Vol 8. No. 3 – September, 2015

African Journal of Computing & ICT

© 2015 Afr J Comp & ICT – All Rights Reserved - ISSN 2006-1781
www.ajocict.net

A Comparative Study of Consistency Theorems in Distributed Databases

C.K. Obasi, P.O. Asagba & A.I. Silas
Department of Computer Science,
University of Port Harcourt
Port Harcourt, Nigeria.
chinedu.obasi@uniport.edu.ng, pasagba@yahoo.com, abasiama_silas@uniport.edu.ng

ABSTRACT

Maintaining consistency in distributed databases requires a user or database administrator's advanced expectation and preparation for failure nodes during the database operation. Though ensuring consistency in a database enhances and ensures database integrity, this integrity could only be easily achieved in centralized databases. Distributed databases requires different mechanisms, theorems and trade-offs to guarantee that consistency can be ensured. This paper discusses and compared several developed theorems which either implements enforced consistency, ensures high-availability consistency or even demonstrates eventual consistency in distributed databases. It describes how several properties of distributed systems are chosen over another in a well-fashioned manner in database designs to maintain consistency in a distributed database.

Keywords: Consistency; Latency; Partition-tolerance; distributed database; CAP; PACELC

African Journal of Computing & ICT Reference Format:
C.K. Obasi, P.O. Asagba & A.I. Silas (2015): A Comparative Study of Consistency Theorems in Distributed Databases.
Afr J. of Comp & ICTs. Vol 8, No. 3. Pp 205-208.

1. INTRODUCTION

A distributed system is a piece of software that serves to coordinate the actions of several computers. This coordination is achieved by exchanging messages, i.e., pieces of data conveying information. The system relies on a network that connects the computers and handles the routing of messages. A distributed system is a system that operates robustly over a wide network. A particular feature of network computing is that network links can disappear, and there are many strategies for managing this type of network segmentation. A distributed database stores a logically related database over two or more physically independent sites and the sites are connected via a computer network.

A centralized database is a database that is maintained, stored and located in a single location. Such locations could be a mainframe computer or a server and can be accessed through a network but every user accesses the same single location.
Consistency is the ability of a system to behave as if the transaction of each user always run in isolation from other transactions, and never fails [7]. Consider for instance a transaction on an e-commerce site. There is a "basket" which is progressively filled with bought items. At the end the user is directed to a secure payment interface. Such a transaction involves many Hypertext Transfer Protocol (HTTP) accesses, and may last an extended period of time (typically, a few minutes). Consistency in this context means that if the user added an item to her basket at some point, it should remain there until the end of the transaction. Furthermore, the item should still be available when the time comes to pay and deliver the product.

In centralized databases, one primary record is maintained because the data is stored in a single location and this makes the data very accurate and highly consistent unlike in distributed databases where the data is stored in multiple locations as such maintaining consistency in all locations will require more complexity. For a distributed database management system, to ensure data consistency across database fragments in the Distributed Databases Management System (DDBMS) and to encourage simultaneous data access, complex mechanisms are required and careful planning on how to partition a database and where to locate the database fragments can help ensure the performance and consistency of a distributed database.

2. CONSISTENCY THEOREMS

There are different theorems designed by developers in the quest for building distributed database systems which will provide maximum performance, maintain consistency and meet the scalability requirements of distributed architectures. This paper discusses details on these theorems to ascertain how they affect the choice of designing distributed database systems.

2.1 CAP Theorem

In a Symposium held on Distributed Computing, [10], [12], proposed a conjecture that "no distributed system can simultaneously provide consistency, availability and partition tolerance. This was later confirmed by [13] as a theorem. The properties gave rise to the acronym CAP (Consistency, Availability, and Partition Tolerance):

a. Consistency
For the nodes of a distributed system to show consistency, all the nodes must show a consistent view of data, meaning the same results is yielded as the system assures that operations have an atomic characteristic and changes are disseminated simultaneously to all nodes [11]. This makes all database clients to see the same data, even with concurrent updates.

b. Availability
The availability property ensures that the database clients can access at any time part of the data. The system must always at the end, process every request, even when failure occurs. This must be true for both read and write operations. This theorem has been confirmed by [11], [13] for unbounded, eventual responses.

c. Partition Tolerance
This property shows that the system continues to operate despite arbitrary message loss. A partition is an arbitrary split between nodes of a system, resulting in complete message loss in between [11].

