O(N2). When N reaches tens of thousands of nodes, this seems to be clearly unworkable.

SWIM uses a distinct method, separating the failure detection and membership update distribution operations. In other words, SWIM illustrates that by removing the necessity for failure detection, the network load may be significantly reduced.

T, the protocol period time, and k, the size of the failure detection subgroup, are the two variables used by SWIM. Each period, each node chooses one peer at random and sends a PING message to it (assuming for the time being that a node already is aware of its peers). When a node gets a PING message, it reacts with an ACK message to the sender, confirming that it is in good health.

Quorum Consensus Protocol:

In distributed database systems, this is one of the distributed lock manager-based concurrency management protocols. This is how it works:;

1. This methodology applies a weight to each site that has a replica.
2. The protocol allocates a read quorum Qr and a write quorum Qw to each data item. Qr and Qw are two values in this case (sum of weights of some sites). And these two integers are picked based on a combination of the following criteria:

- $Q_r + Q_w > S$ – this rule avoids ***read-write*** conflict. (meaning two transactions cannot read and write concurrently)
- $2 * Q_w > S$ – this rule avoids ***write-write*** conflict. (meaning two transactions cannot write concurrently)

Here, S is the net total weightage for all sites among which the data item is replicated.

The development of a nonblocking quorum technique for replica control that ensures one-copy serializability. The benefits of a nonblocking protocol are examined, and it is demonstrated that under certain situations, the benefits can be significant. It is shown that the protocol needs be combined with a dissemination technique in order to be helpful. It is also true that the access latency can be reduced significantly in a replicated environment. The quorum protocol has an unusual feature: it effectively employs a read quorom/write quorom method for concurrency management but a read-one/write-all method for replica management.A

CAP Theorem:

The CAP Theorem states that a distributed database system can have just two out of three characteristics: consistency, availability, and partition tolerance. In the Big Data era, the CAP Theorem is crucial, notably whenever we need to make trade-offs between the three depending on our specific use case.

Partition Tolerance:

This situation indicates that the system continues to function despite the network delaying a large amount of communications between nodes. A partition-tolerant system may withstand any level of network failure without causing the entire network to crash. Records are appropriately duplicated across a variety of nodes and networks to ensure that the system remains operational during interruptions. Partition Tolerance is not an option when working with contemporary distributed systems. It's a must. As a result, we must choose between consistency and availability.

High Consistency:

All nodes view the same data at the same time, according to this criterion. Simply said, when you execute a read operation, the value of the most recent write operation is returned, resulting in all nodes returning the identical data. If a transaction begins with the system in a consistent state and concludes with the system in a consistent state, the system is considered consistent. In this approach, a system can (and does) become inconsistent during a transaction, but if an error occurs at any time during the period, the entire transaction is rolled back.

High Availability:

This condition says that each request receives a success/failure answer. In order to achieve availability in a distributed system, the system must be functioning at all times. Regardless of the status of any one node in the system, every client receives a reply. This statistic is simple to assess: you can either send read/write instructions or you can't. As a result, the database are time-independent, as the nodes must be available at all times.

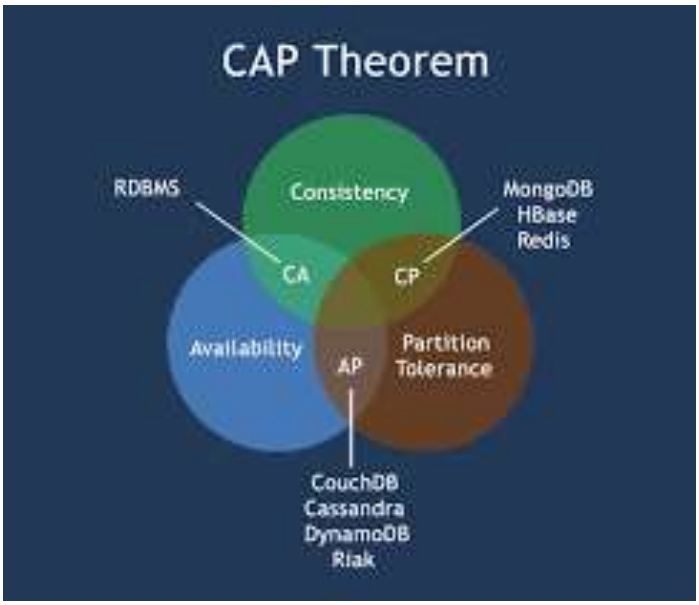


Figure 2. Trade-off between Consistency, Availability and Partition Tolerance

Horizontal scaling necessitates an understanding of the complexity involved in distributed systems, making suitable trade-offs for both the task at hand (CAP), and picking the proper right tools for the job.

Chapter 2
Literature Review

2.1 Fault Tolerance in Real Time Distributed Systems:

Types of Faults: There are various categories of faults in the Real-Time Distributed System that may occur.

