International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Index Copernicus Value (2016): 79.57 | Impact Factor (2017): 7.296

Techniques for Managing Inconsistency to Scale Services in Distributed Memory Systems

Gnana Teja

Abstract: ***Distributed memory systems are essential commodities in present - day computation since they facilitate intricate computations in nodes that are often interconnected. However, as the masses of these systems increase, consistency management becomes increasingly more complex because of factors such as delays in the network, node crashes, and concurrent changes to the data. The scalability of various techniques of managing inconsistency in distributed memory systems is the concern of this paper. In the case of consistency models, this paper focuses on the aspects of strong and weak and their advantages and disadvantages. The strong consistency guarantee makes the data identical across multiple nodes, but this reduces the system's scalability and performance; hence, it is inefficient for applications like real - time data processing. Weak consistency models, in contrast, are less strict and allow some temporary data inconsistency, making them even more performant and scalable. However, they may need to provide more accurate data to consumers. The paper also focuses on the kinds of data, single primary users and multiple users, and how these determine the consistency models. Moreover, methods of consistency management of services evolving further are analyzed, as well as features and tendencies in the implementation of consistency management, including the hybrid and adaptive models and their perspective for growing services in the future. These models claim to balance how much of the actual data is captured. At the same time, it improves the system's response time, making it imperative, especially for developers and businesses aspiring to tackle massive distributed systems. The paper concludes and highlights that as distributed systems evolve to unprecedented complexity, scalability, and resilience of consistency will prove central to confidence in system integrity. It lays down the sum and substance of the existing methods and prospects that will be useful for the study's practitioner and theoretician.***

Keywords: Distributed Memory Systems, Consistency Models, Scalability, Strong Consistency, Weak Consistency, Eventual, Consistency, Concurrency, Network Latency, Data Replication, Hybrid Consistency Models

1. Introduction

Distributed memory systems constitute a significant element of contemporary computing, allowing extensive calculations to be performed on networks of nodes. These systems enable the passing of data and computational requirements across different nodes and thus enable the handling of massive data and computations that, if done on a single machine, would be extremely difficult. In the current world of big data, cloud computing, and real - time analysis, distributed memory systems have become standard building blocks in the architecture of most modern essential applications ranging from social networks to online banking and science applications. However, since these systems are composed of multiple nodes operating in parallel, complexity is created, especially in node consistency.

Several issues present in distributed memory systems include inconsistency issues that arise when many nodes change the state of an object concurrently. These inconsistencies can arise due to several issues. For example, there might be delays in the network. A node might have failed or, in case of simultaneous modifications to the stored data. An update of a social network profile, to announce that the profile is updated in all of the nodes of the system, the other users will be viewing the updated information. However, if different nodes are in some way contrary due to the network latency or other concerns, users will see stale or wrong data. This challenge intensifies as the system grows because the number of nodes and users also rises, meaning that inconsistent information is more probable.

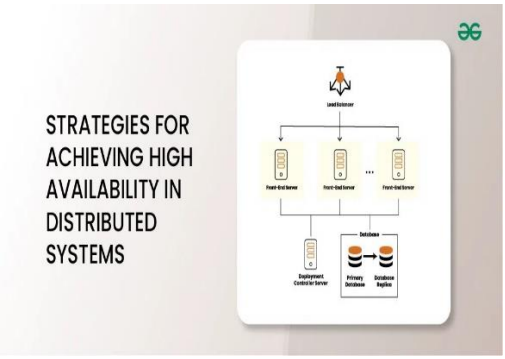

Figure 1: Strategies for Achieving High Availability in Distributed Systems

The rationale for which these disparities should be treated is that they produce negative forms of amplification in terms of the system's operation and the end - user interaction with it. Such data can produce errors and confusion, and, in the bigger picture, the user loses confidence in the system. For instance, microtransactions in financial services may require the records of transactions to be the same across multiple nodes; any disparities can cause issues such as losing money or getting entangled in legal problems. Thus, the measures should be used while bearing in mind that there is always a conflict between the consistency and performance of the system. Finding this balance is not easy because, for example, achieving a firm consistency usually implies that a significant amount of resources has to be used, and they can cause delays, which are not acceptable in cases where the systems have to work in real - time.

2. Purpose of the Study

The purpose of this article is to discuss multiple approaches to handling conflict situations in deploying distributed

memory systems, particularly on scalability services. This is because as systems become large and complex, the coherence issues become sensitive and need to be handled by specific strategies and models. The intent here is to give a brief overview of these methods and then understand how all these can be approached so that enhancing the efficiency and reliability of such systems can be achieved as the systems grow.

One of the significant discussion areas in this section will be analyzing various consistency models that can be incorporated into distributed memory. Consistency models state how data consistency is established at all the nodes in a system and whether all nodes contain identical data, even in the case of concurrent updates (Brooker, 2014). It is essential for anybody designing, implementing, or administering distributed systems because the chosen model will likely significantly impact how well the system scales, how well users accept it, and just how performable it is.