2.1.2 CAP Architectures
CAP basically states that in building Distributed Database Systems, designers can choose two of three desirable properties: consistency(C), availability (A), and partition tolerance (P). Therefore only three architectures are possible: only CA systems (consistent and highly available, but not partition tolerant), CP systems (consistent and partition-tolerant, but not highly available), and AP systems (highly available and partition-tolerant, but not consistent) are possible [6].
Figure1 describes the CAP Theorem:

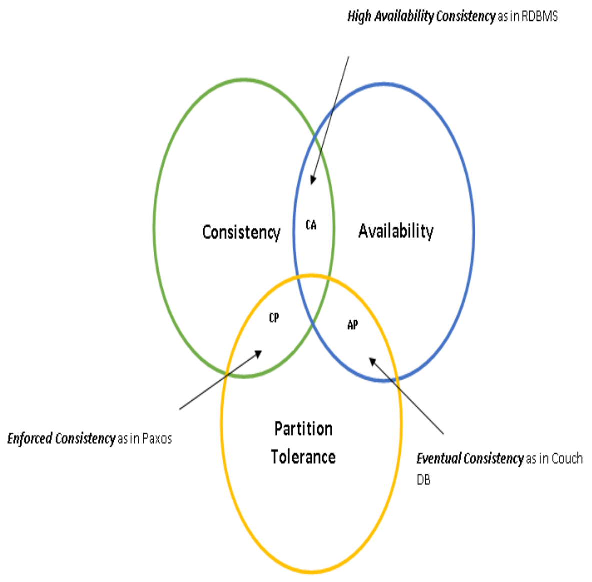
High Availability Consistency as in RDBMS
Consistency
CA
Availability
CP
AP
Enforced Consistency as in Paxos
Partition Tolerance
Eventual Consistency as in Couch DB

Figure 1: CAP Theorem

i. Consistent and Available (CA) Systems
The systems grouped in this architecture ensures that the service of availability and consistency is provided but partitions are not tolerated. When partitions occur, the systems will become inconsistent. The combination is also known as *high-availability consistency*. Most of the traditional relational database management systems use this approach. To achieve high-availability consistency, replication mechanism is important as transaction protocols such as the two-phase commit (2PC) protocol are applied to ensure consistency. The separation into partitions may lead to so-called "split brain" scenarios, in which different partitions, create conflicting replicas as a result of isolation. The system can only recover from such scenarios by using some kind of consensus protocol. This in turn would disallow nodes to service requests unless a consensus is available. We would thus convert our CA approach into a CP approach at the sacrifice of availability. For larger distributed database systems, the CA approach is less suitable because of the shortcomings encountered [11].

ii. Consistent and Partition Tolerant (CP) systems
The combination of consistency and partition tolerant properties provides a strong consistent service in distributed systems. In the presence of a partition, consistency is guaranteed, though if some nodes are temporarily unreachable, it will cause the nodes of a partition not to respond to requests, till an agreement is reached by all. This causes availability not to be always provided. The combination of these properties is also known as *enforced consistency* [11]. In situations where distributed systems needs to be designed and consistency maintained at all costs, the CP approach is the best, for instance in a banking application, where the balance of all accounts is a primary constraint. This model has been found to be implemented in relational database systems. Supporting consistent states even in case of network errors requires the usage of sophisticated algorithms for quorum and majority decisions. Such a protocol for solving consensus is the Paxos protocol [6].

iii. Available and Partition Tolerant (AP) Systems
The AP approach in distributed systems allows availability and tolerates partitions, though this may cause a node to be temporarily inconsistent. The combination of these properties results in *eventual consistency* [11]. Eventual consistency is a model for database consistency in which updates to the database will propagate through the system so that all data copies will be consistent eventually. A well designed distributed system might not appear robust and stable especially when this approach shows that consistency has been sacrificed for availability and partition tolerance, though many applications can favour availability at all costs and tolerate deferred consistency properties. In this case, it is important to keep in mind potential issues due to eventual consistent data on application level during development. Examples of systems that follow this approach are the DNS (Domain Name Systems) or web caches. Stale data (e.g. host mappings respectively cached responses) are acceptable for a

Vol 8. No. 3 – September, 2015
African Journal of Computing & ICT
© 2015 Afr J Comp & ICT – All Rights Reserved - ISSN 2006-1781
www.ajocict.net

while, but eventually the latest version of the data disseminates and flushes older entries [11].

2.2 PACELC Model

The CAP theorem has gained serious criticisms with the rise of the NoSQL (often interpreted as Not only SQL) movement and the increasing interest in eventually consistent data stores. A central issue of the CAP theorem results from the simplifying error model that only targets network failures. It is especially the premature dropping of consistency as the answer to network errors that is raised to question by members of the database community such as Stonebraker [12].