Network fault: A network fault can occur due to network partition, packet loss, packet corruption, destination failure, connection failure, and other factors.

Physical faults: Such fault may occur in hardware such as CPU fault, memory fault, storage fault, etc.

Media faults: This fault may occur because of media head crashes.
CPU faults: processor faults are triggered by processor faults due to the operating system crashes.

In the view of computational power and time, the failure can be categorised. During computing, a fault happens on device resources that may be defined as: omission failure, time failure, reaction failure, crash failure. In terms of time, fault occurs in Figure 1 as:

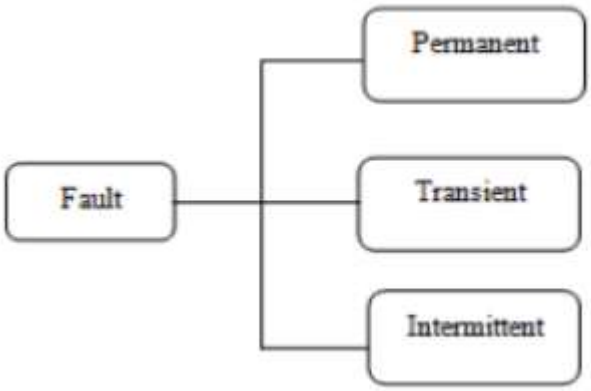


Figure 3. Types of Faults

Permanent: These faults are caused by unintentionally breaking a cable, power failures, etc. This errors can be quickly repeated. Such failures may cause big disturbances which may not be pleasant for any portion of the system.

Intermittent: These are the rare errors that occur. These faults are often missed when the system is being checked and occur only after the system goes into service. The amount of harm these failures will cause to the system is also difficult to estimate.

Transient: Any intrinsic flaw in the mechanism causes these errors. These errors, however, are resolved by retrying the device back to an earlier state/phase, like restarting the program or resending the letter. Those errors are commonly found in the operating systems.

Fundamental spotlight is on equipment adaptation to non-critical failure progressively conveyed framework. Programming adaptation to non-critical failure is frequently disregarded. This is truly astounding on the grounds that equipment parts have a lot higher dependability than the product that runs over them. Most framework fashioners put everything on the line to restrict the effect of an equipment disappointment on framework execution. Anyway they give little consideration to the frameworks conduct when a product module comes up short. There are various strategies for programming adaptation to internal failure (for example break, reviews, task rollback, exemption dealing with, and casting a ballot). Most Real time frameworks must capacity with high accessibility significantly under equipment deficiency conditions. The most helpful equipment adaptation to internal failure strategies are repetition and burden sharing. For enduring any shortcoming from the framework first we need to distinguish the issue happened in the framework and afterward disengaging it to the suitable unit as fast as could reasonably be expected. The fundamental recognition systems are: Sanity Monitoring, Watchdog Monitoring, Protocol Faults, In-administration Diagnostics, and Transient Leaky Bucket Counters. If a unit is genuinely defective, a large number of shortfall triggers will be created for it. The main objective of problem disconnection is to link the failure triggers and identify the faulty item.

Issues:

In a disseminated continuous framework or all in all, adaptation to internal failure is the procedure to give the necessary administrations within the sight of issue or mistake inside the

framework. The point is to stay away from disappointments within the sight of shortcomings and offer types of assistance according to necessity. In adaptation to non-critical failure the issue is recognized first and recuperates them without cooperation of any outer specialists.[1]

The fundamental issue in adaptation to internal failure is, where, and which procedure is utilizing to endure shortcoming in appropriated framework. As we have seen many sort of flaw and disappointment emerges in a framework, so there should be a suitable technique which can endure such issue. In this paper will we will see different strategy for enduring diverse deficiency. In any constant disseminated framework there are three primary issues.

1. Possibility this implies that an assignment running should be done on its cut-off time despite the fact that there is a deficiency in the framework. Dead line continuously framework is the significant issue in light of the fact that there is no importance of such an errand which isn't completing before its cut-off time. So the inquiry is what technique is to be applied by which the errand can complete on cut-off time within the sight of issue.

2. Unwavering quality progressively appropriated framework dependability implies accessibility of start to finish administrations and the capacity to encounter disappointments or efficient assaults, without affecting clients or activities.

3. Adaptability refers to a framework's capacity to deal with changing workloads and its ability to construct complete throughput under increased load as assets are added. Currently, the question is how these flaws may be discovered, removed, or tolerated in varied climates. An assignment running in appropriated climate should be done on its cut-off time. It very well might be hard or delicate rely upon task necessity. In hard cut-off time an assignment should be done by its cut-off time strongly however in delicate cut-off time errand can completed close by its cut-off time.