In the article, the author will present both strong and weak consistency models, emphasizing simultaneously the positive effects and negative consequences of their application. The strong consistency models, for instance, guarantee that all the nodes in the system are up to date on the change as soon as there is an update. Although this ensures the maximum degree of preciseness, it is time - consuming and could be costly, which makes it less suitable for systems that interact with large datasets in real time. Weak ones permit temporary while making it less costly and more scalable inconsistencies in the data seen by the nodes; this is why sometimes users may notice inconsistent data at different nodes. Through the discussion of these models and the techniques related to them, readers of this article will be equipped with enough information to enable them to make sound decisions regarding the management of consistency in distributed memory systems. Whether the aim is to optimize efficiency, work with larger numbers of users, or employ the system in larger circumstances, the information provided here will be helpful for practitioners.

Understanding Consistency in Distributed Memory Systems

What is Consistency?

Consistency in distributed memory systems deals with how close the multiple nodes in the particular system are synchronized in terms of the data they have at a particular moment. For an enhanced distributed system, it must be incorporated that every bit of data modified in one or any portion of the system must be immediately accessible to all the other parts so that all nodes are working with the same or an updated version of the data. However, realizing this notion, referred to as solid consistency, is usually not easy because of the characteristics of distributed systems, which include occasional delays in data transmission, some nodes may fail, and some data being changed may be changed simultaneously by different nodes. That is primarily true since consistency ensures that distributed systems behave correctly, mainly when several nodes are involved and perform operations together. Lack of consistency could lead to distributed systems providing the wrong or out - of - date information and producing less than accurate results, which is detrimental to the use (Bailis et al., 2014).

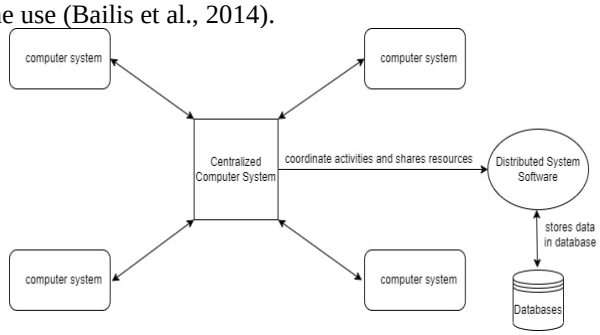

Figure 2: Consistency Model in Distributed System

Another common and vital cause of inconsistency in distributed systems is the extra time taken by the network. In case the update is made to data often, the change has to extend throughout all nodes of the whole system. The propagation period involves replication of the data to all the nodes, and at this time, nodes could be holding different versions of the same data, leading to inconsistency. Further, node failures can worsen this issue, too, because the nodes are the main components of the ad hoc network. The problem is that if a node goes offline before it has received the last update, then after it has restarted, it will return a stale answer. This can be especially so in those contexts where scrubbing is accompanied by a real - time requirement of data – integrity, which is often the case, for example, in the financial field or records in a hospital (Bazzi, 2002).

One of the other typical cases, when conflicts occur, is concurrent data updates. In distributed systems, it is possible to have two nodes trying to write simultaneously. However, these multiple updates can work in parallel and sometimes need to be more consistent, where there shall be different versions of the data in different nodes. This problem is solved by employing different consistency models, each providing assurance on where, when, and how updates are made and observable across the system (Brooker, 2014). For example, sequential consistency guarantees that all nodes will see the operations in the same order as they were performed, even if this order may not reflect the order in which the operations were executed (Lamport, 1979).

Types of Data in Distributed Systems

When it comes to distributed systems, it is important to know the kinds of data in order to manage consistency. Data in distributed systems can be classified primarily into single primary user information and multiuser information. The first of these classification types depends on the number of users allowed to modify the data apart from updating it. In this paper, the type of data significantly influences both the choice of consistency model and the system's performance.

Single Primary User Data

Single primary user data is updated chiefly by one person or process, even though others can use it. This data type is relatively easier to maintain in a distributed system architecture because the probability of concurrent update transactions is lower. The probability of getting into a inconsistent state is lesser. For instance, on a social media platform, a user's profile, which contains information about the user, can only be edited by the user, although others can

see the profile. In such cases, consistency can be maintained reasonably easily because if the user changes something in the system, it must be reflected across several nodes.

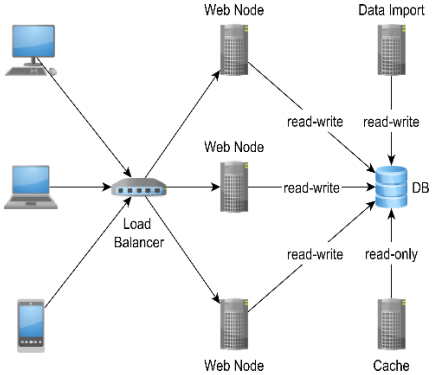

Figure 3: Single - Primary Database Replication

The effect on performance in managing single primary user data is positive, as it is easier to maintain consistency with fewer resources. As long as only one user is making changes at the moment, the system can make these changes fast and have all nodes updated within the shortest time possible without dealing with issues that arise when several users are making changes simultaneously. Still, the difficulty appears when the system has to scale and simultaneously serve a high number of read requests along with providing access to the most recent data version. This area calls for proper formulation of caching and replication to enhance overall consistency and performance.