Other fall-shorts of the CAP theorem as mentioned by [3] include the asymmetry of availability and consistency and the generalizing trade-off between consistency and availability. These disadvantages becomes obvious when regarding systems in the absence of partitions. A better way of portraying the space of potential consistency tradeoffs for DDBSs can be achieved by rewriting CAP as PACELC (Partition Availability Consistency Else Latency/Consistency): if there is a partition (P), how does the system trade off availability and consistency (A and C); else (E), when the system is running normally in the absence of partitions, how does the system trade off latency (L) and consistency (C)? As a consequence, systems can now be categorized more precisely [6].

As an example, eventually consistent systems (AP in terms of CAP) can be split up into PA/EL or PA/CL systems, yielding more details on their regular operational mode in the absence of partitions.
Note that the latency/consistency tradeoff (ELC) only applies to systems that replicate data. Otherwise, the system suffers from availability issues upon any type of failure or overloaded node. Because such issues are just instances of extreme latency, the latency part of the ELC tradeoff can incorporate the choice of whether or not to replicate data.

2.2.1 PACELC Architectures

PACELC systems can be subdivided into different types depending on which of the properties the database systems focuses on.

i. Partition-occurs maintain Availability Else Latency (PA/EL) systems

In this type of systems, if a partition occurs, they give up consistency for availability, and under normal operation they give up consistency for lower latency. Giving up both Cs (Consistency) in the PACELC architecture makes the design simpler; once a system is configured to handle inconsistencies, it makes sense to give up consistency for both availability and lower latency. This can be observed in the default versions of these databases namely Amazon's Dynamo, Facebook's Cassandra, and Riak databases [9], [10]. These systems employ eventual consistency as is seen in AP systems of the CAP theorem.

ii. Partition-occurs maintain Consistency Else Consistency (PC/EC) systems

These types of systems will refuse to give up consistency, and will pay the availability and latency costs to achieve the consistency in its database. It can be found in databases with full ACID (Atomicity, Consistency, Isolation, Durability) properties. These database systems include VoltDB/H-Store, MegaStore, BigTable and Hbase.

iii. Partition-occurs maintain Consistency Else Latency (PC/EL) systems

This system cannot be said to be fully consistent, but it can be rather said that the system does not reduce consistency beyond the consistency level when a network partition occurs, instead it reduces availability. This can be seen in the PNUTS database built by Yahoo. The PACELC system can be seen as shown in Figure 2.

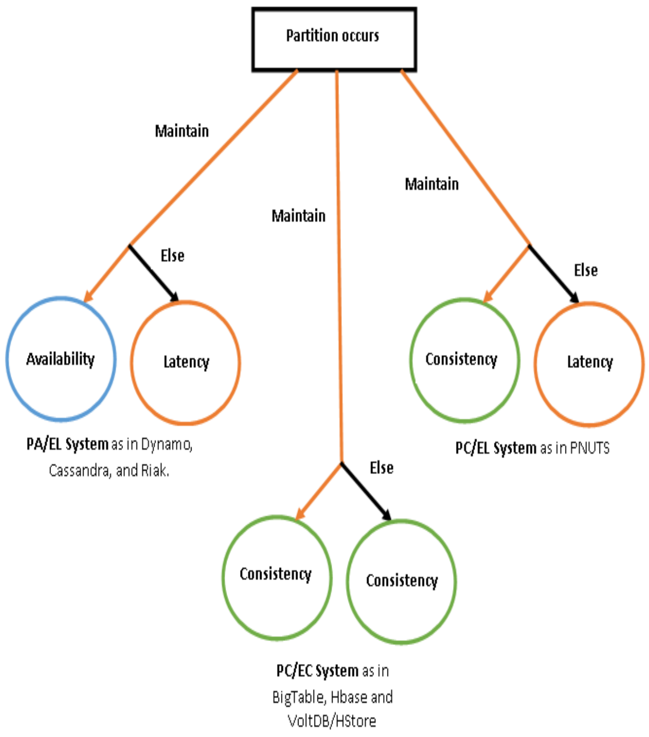

Figure 2: PACELC Model

3. RECOMMENDATIONS

Building and adopting a consistency model that will deliver stronger consistency guarantees will be very vital because there are applications that need to justify the responses they provide to users, such as medical systems that monitor patients and control devices, security systems. This calls for more research.