Failed System Behaviour:

What occurs if a framework fizzled? What are the potential effects of a framework disappointment? A framework can carry on after disappointment in three different ways, for example,

- Fail Stop System

- Byzantine framework
- Fail-quick framework.[2]

When a framework petered in a bomb stop framework, there really is no yield. It immediately ceases transmitting any message or event, and it also does not respond to any messages accepted by the organisation. Any failure in a framework in a bomb-proof manner might result in a long-term problem. Byzantine frameworks are ones that do not cease after failure but nevertheless produce incorrect results. These frameworks continue to function despite the lack of desired results. They may communicate the incorrect yield or react to transitory flaws later. For a long time, the Fail-quick framework behaves like a Byzantine framework, but after a short period of time, it switches to a fall flat halt mode. It does not make a difference what sort of flaw or disappointment has caused this conduct however it is essential that the framework doesn't play out any activity whenever it is fizzled.

Failure Detection:

[3] Disappointment identification is the primary issue in any framework. Choosing a trustworthy disappointment indicator is troublesome errand. For identifying a flaw precisely, a solid issue identifier is required. It tends to be eliminated by applying fitting eliminating methods. Unwavering quality of flaw locator and deficiency recuperation strategy is relying on the sort of issue. For eliminating a deficiency/disappointment from the framework it must be identified first and afterward adaptation to non-critical failure procedure can be applied. Numerous disappointment location have been clarified in different papers as an autonomous help. Numerous specialists have given issue indicator for dispersed framework too. A disappointment identifier needs to give great nature of administration (QoS) yet it is up until this point. . Several ideas have been implemented, although none of those have been successful in resolving the issue. A disappointment recognition system must adapt to changing business conditions and application requirements. The Heartbeat component is used to implement the disappointment location administration. The heartbeat components are as per the following: Centralized, Virtual ring based, All-to-all, and Heartbeat gatherings.

Approaches of Fault Tolerance:

There are numerous methodologies for adaptation to non-critical failure continuously circulated framework. An issue can be endured based on its conduct or the method of event. Following are the techniques for adaptation to internal failure in a framework.

1. Replication
    a. Occupation Replication
    b. Part Replication
    c. Information Replication
2. Registration
3. Planning/Redundancy
    a. Space Scheduling
    b. Time Scheduling
    c. Cross breed Redundancy

1. Replication:

Replication is a method of exchanging data that ensures consistency across surplus assets (such as programming or equipment components) in order to increase reliability, adaptability to internal failure, and availability. Occupation replication is the strategy for imitating position on numerous worker, for example, in lattice processing administration is fit for accepting positions, performing them, carrying out checksum procedure, and transfer the outcome to the customer shows an appropriated network wherein each worker S can impart to one another and every worker is associate with different customers C. A job can be sent to employees for action, with the end result being returned to the client. Information Replication is also commonly used by adaptation to intrinsic failure mechanisms to improve accessibility in Grid-like environments where disappointments are inevitable.

2. Check Pointing:

Check-pointing This is the process of preventing a task from being completed completely. It runs the acceptance test, and if it fails, it returns to the last checkpoint rather than starting over. Based on its features, a check point might be system, application, or mixed level. As illustrated in Figure 4, check-pointing can also be classified as in-transit or orphan messages**.** Uncoordinated Checkpointing, Coordinated Checkpointing, and Communication-Induced

Checkpointing are the three types of checkpointing. Check-pointing could also be categorised according to who is responsible for actually collecting and re-establishing the application execution state. Manual code insertion, pre-compiler check pointing, and post-compiler check pointing are the three options.

On the grounds of their scope, a check point might be local or global. A check-point assigned to a specific process is known as a local check-point, whereas a check-point applied to a group of processes is known as a global check-point. Check-pointing has several drawbacks, such as the fact that it adds to the execution time even when there are no failures. The check-pointing cost is the cost of recording check-point data to stable storage every time a check-point is taken.

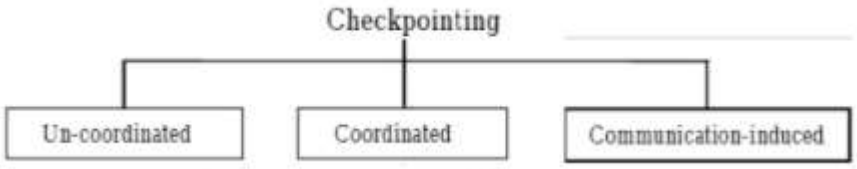


Figure 4. Types of Check Pointing

Scheduling: It is additionally one of the techniques to endure issue from dispersed framework. It is utilized to conquer the disadvantage of registration in dispersed climate. It is sorted as time-sharing planning, space-sharing booking, and mixture blend. Planning may be utilized for adjusting load just as adaptation to internal failure in disseminated framework based on space or time sharing. There have been three types of planning techniques, such as space, time, and crossover. A framework's space booking is used to withstand a long-term or equipment-related difficulty. When there is a surplus of space, the Primary-Backup strategy is used. Whenever there is a discontinuous type of deficiency in the framework, time excess is used. Furthermore, mixture excess is utilized when both are required. There are some significant techniques for enduring flaw in different frameworks given by numerous creators in their examinationAccording to Alain Girault et al., the Algorithm Architecture Adequation (AAA) technique will result in static code, which will be used to implant a constant appropriated framework. This technique fundamentally utilized for processor disappointment with bomb stop conduct. There are two fundamental planning procedures (disconnected, on the web) utilized in programming or equipment disappointment of a framework.