Multiple User Data

The other type of multiuser data is the situation where a data set is changed by multiple users or at the same time by other processes. It is challenging in distributed systems because the probability of having concurrent updates, which lead to having updates happen simultaneously, is relatively higher. Some examples of multiple - user data are documents created and edited by multiple users at a given time, calendars, or any system designed so that a number of users can set data simultaneously. In these cases, the system needs to implement techniques to ensure that all changes are correctly applied and that the event ends in the correct state of the data across the nodes (Vogels, 2009).

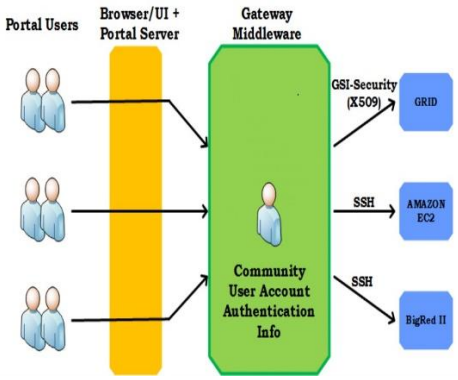

Figure 4: Multiple User Database Overview

In the case of managing multiple user data, keeping consistency often necessitates more complex approaches, such as reconciliation strategies or the utilization of consistency models that enable eventual consistency (Brooker, 2014). For instance, eventual consistency permits inconsistency for a limited time but promises that every node will become in sync with the others. This model is ideal when the adoption requires consistency, and the system can wait a long time before fixing the disparities in consistency and propagating updates. However, this will negatively affect the user interface, particularly in applications whose users demand updates or feedback in real - time (Brewer, 2000). Knowledge of the nature of data to be managed in a distributed system seems crucial in choosing the right consistency model and ensuring that the system can grow horizontally and perform well. Single primary user data and multiple user data also imply different problems, which recalls the need to customize accommodating solutions for the system.

Consistency Models in Distributed Systems

Distributed systems are very important in the current computing environment since they help in large - scale computing in distributed nodes. One of the main problems in such systems is the problem of Consistency across distributed data, which only worsens as services grow. Consistency models are essential in guiding how data should act in distributed settings. These models explain how much of the system the users can access, how different parts of the system can be affected or accessed independently, and how data will be consistent across the system. This section will discuss the several consistency models, their classification, and exemplar use cases. However, we will only consider some appropriate consistency models for a given model.

Overview of Consistency Models

Consistency models in distributed systems are broadly classified into two categories. Two subcategories can be further divided into Object Consistency, Strong Consistency, and Weak Consistency. Strong Consistency promises that all nodes in a system will have a similar new data set within any update, and all the read operations will get the latest written data. Thus, in the Weak Consistency model, it is permissible to have differential views of nodes for a specific time (Tanenbaum & Van Steen, 2007).

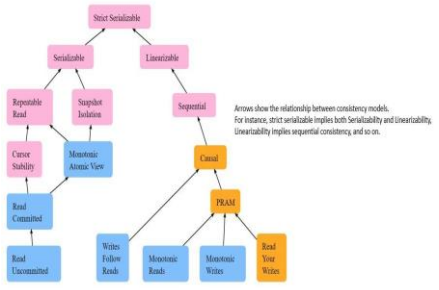

Figure 5: Introduction to Consistency Models

The suitable consistency model must be chosen carefully, as it dramatically influences the system's characteristics and perceived usability. The choice, therefore, depends on the nature of the application and what relative preference between Consistency, availability, or performance is desirable. For example, if an application processes transactions that should be available in real - time, such as a banking application, it should have a firm consistency requirement for data. In contrast, in systems where the 'eventual' Consistency is acceptable, for example, in social networking, it is quite preferable to utilize the weaker consistency models than the

strong ones to increase scalability and performance (Vogels, 2009).

3. Strong Consistency

Definition and Characteristics

Strong Consistency means that every read operation receives the most up - to - date version of data that is even written in every node of the system; this is also called linearizability. Once a write operation has been performed, any read operation, irrespective of which node is carried, will return this new value. The high degree of Consistency impacts the system with the highest level of data consistency and is thus preferred for systems where correctness and currency of information are paramount (Herlihy & Wing, 1990).

Challenges in Achieving Strong Consistency

Getting to very high levels of Consistency in a distributed system is easy due to the latency and partitioning problems inherent in the distributed system environment. Synchronizing all the nodes to make them display up - to - date information entails massive performance issues in most large - scale systems. Waiting for acknowledgment from the many nodes before a write operation is completed adds latency that limits the possible performance while at the same time having strong Consistency (Charron - Bost & Schiper, 2000).

Figure 6: Challenges in Achieving Consistency

Use Cases Where Strong Consistency is Necessar

High Consistency is required in cases where data accuracy is a crucial factor, and the simultaneous existence of inconsistent data is dangerous. Some examples are financial systems where the disparity of a single account results in vast loss and distributed databases that contain delicate information such as personal health records (Brewer, 2000). These systems always want a read operation to return the result of the last transaction so that end users always work on the most up - to - date information.