In as much as these models in some cases implement eventual consistency, full consistency becomes a necessary property in the development of sensitive systems because they cannot at some point base their results on stale or incorrect data.

4. CONCLUSION

It was clearly observed that consistency, availability, and partition tolerance cannot be guaranteed at the same time for a distributed system. In building distributed database systems, the trade-offs considered are so complex that neither CAP nor PACELC can explain them all. It is important to state that bringing in the consistency/latency tradeoffs into the modern design of Distributed Database System Design becomes relevant to building a more robust distributed database systems, and unifying CAP and PACELC into a single formulation can lead to a deeper understanding of modern Distributed Database System designs.

REFERENCES

[1] A. Lakshman and P. Malik , "Cassandra: Structured Storage System on a P2P Network", Proc. 28th ACM Symposium Principles of Distributed Computing (PODC 09), 2009, ACM, vol. 5; doi:10.1145/1582716.1582722.
[2] B.F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz., D. Weaver and R. Yerneni, "PNUTS: Yahoo!'s Hosted Data Serving Platform", Proc. VLDB Endowment (VLDB 08), 2008, ACM, pp. 1277-1288.
[3] D.J. Abadi, "Problems with CAP, and Yahoo's Little Known NoSQL System", DBMS Musings, blog, on-line resource, 2010.
[4] D.J. Abadi, "Consistency tradeoffs in Modern Distributed Database System Design: CAP is only part of the Story", IEEE CS, Issue 2, vol. 45, 2012, pp. 37-42.
[5] E. Brewer, "Towards Robust Distributed Systems", Proc. 19th Ann. ACM Symposium Principles of Distributed Computing. (PODC 00), ACM, 2000, pp. 7-10.
[6] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall and W. Vogells, "Dynamo: Amazon's Highly Available Key-Value Store", Proc. 21st ACM Symposium on Operating Systems Principles (SOSP 07), ACM, October 2007, pp. 205-220.
[7] http://webdam.inria.fr/Jorge/html/wdmch15.html.
[8] K.P. Birman, D.A. Freedman, Q. Huang and P. Dowell, "Overcoming CAP with Consistent Soft-State Replication", IEEE CS, Issue 2, vol. 45, 2012, pp. 50-58.
[9] L. Lamport, "Paxos Made Simple", SIGACT News, Distributed Computing Column, vol. 32, Issue 4, 2001, pp. 51-58.
[10] http://webdam.inria.fr/Jorge/html/wdmch15.html
[11]http://berb.github.io/diplomathesis/original/061_challenge.html
[12] M. Stonebraker, "Errors in Database Systems, Eventually Consistency, and the CAP Theorem", blog. Comm. ACM, 5, 2010.
[13] S. Gilbert and N. Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services", ACM SIGACT News, 2002, pp. 51-59.

Authors' Briefs

Obasi Chinedu Kingsley holds a B.Sc (Hons) in Computer Science from Nnamdi Azikiwe University, Awka, Anambra State, Nigeria, in 2008, a M.Sc in Computer Science at the University of Port Harcourt, Nigeria in 2015. His research interest area includes Machine Learning, Distributed systems, and Cloud Computing. He is a certified professional in international certifications like CCNA, MCTS and STS. He is a member of IEEE and IEEE-Computer Society. He can be reached through chinedu.obasi@uniport.edu.ng.

Prince Oghenekaro Asagba had his B.Sc. degree in Computer Science at the University of Nigeria, Nsukka, in 1991, M.Sc. degree in Computer Science at the University of Benin in April, 1998, and a Ph.D degree in Computer Science at the University of Port Harcourt in March, 2009. He is a Senior Lecturer and a visiting lecturer to several Universities in Nigeria since 2010. His research interest includes: Network Security, Information Security, Network Analysis, Modeling, Database Management Systems, Object-oriented Design, and Programming. He has published several articles in Learned Journals both in Nigeria and Internationally. He has authored and coauthored several books in Computer Science. He is a member of Nigeria Computer Society (NCS) and Computer Professional Registration Council of Nigeria (CPN).

Silas Abasiama Ita holds a B.Sc Hons in Computer Science/Mathematics at the University of Port Harcourt, Rivers State, Nigeria, in 2009, a M.Sc in Computer Science at the University of Port Harcourt, Nigeria in 2015. Her research interest includes Distributed Database/Distributed Processing, Network/Data security, Modeling, Software Engineering and Artificial Intelligence. She is an Associate member of CISCO, member of IEEE and IEEE Computer Society. She can be contacted via abasiama_silas@uniport.edu.ng.