Disconnected is one generally favored progressively installed framework. Disconnected planning is favored in light of the fact that two kinds of disappointment can be present: fizzle quiet, exclusion that can be endured by replicating the tasks, and/or information interchanges.

In the process of replication the planning method is put under the state-machine, and the essential/reinforcement assertions among copies and in correspondence we accept that diverse correspondence utilizes particular cushion.
The majority of the booking calculations depend on the essential reinforcement conspire for intermittent constant assignments, present pointless redundancies by forcefully utilizing dynamic reinforcement duplicates however these calculations depend on fixed-need to upgrade schedulability. A portion of the fundamental calculation alike RMFF utilizes the WCRT where essential duplicate, also reinforcement duplicates are usually have same size to keep check of the schedulability of assignment, also decide the level of reinforcement duplicates executes in aloof structure. In the Tercos, the WCRTs of essential duplicates are never precisely or comparable to the WCRTs of corresponding reinforcement duplicates, allowing us to improve schedulability by using best-fit technique.

Processor and correspondence connect disappointment can be endured by utilizing disconnected booking procedure and create a flaw endure disseminated plan. In this the calculation dependent on diagram change can endure fixed number of subjective processor and correspondence connect disappointment.
There are different definitions to what exactly adaptation to internal failure is. In managing adaptation to non-critical failure, replication is commonly utilized for general adaptation to internal failure strategy to ensure against framework disappointment. The State Machine, Process Pairs, and Roll Back Recovery are three important types of replication systems described by Sebepou et al.
1) State Machine: Concerning this instrument, cycle condition of a PC framework is duplicated over independent PC framework simultaneously, all imitation hubs measure information in practically equivalent to or coordinating method and furthermore there exists coordination in their cycle between the reproduction hubs and the data sources may be shipped off all copy simultaneously. A functioning replica is considered an illustration of state machine.

2) Process Pairs: In replication coordination, the cycle establishes capacities such as an expert (required)/slave (optional) connection. Instead of acting as an expert, the essential workstation communicates its related contribution to the optional hub. The two hubs keep up a decent correspondence interface.

3) Roll Back Recovery (Check-Point-Based) This component gathers check point quickly and moves these checkpoint states to a steady stockpiling de-bad habit or reinforcement hubs. This empowers a move back recuperation to be done effectively when or during recuperation measure. The checkpoint is been remade preceding the ongoing state .

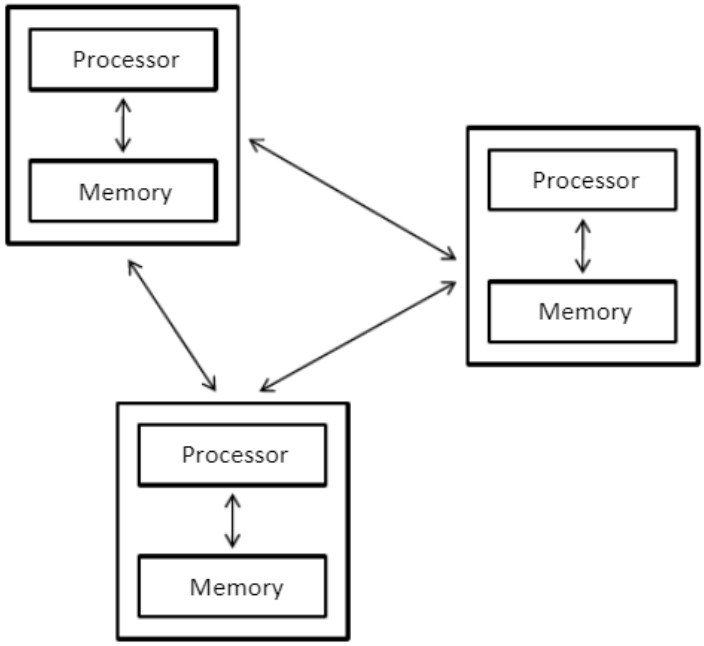


Figure 5. Distributed System

Distributed Systems:

Hubs in circulated frameworks associate and hand-off data by trading the data across a communication channel, whereas dispersed frameworks do not share memory or a clock. The many PCs in a distributed framework have their very own storage and operating system, and the hub claims neighbourhood assets using the assets. The assets that are accessed through an organisation or correspondence medium are referred to as remote assets. The relationship network between frameworks in the dispersed climate is depicted. In circulated framework, bucket of rules is executed to synchronize the activities of different or various cycles on a correspondence organization, subsequently shaping a particular arrangement of related errands. The free framework or PCs access assets distantly or locally in the dispersed framework correspondence climate, these assets are assembled to shape a solitary comprehensible framework. The client in the dispersed climate doesn't know about the

numerous interconnected framework that guarantees the undertaking is done precisely. In circulated framework, no single framework is required or conveys the heap of the whole framework in handling an assignment.