Weak Consistency

Definition and Types: Weak Consistency models allow nodes to have different copies of the data for some time. This approach is usually used in distributed systems where some priority is given to performance and availability rather than data integrity. These include Operation - Centric, Transaction - Centric, and Application - Centric models, which provide flexibility tailored to various requirements and conditions (Tanenbaum & Van Steen, 2007).

Operation - Centric Consistency Models: Two primary versions of Operation - Centric Consistency models are known. Although these models are similar to other CCMs, they are different because they concentrate on the order and visibility of operations rather than on the state of data. These models are utilized to ensure that users are given a seamless experience regardless of the fact that there are simultaneous updates. Some of the Operation - Centric models include Sequential Consistency, Causal Consistency, and Eventual Consistency.

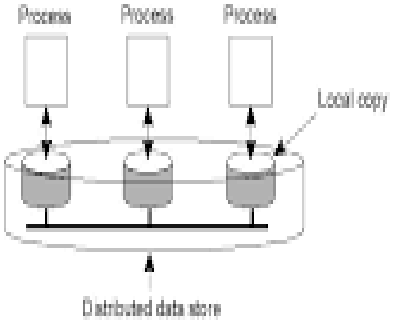

Figure 7: Centric Consistency Model

Sequential Consistency: In Modified or Sequential Consistency, all nodes observe operations in the same order, although this order is not necessarily the same for real - time orders. Since this model is more accessible to implement than the robust consistency model, it is preferable. It is beneficial in those systems where the order of operations is more important than the time to complete them (Lamport, 1979). For instance, in distributed game middle - ware systems, all players at the different sites must see them in the same sequence as they happened, although the time differences could be significant.

Causal Consistency: Causal Consistency states that all nodes observe operations with a particular causal relation in the same order. However, those operations that are not necessarily dependent on each other can be viewed in different sequences with different nodes. This model is especially desirable in cooperative environments where the temporal order of the users' actions must be retained with a view to consequent Consistency (Hutto & Ahmad, 1990). For instance, in a collaborative editing tool, it is essential to be sure that the updates made are done causally consistently to avoid spoiling the document's integrity.

Monotonic Reads and Writes: Monotonic Reads guarantee that each time the same process reads an object, it only gets a time value that is not less than the time value of the prior read. Monotonic writers make a provision to control subsequent writes made by a particular process in the correct sequence. These properties are crucial in in - use scenarios such as online calendars and inventory management, where the order of operations impacts the interaction as well as the efficiency and stability of the systems (2016).

Read Your Writes and Writes Follow Reads: The Read, Your Writes model ensures that the client who succeeds in a write and immediately follows it with a read operation receives the latest value of the variable. The Writes Follow Reads model guarantees that before a client writes into the shared space, he first reads from it, or, at the least, he writes what was read by another client at an earlier time. These models are essential in all the applications where the

consumer is in a position where he or she expects instantaneous comments on his or her actions, such as social media and web applications (Ladin et al., 1992).

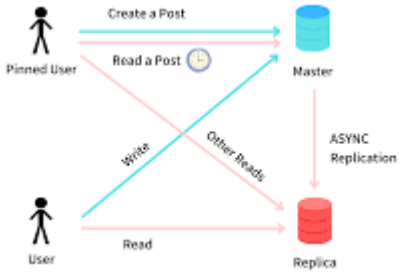

Figure 8: Read - your - write consistency

Eventual Consistency: Eventual Consistency is even weaker than Hybrid Consistency as it only provides solutions that guarantee that all the nodes will eventually be updated if no new updates are made. This model permits transient divergence of the copies but guarantees that these divergences will eventually fade away (Vogels, 2009). Despite some weaknesses, eventual Consistency enjoys high popularity among distributed systems that call for availability, including content delivery networks and extensive web services. It is used to ensure that systems run and continue to function even if the network is split, with the expectation that data will be refreshed across all the nodes at some later time (Vogels, 2009).

Transaction - Centric Consistency Models

Transaction - centric consistency models aim to achieve Consistency within the context of transactions, which are sequences of operations intended to be executed in a single and abrupt manner. These models offer varying levels of assurance on the sight and scheduling of activity within transactions over many systems.

ACID (Atomicity, Consistency, Isolation, Durability): Acid properties are central to transactions and reliability in a distributed system. Atomicity ensures that all changes made as part of a given transaction are fully processed or none are made. The principle of Consistency means that transactions take the system from one legal state to another. Isolation ensures that two transactions do not conflict with each other. In contrast, Durability means that once a transaction has been completed, the results are permanent even when the system has crashed (Gray & Reuter, 1993). ACID transactions are helpful in systems that insist on a high level of Consistency, such as financial applications and distributed databases.

Figure 9: ACID (Atomicity Consistency Isolation Durability) Model of database

BASE (Basically Available, Soft state, Eventual consistency): The BASE is a model for distributed systems with an ACID opposite: B for breakpoint, A for availability, S for soft state, and E for eventual consistency. As with most BASE systems, these are often "Basically Available, " implying that the system was intended to continue running even when there is a partial failure. Some of them are in a "Soft state, "which can sometimes show a state of the system that is different from the current one. Last, they provide "Eventual consistency, " which means the system will eventually be consistent (Pritchett, 2008). BASE is particularly well suited for massive - scale distributed systems in which high availability and tolerance to faults are valued higher than strict Consistency, which is why it is extensively used in distributed caches and CDN - like systems.