Fault Tolerance Systems:

Internal Failure adaptation in communicated registration, framework is critical; it preserves the framework in functioning order in the event of a failure. Its major goal is to keep the framework running even if one or more of its components fail or break. It is identified with trustworthy frameworks for a framework to be issue open minded. [4] Availability, Reliability, Safety, and Maintainability are some of the needs covered by trustworthiness in the adaptation to internal failure paradigm. Accessibility: When a framework is in a ready condition, it is ready to communicate its capabilities to its contrasting clients. Frameworks that are exceptionally accessible perform as intended at any given time. Unwavering quality: This refers to a computer's ability to perform consistently and without failure. Unwavering quality, in contrast to accessibility, is measured throughout time rather than in a single instant. A very dependable structure that continues to function without interruption over a long period of time. Security: This is when a framework fails to perform its corresponding measures correctly and its actions are incorrect, but no breaking event happens. A genuinely realistic framework may also demonstrate an exceptional assessment of openness, particularly if the corresponding disappointments can be exactly recognised and corrected. As seen, adaptation to internal failure framework is a framework which has the limit of or to continue running effectively and legitimate execution of its projects and keeps working in case of a halfway disappointment. Although, on occasion, the framework's exhibit is impacted by the dissatisfaction that occurred. Unauthorized Access or Hardware or Software Failure (Node Failure) account for a component of the fault (Machine Error). Mistakes caused by adaptations to non-critical failure events are divided into five categories: execution, oversight, timing, crash, and comeback. Execution: this is the point at which the equipment or programming parts can't satisfy the needs of the client. Oversight: is when parts can't actualize the activities of various unmistakable orders. Timing: this is when segments can't execute the activities of an order on the perfect time. Crash : some segments crash with none reaction and can't be fixed. Come up short stop: is the point at which the product recognizes blunders, it closes the cycle or activity, this is the most straightforward to deal with, now and then its effortlessness denies it from taking care of genuine circumstances.

2.2 Fault resolution mechanisms:

2.2.1. Fault Tolerance Technique based on Replication:

Probably one of the best techniques is replication-based adaptability to non-critical failure [5]. This technique effectively duplicates data from many different frameworks. A request can be sent from one imitation framework to the other replication framework throughout the replication operations. As a result, if one or more hubs fail to function, the entire system will not be affected. In a framework, replication adds repetition. Customer interaction, worker coordination, execution, comprehension, coordination, and customer reaction are all stages of the replication standard. Consistency, degree of reproduction, imitation on interest, and other factors are important in replication-based tactics. Consistency is a crucial aspect of the replication process. Because of the refresh that ought to be possible by every client, a few replicas of an identical element cause a consistency issue. Some rules ensure information consistency, such as linearizability, successive consistency, and easy-going consistency, among others. Consecutive and linearizability consistency provides strong consistency, but easy-going consistency is a fragile consistency norm. An important reinforcement replication technique, for example, ensures consistency by linearizability, and a dynamic replication approach does the same. The replication techniques employ a number of conventions in the replication of data or an article, including primary reinforcement replication, casting a vote, and important per parcel replication. To achieve a substantial level of consistency at the replication level, a large number of copies are necessary. If the quantity of imitations is minimal or non-existent, it will have an impact on flexibility, execution, and other adaptations to internal failing abilities. A flexible reproductions creation calculation was presented to answer the issue of a low quantity of imitations.