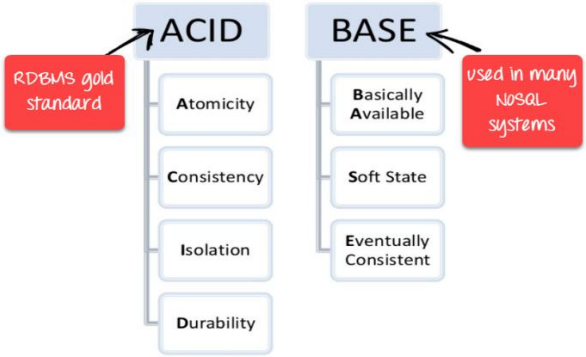

Figure 10: ACID VS BASE

Application - Centric Consistency Models

Application - centric consistency Models give different guarantees that are entirely adjustable to suit the application's desired goal. These models aim to achieve performance, availability, or application usage parameters with trusted strong Consistency and high availability.

Tailoring Consistency for Specific Applications: Application - centric models accept that various applications possess and should have different consistency needs. For instance, it is okay in a social media app if updates are delivered a couple of seconds late; in a banking app, it could be better. For this reason, developers should propose consistency models according to the needs of the developed application so that high performance, good usability, and data Consistency can be provided (Bailis et al., 2013).

Conflict - Free Replicated Data Types (CRDTs): Conflict - free replicated Data Types (CRDTs) are concepts that entail structures that can be updated simultaneously to avoid consistency conflicts while maintaining an aspect of the eventual consistency design. CRDTs are designed so that replication and synchronization are performed in a fault - tolerant manner, conflicts are resolved automatically per specific rules, and all data replicas reach some common state (Shapiro et al., 2011). Due to the nature of concurrent data updates in collaborative CPUs and distributed schemes, CRDTs play an optimal role.

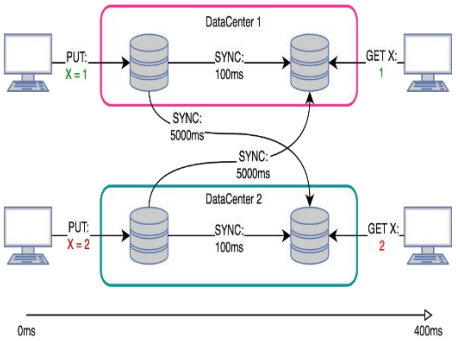

Figure 11: Conflict - free Replicated Data Types

Challenges and Trade - offs in Consistency Management

Resource Consumption vs. Performance

The search for high levels of consistency in distributed systems increases resource consumption and thus incurs consequences on general performance. High consistencies ensure that all nodes within a system implement the same data state right after an update; this feature is essential for organizations that depend on data consistency, such as in finance and healthcare through database systems (Charron - Bost & Schiper, 2000). However, this level of consistency requires a lot of computational and networking powers, which are costly to supply. Each update requires an identical copy on all nodes and results in what can be a significant consumption of resources, chiefly within large installations. This synchronization requires ordering and acknowledging of operations, and since the algorithms used, these are likely to cause high latency and low throughput (Tanenbaum & Van Steen, 2007). For instance, the Paxos algorithm, well - known as a solution for the consensus problem in the distributed environment, requires high resource consumption mainly when applied to an environment with many nodes and updates (Lamport, 2001).

However, there are weaker consistency models like eventual consistencies, which are much more resource - favorable because they allow inconsistencies between the nodes for a certain period. According to Vogels (2009), these models emphasize availability and partition tolerance; hence, they can be helpful where high scalability and response time are desired. For example, Amazon's Dynamo DB uses an eventually consistent model for faster handling of queries for data across different servers. In contrast, the consistency associated with this model is less of a priority for large e - commerce websites (DeCandia et al., 2007). On the other hand, they may sometimes get old news, which is quite unbeneficial in a condition where updated information is needed.

User Experience Considerations

It is tricky to consider accurate user requirements at one moment and provide them with constant results. However, it is needed in applications with high performance and data accuracy. Although they guarantee data correctness, strong consistency models introduce more latencies, and users are affected by it. For instance, in real - time applications such as online games or stock trading, small latencies of data synchronization may be displeasing among users (Herlihy & Wing, 1990). Hence, the middle ground is pursued by developers – they use the so - called hybrid consistency models that provide for solid consistency during the most critical operations and simultaneously allow, for instance, eventual consistency in the case of the less critical tasks.

A successful implementation of this concept can be observed in efforts to construct Google's Spanner. This global database product delivers strong consistency by employing a synchronization method in combination with atomic clocks (Corbett et al., 2013). Specifically, Spanner's architecture is designed to maintain low latency and high throughput. At the same time, Spanner puts much effort into controlling the consistency required for all operations so as not to affect the users negatively. Another example is LinkedIn's Voldemort distributed key - value storage system, where dev and elopers can set tunable consistency models depending on application demands (Sumbaly et al., 2012). Such flexibility allows LinkedIn to adhere to a versatile user experience while maintaining the data unvarying where necessary.

Impact of Network Delays and Failures

Failure of some of the network components and delay issues will significantly affect the synchrony of the distributed systems. Events such as network partitions that render nodes incapable of communicating, which results in different data updates, are some of the causes or instances in which inconsistent data states can be realized. The strong consistency models, for instance, the ones utilized in the Raft consensus algorithm, help to solve this problem by ensuring no commitment is made whereby there is no agreement by the majority, hence preventing conflicts during the network partition (Ongaro & Ousterhout, 2014). However, this approach can cause significant lags in high - latency systems or networks with many failures, as the algorithm has to wait for slow or failed nodes to come through or get ejected.

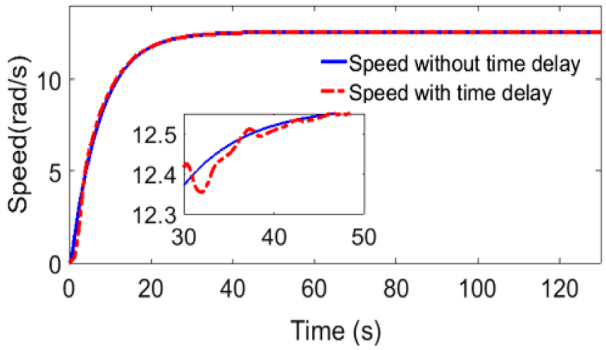

Figure 12: The impact of network delay

To counter these, different approaches, such as replication and data partitioning, are used to improve the system's availability and cushion the effect of delays occasioned by an extended network. For example, Dynamo – the data store for Amazon web service –uses partitioning of data and replication of data in order to keep data available all the time, and thus, in case of a network split, the system can continue to perform with low latency using the eventual mode of consistency (DeCandia et al., 2007). Another approach is quorum - based techniques, where only some nodes have to approve each update before the modification is complete. This approach decreases the effect of network stability and reduces the time needed to reach consistency (Chandra et al., 2007). However, these methods are only suitable where consistency and availability have been traded off, for example, where the network is partitioned most of the time.

Strategies to Mitigate Network Challenges

To minimize the effect of network delays and failures on consistency, the following approaches can be taken. One such technique is consensus algorithms, which are meant to work while the network is partitioned while at the same time guaranteeing that, in the long run, all nodes will be consistent. The CAP theorem states that distributed systems can only guarantee two properties: Consistency, Availability, and Partition Tolerance (Brewer, 2000). Dynamo and Cassandra, for instance, use the CP model where availability and partition tolerance are valued more than consistency while using eventual consistency to allow convergence of data in different time phases (Lakshman & Malik, 2010). These systems employ this and other methods – such as anti - entropy and read - repair – to slowly bring all the nodes in sync so that all replicas are identical.

The other consistencies use adaptive consistencies, where the consistency models modify their consistency behavior depending on the existing network conditions. For instance, the TAO system of Facebook adapts the degree of consistency of the data in its storage to the degree of the network latency observed from the system: it uses strong consistency for operations that require it and uses eventual consistency for all other operations (Bronson et al., 2013). This allows the system to be highly available and offer high performance while ensuring strong consistency when the network is good. Likewise, Google's F1 system on replication uses a hierarchical replication model in which the data is replicated across regions with varying consistency level needs. Hence, the system achieved consistency, availability, and performance depending on the application type (Shute et al., 2012).

Achieving consistency in the distributed system means addressing various trade - offs related to resource consumption, system performance, or end users' satisfaction with the result. Strong coherence is effective in providing accurate data. It has the demerit of high latency and higher use of resources. Though weaker consistency models are resource - effective, the use of these models may lead to loss of data integrity, especially in systems with high availability requirements. Through the use of adaptive methods and different consistency models, the developers can work on systems that meet all these demands, hence catering to both the efficiency of the systems and the usage by the clients despite network - based challenges.

4. Future Trends and Developments

Emerging Technologies in Consistency Management

Due to evolving technologies, the semantics of consistency in distributed memory systems are expected to undergo revolutionary changes. Distributed memory systems are nowadays paramount in computing extensively in networks of nodes. It is believed that as such systems develop constantly, such aspects as distributed memory and changes in consistency models will be of great importance.

One of the most exciting developments is the system's incorporation of distributed memory. Traditional systems have, therefore, required that all nodes operate in a coordinated manner, and this has proved to be quite resource - demanding and slow, especially where the governing framework is extensive. Newer work, however, concentrates on improving the effectiveness of these systems through various schemas, such as in - memory computing and hardware accelerations (Herlihy & Wing, 1990). Pacemaker technologies lower latency and enhance the speed of calamity, bringing into focus synchronization nodes so that the mass following technique is a more scalable proposition for managing distributed data.