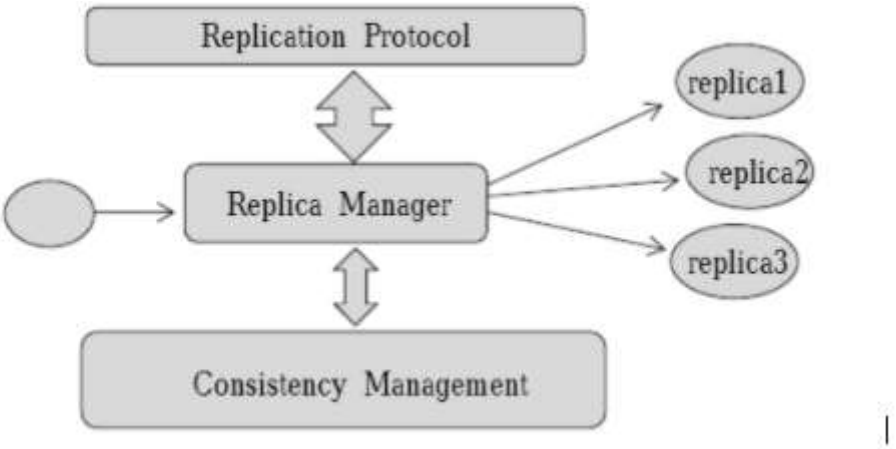


Figure 6. Replica Management

2.2.2. Process level Redundancy Techniques:

This method for adapting to internal failure is frequently used for flaws that disappear without being addressed; this type of fault is referred to as transitory issues. When there is a temporary fault in any of the framework parts, or in some situations, when there is an ecological impedance, transient flaws occur. Transient faults have the disadvantage of being difficult to deal with and understand, but they are less significant in nature. Programming-based adaptation to non-critical failure methods, such as Process-Level Redundancy (PLR), are used to cure transitory flaws since hard-product-based adaptation to non-critical failure procedures are more expensive to transmit. As is well known, the PLR considers cycles in order to ensure proper execution, and it also performs a series of repeating cycles for each application cycle. Excess at the cycle level allows the OS to arrange measures efficiently across all available equipment resources. With a 16.9% overhead for programming transient adaptation to non-critical failure techniques, the PLR provides better execution than current programming transient adaptation to non-critical failure methods. PLR employs a product-driven paradigm, which shifts the focus from assuring correct equipment execution to ensuring proper programming execution.

2.2.3. Fusion Based Technique :

Replication is indeed the most widely used approach or method for non-critical failure adaptability. The main drawback is the large number of reinforcements it necessitates. The combination based technique addresses this issue for reinforcements increase as defects

increase and the board's cost is exorbitant. The combination-based technique is still a possibility since it requires fewer reinforcement machines than the replication-based method. In comparison to the supplied machine layout, the reinforcement machines are interwove. The overhead of a combination-based method is significant throughout the recovery cycle, but it is appropriate when there is a minimal chance of insufficiency in a framework.

In conclusion, adapting in scenario of non-critical failure is a significant piece of dispersed framework, since it guarantees the coherence and usefulness in respect to a system at some point when there's an issue or disappointment. This examination demonstrated the diverse kind of adaptation to internal failure technique in circulated framework, for example, the Fusion Based Technique, Check Pointing and Roll Back Technique, and Replication Based Fault Tolerance Technique. Every system is beneficial over the other and expensive in organization. Here we look into degrees of adaptation to non-critical failure, for example, the equipment adaptation to non-critical failure which guarantees that extra reinforcement equipment, for example, memory block, CPU, and so forth, programming adaptation to non-critical failure framework includes checkpoints stockpiling and rollback recuperation instruments, and the framework adaptation to non-critical failure is a complete framework that does both programming and equipment adaptation to internal failure, to guarantee accessibility of the framework during disappointment, mistake or issue. Future examination would be led on contrasting the different information security systems and their exhibition measurements.

2.3. Key-Value Store:

A key-value store, also known as a key-value information base, is a simple data collection that uses an associated cluster (think of it as a guide or word reference) as the main information model, with each key being linked to one and only one incentive in an assortment. This duo is referred to as a key-value pair.

Table 1. Key-Value Store Example

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

A subjective string, such as a filename, URI, or hash, is spoken to the key in each key-value combination. Any type of data, such as a photograph, a customer preference record, or an archive, might be worth anything. The value is stored as a mass that does not require any upfront information showing or diagram definition.

The incentive's bulk capacity reduces the need to keep track of data in order to enhance performance. However, because the value is hazy, you can't channel or control what's returned from a request based on it.

When everything is said and done, there is no question language in key-esteem stores. They let you to save, retrieve, and update data using simple get, put, and erase commands; retrieving data is as simple as sending an instantaneous request to the item in memory or on plate. The simplicity of this architecture makes a key-value store quick, easy to use, flexible, handy and adaptable.

Partitioning (storing data on several nodes), replication, and auto recovery are used to scale out key-value stores. They can scale up by keeping the database in RAM and eliminating locks, latches, and low-overhead server calls to reduce the impact of ACID guarantees (a assurance that completed transactions persist someplace).

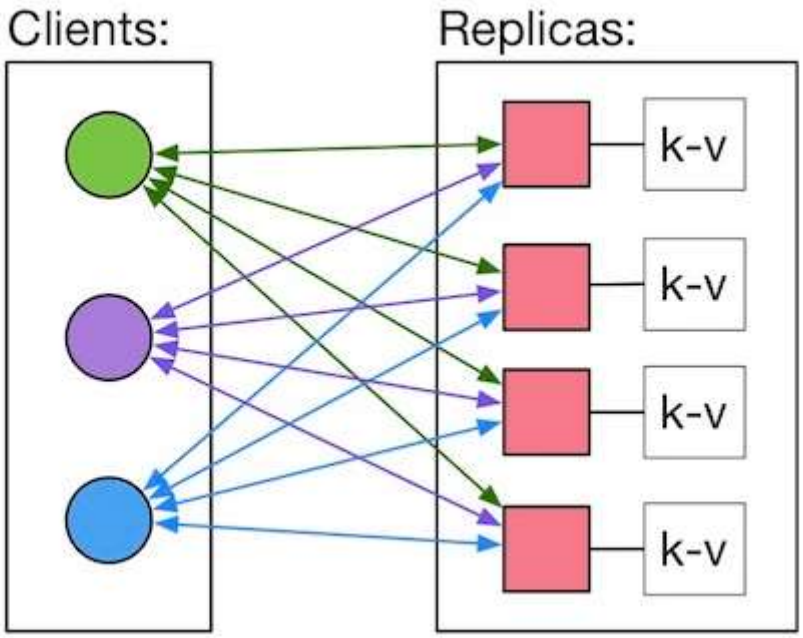


Figure 7. Connectivity between clients and replicas in key-value store

Chapter 3
System Development

3.1. Building a distributed key-value store:

A key-value store, often known as a dictionary or hash, is a file storage paradigm for storing, accessing, and maintaining associative arrays, a data structure that is more generally known as a dictionary or hash today. A distributed Key Value store is one where data is replicated across different nodes such that there is:

- High-Availability, and
- Non presence single points of failure

We are going to follow:

- Consistency type: Strong, Quorum based
- Membership protocol: SWIM
- No authentication or SSL support of as of now – plain old open http
- Local clocks will be used, as they are already in sync with system clock.
- The data will be stored in memory (in context of the process), no commit logs will be maintained; If all process die or some most die before replication data will be lost.

3.2. System Architecture:

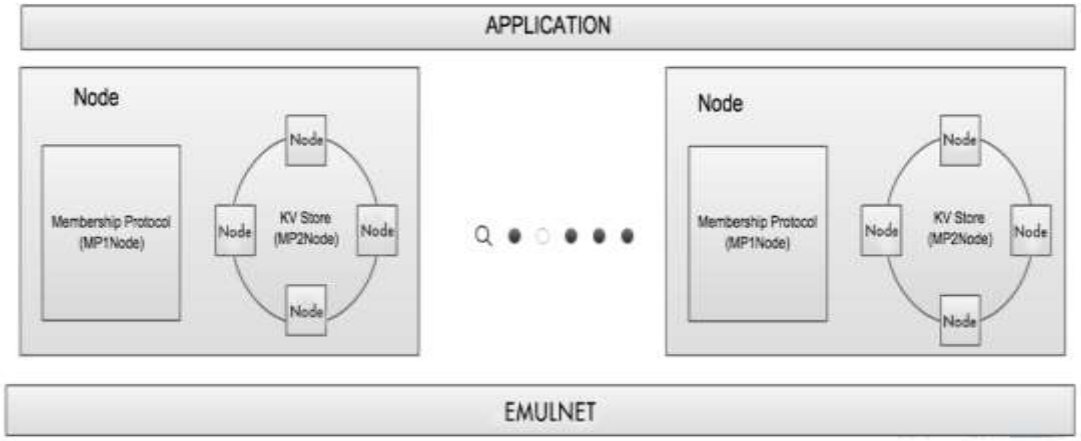


Figure 8. System Layers

3.3 Classes/Entities Used:

1. **Hash Table:** A class that wraps C++11 std::map. It supports keys and values which are std::string. This has already been implemented and provided to you.

2. **Message:** This class can be used for message passing among nodes.

3. **Entry:** This class can be used to store the value in the key-value store.

4. **Node1:** This class wraps each node's Address and the hash code obtained by consistently hashing the Address.

5. **Node2:** This class must implement all the functionalities of a key-value store, which include the following:

- Ring implementation including initial setup and updates based on the membership list obtained from Node1
- Provide interfaces to the key value store
- Stabilizing the key-value store whenever there is a change in membership
- Client-side CRUD APIs
- Server-side CRUD APIs

3.4. Storage Replication and Stabilization:

3.4.1. Write Request:

With a key-value pair, the client can send a WRITE REQUEST to any node; this node will serve as a coordinator and during life-cycle of this request; the coordinator will measure the key hash and will be able to find the nodes in which key should be stored. The hash feature will essentially point to one node, let's called it the key's primary store. There'll also be NoOfReplicas = MaxReplicationCount - 1 replicas in my plan. NoOfReplicas nodes will be chosen as replicas after the primary key store in the ring; the coordinator will send the internal write request to each of these nodes and wait for response; as soon as quorumCount node responses return, the write request will be set as completed and the performance code will be returned to the client; otherwise in case of timeout or malfunction requests > MaxReplication.

3.4.2. Read Request:

Similar to WRITE Order, the request is sent to replicas and the value is answered after the reply is received from the lease quorumCount of replicas and the values are consistent. If any of the replicas have older values, the coordinator initiates READ Fix for them if the message is not obtained from quorumCount with value 404, it responds to Not Found; It may be that DELETE failed them or failed to write for them if less than quorumCount responds with a value. In any event, we can either initiate or leave an internal DELETE request to them.