The most significant advancement is creating a new generation of hybrid consistency models. Early notions of consistency models can be distinguished into strong and weak categories with disadvantages of worse performance and data consistency. However, current models are gradually differentiated from the previously described ones and include options that open a much more comprehensive range of possibilities to select the necessary functionality for specific requirements for the application (Tanenbaum & Van Steen, 2007). For instance, models that integrate some of the features of the strong consistency properties with the properties of the eventual consistency models are being evolved to achieve the best of both worlds: consistency means correct value and system response times. These 'twice gentle' young man models may be expected to become more widespread because they provide an adaptable way of standardizing engagement that can be applied selectively.

Potential Shifts in Consistency Models

Self - organizing systems become more effective when distributed computers continue to expand, and traditional consistency management models may need to be revised. One of the earliest and still most influential principles in distributed systems is the CAP theorem, which claims that the system can provide only two of the three properties: Consistency, Availability, and Partition Tolerance (Brewer, 2000). However, newer research and technologies have advocated these, proving that a better balance can be realized at better performance consistency without compromising availability.

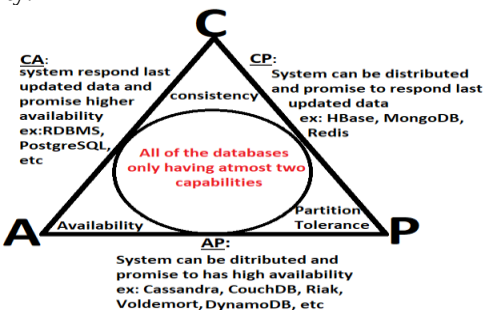

Figure 13: The CAP Theorem in DBMS

The most significant transformation of consistency models observed at the current stage is moving toward adaptive consistency models. These models fluctuate concerning the guarantees of consistency dependent on the state of the system under which the models are being used and the nature of the task in question in recommending them, as per Pritchett (2008). For example, suppose the network utilization is low, or the system deals with transactions not deemed sensitive to timing details. In that case, the system may be run with a level of consistency slightly lower than the final one, which would be desirable. On the other hand, when critical operations are

being carried out, or the signal strength is good, the system might require higher data consistency. This ability is one of the properties that will be characteristic of future distributed systems and will help to deal with the tradeoff between convergence to the consistent state, availability of the system, and its throughput.

Predictions for Scaling Services in the Future

The beauty and the challenge in the growing realm of consistency models and distributed memory systems lie in their impact on the scalability of the services. The degree of scalability of consistency is fundamental in distributed systems in the modern world. As systems get more prominent, the problems with version control across many nodes become significantly different. However, solutions in the technologies and models presented above outline new opportunities for production systems.

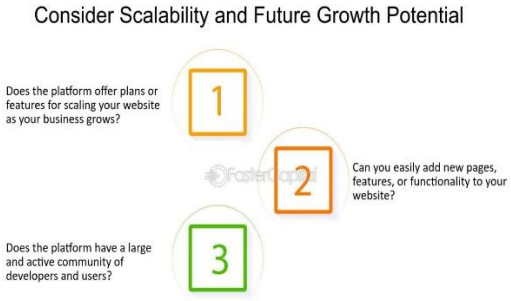

Figure 14: Preparing For Scalability and Future Growth

One major prediction is that higher scalability of distributed systems will be reached with the growth of hybrid and adaptive consistency models. These models enable systems to manage the execution to control correspond to current conditions - this is always critical when managing vast volumes of data and transactions that are the norm in today's large environment (Gray & Reuter, 1993). This enhances efficiency but also reduces the cost of resource use in establishing the consistency required in delivering services, increasing the possibility of expanding services to cater to the demands of future uses.

Implications for Developers and Businesses

Going deeper into the analyses of the changes in consistency management, it is possible to note that it has inevitable consequences for developers and businesses. The tendency towards more freedoms and of the emerging Consistency Models consequently calls for a new conceptualization in system development. Such models will need to be incorporated into the applications by the developers in a way that will address their application needs, and the tradeoff between the performance and consistency of the models will have to be made to satisfy the end user (Shapiro et al., 2011). It may imply that certain basic assumptions regarding design must be readdressed, and new paradigms and toolsets must be employed to support these new models.

From the business needs perspective, the effectiveness and control of service scale and service quality homogeneity across multiple distributed systems are vital factors for enterprises to be competitive in the digital economy. The need to develop sound and scalable distributed systems will augment because more business transactions are online, and real - time data is becoming increasingly popular. Those organizations that can maximize the potential of the newest technologies of consistency management will be at a vantage point to offer dependable and excellent value - added services to their clients and thus have an edge in the marketplace (Bailis et al., 2013).

5. Conclusion

Handling data inconsistency in distributed memory systems is an essential feature of modern computing systems, which, in one way or another, affects the system's performance, scalability, and usability. Logically distributed systems are broadly classified as strongly and weakly consistent, and many of them are challenging as distributed systems scale up in size and complexity. This paper has discussed various techniques and models that are expected to solve these problems with the understanding that to meet the two aspects of the systems – consistency and performance – some balance has to be achieved. Linearizability, for example, ensures data consistency at the highest level without considering the consequences of high resource utilization and possible performance limitations. These models are essential for use where data precision and update time are critical, for example, in compound currency environments and other distributed data archives containing necessary information. Weak consistency models, like eventual consistency, provide a cheaper model for resource use but may have periods of inconsistency. Hence, they should be used in the areas where consistency is not highly desired but rather the availability and performance of the system.