3.4.3 Delete Request:

Similar to above two, initiate request to all replicas and respond back OK if quorum responds with that number.

3.4.4. Stabilization:

Each when a new node enters or an existing one exits the system, stabilization needs to be completed. The configuration of the ring varies a little in both cases and the mapping of the keys changes to the serverand any node that detects a mistake or kicks the node joining in the stabilization protocol.

Chapter 4
Performance Analysis

The tests for checking the functioning of our model include:

1. Basic CRUD tests that test if three replicas respond
2. Single failure followed immediately by operations which should succeed (as quorum can still be reached with 1 failure)
3. Multiple failures followed immediately by operations which should fail as quorum cannot be reached
4. Failures followed by a time for the system to re-stabilize, followed by operations that should succeed because the key has been re-replicated again at three nodes.

Corresponding to every CRUD operation, the nodes log few messages:

- All replicas (non-faulty only) should log a success or a fail message for all the CRUD operations
- If the coordinator gets quorum number of successful replies then it should log a successful message, else it should log a failure message.

We tested our membership-protocol in three states and grade all of them on three different metrices. The scenarios to check are as follows:

1. Single-Node Failure
2. Multiple-Node Failure
3. Single node failure with lossy network

Our grader will test the following things:

1. If all the nodes join the peer-group rightly
2. If all the nodes detect the failed node (represents completeness)
3. If correct failed node was detected (represents accuracy)

The parameters used by the application for the same are:

a) MAX_NNB : val – represents max no. of neighbours

b) SINGLE_FAILURE : val – is a one bit 1/0 variable that sets single/multi failure scenarios

c) MSG_DROP : val – is a one it 0/1 variable that decides if messages will be dropped or not.

d) MSG_DROP_PROB : val – represents the message drop probability (between zero and one)

	MAX_NNB	SINGLE_FAILURE	DROP_MSG	MSG_DROP_PROB	Join	Completeness	Accuracy
Single Node Failure	10	1	0	0.1	100%	100%	100%
Multiple Node Failure	10	0	0	0.1	100%	100%	100%
Single Node Failure under Lossy Network	10	1	1	0.1	100%	100%	100%

Table 2: Membership Protocol test results

CRUD operations' tests:

1. For create test, three replicas of every key is created.
2. For delete test, all created replicas are deleted. Deletion of an invalid key is also attempted.
3. For read test, following tests are performed:

TEST 1: Read a key, then check for correct value being read at least in quorum of the replicas.

TEST 2: Read a key after failing a replica. Then check for the correct value being read at least in one quorum of replicas.

TEST 3 PART 1: Read a key after failing two replicas. In such a scenario, the read should fail

TEST 3 PART 2: Read the key after allowing stabilization protocol to kick in. Check for correct value being read at least in quorum of replicas.

TEST 4: Read a key after failing a non-replica. Check for correct value being read at least in quorum of replicas

TEST 5: Attempt read of an invalid key.

4. Update test consists of following tests:

   TEST 1: Update a key. Check for correct value being updated at least in quorum of replicas
   TEST 2: Update a key after failing a replica. Check for correct value being updated at least in quorum of replicas
   TEST 3 PART 1: Update a key after failing two replicas. Update should fail
   TEST 3 PART 2: Update the key after allowing stabilization protocol to kick in. Check for correct value being updated at least in quorum of replicas
   TEST 4: Update a key after failing a non-replica. Check for correct value being updated at least in quorum of replicas
   TEST 5: Attempt update of an invalid key.

All the above stated tests passed successfully. The analysis was done using a .sh file that read logs from .log files of all the nodes being created, nodes joining other nodes, nodes being deleted, and message count of each node.

Conclusion:

Implementation of a Gossip style protocol such that it satisfies completeness at all times, with no message delays or losses leading to successful simulation of a fault tolerant system is done. The simulated network continues to work correctly during node failures. Understanding of working such networks in real world is deepened.

References:

[1] Chen, W.H. and Tsai, J.C. (2014) Fault-Tolerance Implementation in Typical Distributed Stream Processing Systems.

[2] Sari, A. and Necat, B. (2012) Securing Mobile Ad Hoc Networks against Jamming Attacks through Unified Security
Mechanism. International Journal of Ad Hoc, Sensor & Ubiquitous Computing, 3, 79-94.

[3] Balazinska, M., Balakrishnan, H., Madden, S. and Stonebraker, M. (2008) Fault-Tolerance in the Borealis Distributed Stream Processing System. ACM Transactions on Database Systems, 33, 1-44.

[4] Sari, A. and Çağlar, E. (2015) Performance Simulation of Gossip Relay Protocol in Multi-Hop Wireless Networks. Social and Applied Sciences Journal, Girne American University, 7, 145-148.

[5] Harper, R., Lala, J. and Deyst, J. (1988) Fault-Tolerant Parallel Processor Architectural Overview. Proceedings of the
18st International Symposium on Fault-Tolerant Computing, Tokyo, 27-30 June 1988.