New and unforeseen consistency models, which combine traditional consistency models and can, therefore, be referred to as hybrid and adaptive consistency models, should be seen as future trends in the given domain. These models allow one to have variants of consistency comparable to different effects of client needs and the condition of a system to achieve an optimal balance between performance and reliability. Such adaptability proves beneficial in large - scale distributed systems in which the maintenance of consistency against the provision of performance determines remarkable scalability. As distributed systems advance, it remains challenging to observe that the proper management of inconsistency cannot be underestimated. Consistency models should be chosen and applied according to the specifics of the applications, which will allow the developers to ensure that the systems can be easily scaled and provide the necessary level of consistency. All businesses must exploit these sophisticated consistency management mechanisms to sustain their competitiveness in the prevailing digital and data - intensive environment. As such, more advancements in future research on consistency must be made to address the challenges of growing distributed systems. Further analysis of new models, technologies, and strategies will be crucial for forming the subsequent generation's novel, high - level, efficient distributed systems.

References

[1] Bailis, P., Fekete, A., Hellerstein, J. M., Ghodsi, A., & Stoica, I. (2014). Scalable atomic visibility with RAMP

transactions. ACM Transactions on Database Systems (TODS), 39 (4), 1 - 41.
[2] Bazzi, R. A. (2002). Consistency conditions for multi - object distributed operations. Distributed Computing, 15 (1), 1 - 15.
[3] Brooker, M. (2014). Cap and pacelc: Thinking more clearly about consistency. *Blog, July, 16*.
[4] Brewer, E. (2000). Towards robust distributed systems. Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing, 7 - 10.
[5] Bronson, N., Amsden, Z., Cabrera, G., Chakka, P., Dimov, S., Ding, H.,. . . & Malik, T. (2013). TAO: Facebook's distributed data store for the social graph. USENIX Annual Technical Conference.
[6] Chandra, T. D., Griesemer, R., & Redstone, J. (2007). Paxos made live: an engineering perspective. ACM Symposium on Principles of Distributed Computing.
[7] Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J., & Woodford, D. (2013). Spanner: Google's globally distributed database. ACM Transactions on Computer Systems (TOCS), 31 (3), 1 - 22.
[8] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A.,. . . & Vogels, W. (2007). Dynamo: Amazon's highly available key - value store. ACM SIGOPS Operating Systems Review, 41 (6), 205 - 220.
[9] Gotsman, A., Yang, H., Ferreira, C., & Najafzadeh, M. (2016). Transactional consistency and automaticity in replicated databases. Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data.
[10] Gray, J., & Reuter, A. (1993). Transaction Processing: Concepts and Techniques. Morgan Kaufmann.
[11] Herlihy, M. P., & Wing, J. M. (1990). Linearizability: A correctness condition for concurrent objects. ACM Transactions on Programming Languages and Systems (TOPLAS), 12 (3), 463 - 492.
[12] Hutto, P. W., & Ahamad, M. (1990). Slow memory: Weakening consistency to enhance concurrency in distributed shared memories. Proceedings of the 10th International Conference on Distributed Computing Systems.
[13] Ladin, R., Liskov, B., Shrira, L., & Ghemawat, S. (1992). Providing high availability using lazy replication. ACM Transactions on Computer Systems (TOCS), 10 (4), 360 - 391.
[14] Lakshman, A., & Malik, P. (2010). Cassandra: a decentralized structured storage system. ACM SIGOPS Operating Systems Review, 44 (2), 35 - 40.
[15] Lamport, L. (2001). Paxos made simple. ACM SIGACT News, 32 (4), 18 - 25.
[16] Ongaro, D., & Ousterhout, J. (2014). In search of an understandable consensus algorithm. USENIX Annual Technical Conference.
[17] Pritchett, D. (2008). BASE: An ACID alternative. Queue, 6 (3), 48 - 55.
[18] Shapiro, M., Preguiça, N., Baquero, C., & Zawirski, M. (2011). Conflict - free replicated data types. Proceedings of the 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS).
[19] Shute, J., Vingralek, R., Samwel, B., Handy, B., Rollins, E., Oancea, M.,. . . & Bittman, R. (2012). F1: A distributed SQL database that scales. Proceedings of the VLDB Endowment, 6 (11), 1068 - 1079.
[20] Stonebraker, M. (2010). SQL databases v. NoSQL databases. Communications of the ACM, 53 (4), 10 - 11.
[21] Sumbaly, R., Kreps, J., & Shah, S. (2012). The big data ecosystem at LinkedIn. ACM SIGMOD Record, 40 (2), 52 - 58.
[22] Tanenbaum, A. S., & van Steen, M. (2007). Distributed Systems: Principles and Paradigms (2nd Ed.). Prentice Hall.
[23] Vogels, W. (2009). Eventually consistent. Communications of the ACM, 52 (1), 40 - 44.
[24] Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., & Alonso, G. (2000). Understanding replication in databases and distributed systems. Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS).