The CAT Theorem and Performance of Transactional Distributed Systems

Shegufta Bakht Ahsan
Department of Computer Science
University of Illinois, Urbana Champaign
sbahsan2@illinois.edu

Indranil Gupta
Department of Computer Science
University of Illinois, Urbana Champaign
indy@illinois.edu

ABSTRACT

We argue that transactional distributed database/storage systems need to view the impossibility theorem in terms of the contention, abort rate, and throughput, rather than via the traditional CAP theorem. Motivated by Jim Gray, we state a new impossibility theorem, which we call the CAT theorem (Contention-Abort-Throughput). We present experimental results from the performance of several transactional systems w.r.t. the CAT impossibility spectrum.

Keywords

CAP; Distributed Transaction; Contention; Throughput

1. INTRODUCTION

NoSQL database systems, in their infancy, supported basic CRUD (Create Read Update Delete) operations at low latency, and with high availability and weak consistency. The weakness of consistency arose from the CAP theorem, which in its original form stated that of three properties considered desirable in database systems–consistency, availability, and partition-tolerance–at most two are achievable simultaneously [19].

The most practical interpretation of this theorem is that a system cannot support both (strong) consistency and (high) availability (or low latency [6]) in a partitionable system. As a result, many NoSQL systems, such as MongoDB, Riak, Dynamo, Cassandra [3, 5, 14, 23], all support weak eventual consistency.

New Consistency Models and New CAPs: A few years ago, stronger models of consistency such as red-blue [25], causal+ [26], etc., started to emerge. Recently, at SOSP 2015, there were six papers that supported strong notions of ACID consistency and transactions. These papers included: 1) systems that layered atop unreliable replication and yet supported transactions with high throughput, e.g., Yesquel [7], Callas [29] and Tapir [30], and 2) systems that supported transactions by relying on strong hardware abstractions, e.g., FaRM [16], RIFL [24] and DrTM [28].

DCC'16, July 25-28, 2016, Chicago, IL, USA
© 2016 ACM. ISBN 978-1-4503-4220-9/16/07…$15.00
DOI: http://dx.doi.org/10.1145/2955193.2955205

Alongside classical transactional databases, the emergence of this new class of transaction-based databases (sometimes called NewSQL systems) has led to some questioning about the validity of the CAP theorem. We first clarify that the CAP theorem, as stated in [19] is correct–it has been formally proved. However, the prevailing discussions in the community have led some researchers to opine that there is a gap between the CAP theorem and the practical needs of today's systems. Some argue that partitions are relatively rare, while others say that availability (or latency) is a fuzzy term and not a "correctness" issue (unlike consistency, which has strong notions such as linearizability and sequential consistency).

As a result, variants of the CAP theorem have started to emerge. One of the popular variants is that the P in CAP ought to stand for performance rather than partition-tolerance [27]. This new notion has been considered by some in the community to be applicable to transaction systems.

Limitations of Transactional Systems: In 1996, Jim Gray, in his famous paper on the dangers of replication [21], argued that when the contention across transactions increases, the rate of transaction aborts also increases. Concretely, Gray showed, via a back of the envelope calculation, that the rate of transaction aborts increases at least proportional to: 1) the square of the TPS (throughput) of the system, and 2) the third to fifth power of the number of actions in the transaction[1]. In other words, scaling up the throughput of a transaction system inherently comes associated with a higher rate of aborts for transactions.

In a different work, Bailis et. al. proposed the notion of highly available transactions, or HAT [8]. While practically useful, this paper did not state an impossibility result, nor measured the behavior of transactional systems w.r.t. an impossibility.

Our Contributions: In this paper, we argue that transaction based database systems need a more practical version of a CAP-like impossibility theorem, one that is focused on realistic metrics like abort rate and throughput. We present a new theorem, called the CAT theorem, whose initials stand for Contention-Abort-Throughput. The theorem and its proof are based on Jim Gray's classical paper, but we feel that this theorem needs to be explicitly stated and reiterated. The CAT theorem neither replaces nor contradicts, but instead sits alongside, both the classical CAP theorem and existing CAP variants for transactions.

To be practical, we explore implications of the CAT theo-

[1] In reality, Gray calculated the rate of deadlocks, but this is a subset of the aborts.

rem for both recent and cloud-based transactional systems. For recent systems, we approached the authors of three SOSP 2015 papers on transactions, focusing on the ones that did not rely on special hardware like RDMA/HTM, in order to obtain their code. Based on the code that was shared with us, and the practicalities of getting those systems running, we were able to perform experiments on one system: Yesquel. Among the cloud-based transactional database systems, we selected: i) Microsoft Azure SQL [4] and ii) Amazon RDS (MySQL) [1]. We present results that show the behavior of these three systems w.r.t. the CAT theorem. We reiterate here that our results should not be seen as a reflection on the performance of our selected system, but as representative of transaction systems in general.

2. THE CAT THEOREM

The CAP theorem was originally intended for CRUD-supporting NoSQL systems. To warm up discussion leading to our CAT theorem, we discuss an analogy between the CRUD/NoSQL world and the transactional/NewSQL world.

First, we argue that abort rates are important to consider in transaction systems because they are the counterpart of unavailability in a NoSQL system. In a NoSQL system, unavailability for a single operation (CRUD) is the lack of an immediate answer for a read/write. As far as the client is concerned, the outcome of this abort/unavailability is no different from a "failed–try again" answer, or a starvation for that try. The client retries the operation, and after some retries there is a chance that the operation will succeed.

In the world of a transactional database, an abort of a transaction has similar semantics. The client is of course welcome to retry the entire transaction. Because transactions are atomic and indivisible operations (just like any individual CRUD operations), we observe that these two scenarios, in the CRUD world and in the transaction world, are equivalent to each other, as far as the client's observed behavior is concerned.

Second, we argue that contention across transactions is also important to consider. In a CRUD system, an immutable database that supports only read operations can trivially support strong consistency. Analogously, in a transactional system, if all transactions are read-only, no transactions will ever abort. When there is contention, the abort rate is likely to rise.

CAT Definitions: Before we state our CAT theorem, we define the C, A, and T terms:

- Contention (C): The Contention Level of a workload indicates how much transactions "overlap" with each other in the objects that they read and write concurrently. This is indicative of the likelihood that a transaction will abort due to conflicts with other transactions. Section 3 proposes a new metric to measure contention, and compares against existing metrics.

- Abort Rate (A): This measures the fraction of submitted transactions (attempts) that are aborted.

- Throughput (T): Throughput is measured as transactions per second supported by the system. This is often reported as TPS, or rate of committed transactions.

We state the CAT theorem as follows:

CAT Theorem: *No transactional database can support arbitrarily high levels of contention while yielding both a zero abort rate as well as a high throughput.*

Proof: The proof follows directly from Jim Gray's paper [21]. Specifically, equations 12 and 14 in the paper state that (for both eager and lazy replication) when there is non-zero contention in the system, the abort rate grows as the square of the throughput, and thus the abort rate cannot be zero.

CAT Possibilities: We further illustrate this theorem intuitively by showing three scenarios that are each *possible*:

- Scenario-I (C and A): By executing one transaction at a time, arbitrarily conflicting transactions (C) can be supported with a zero abort rate (A). However, the throughput stays very low.

- Scenario-II (C and T): By executing all transactions concurrently (resulting in high levels of contention, or C), a high-throughput performance (T) can be achieved. This is the default approach used by many optimistic concurrency control techniques in databases today [22]. Such systems need to track the conflicts across transactions, e.g., by using mechanisms like multi-versioned concurrency control (MVCC) or variants, locks with deadlock detection, etc. This leads to a non-zero abort rate for transactions.

- Scenario-III (A and T): In a system with read-only transactions (zero contention), by executing all transactions concurrently, the highest throughput possible (T) can be achieved. The abort rate is zero (A).

3. MEASURING CONTENTION

In Section 2, we discussed contention (C) as an important parameter in the CAT theorem. However, contention is a non-trivial metric to measure. We explore two ways in which this measurement can be done. This measurement is not needed for transactions to proceed, but is useful later in the paper to show the CAT trade-offs in plots.

To measure contention across transactions, we propose a new simplified metric called *Contention Level*. We then show its relation to the Jim Gray's metric, and also to the actual abort probability.

Consider a set of concurrent transactions. Define *common objects* among this concurrent set as those objects which are accessed by at least 2 transactions in the set. Contention level is defined as the ratio between the number of accesses to such common objects, and the total number of object accesses, in that set. This metric lies in the interval $[0, 1]$. The intuition is that higher Contention Level implies a higher likelihood of aborts in a concurrent set of transactions (here again, like [21] we assume all operations are writes). For example, consider three concurrent transactions T_1, T_2 and T_3 accessing objects respectively: $\{obj_1, obj_2, obj_3\}$, $\{obj_4, obj_2, obj_5\}$, $\{obj_5, obj_6, obj_7\}$. Then the total number of object accesses is 9, total number of common object accesses is 4 (obj_2, obj_5 with 2 accesses each) and Contention Level is calculated as 0.44.

We first compare this against a contention metric we derived based on Jim Gray's model [21]. In this model the database has D keys, transactions are of length η and there

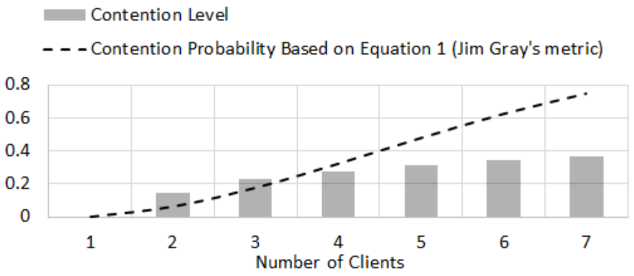

Figure 1: Contention Level Vs Jim Gray based contention metric.

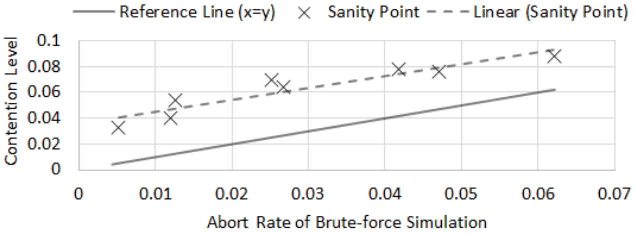

Figure 2: Contention Level Vs Abort Rate (Brute-Force method). For $c = 2$, η varies from 2 to 6; for $c = 3$, η varies from 2 to 4. $\alpha = 0.9$.

are c concurrent transactions. All operations are write operations. A pair of transactions is considered to be contending if they access at least one object in common. Assuming uniform selection of keys across transaction operations, and using combinatorics, we derive Equation 1 as the probability of having at least one pair of contending transactions among a set of c concurrent transactions (this equation did not appear in [21]).

$$P_{contention\_Jim\_Gray} = 1 - \frac{\prod_{i=1}^{c} \binom{D-(i-1)\eta}{\eta}}{\binom{D}{\eta}^{c}} \tag{1}$$

As Figure 1 shows, our Contention Level metric is different from Equation 1 for keys accessed with Zipf distribution with $\alpha = 0.9$ ($\eta = 8$, $D = 1000$, 7 clients, averaged over 100K slots where in a single slot, each client generates a transaction with probability 0.8)–this is because Jim Gray's model does not work for non-uniform access distributions. To further validate our metric, in Figure 2, we calculate the abort probability via the brute-force technique based on the rules for serial equivalence and considering all possible interleavings of transaction operations [13]. This value appears on the X axis (for several values of c, η). The Y value of each (x,y) point refers to our Contention Level metric for the same setting. We observe that while the Contention Level has an error (i.e., is different from x=y line), the linear regression line largely runs parallel to the x=y line. This indicates the error is nearly constant. As a result the rest of the paper only uses the Contention Level metric.

4. EXPERIMENTS

In order to evaluate the behavior in the C-A-T tradeoff space of real transaction systems, we evaluate three transactional database systems: a) Yesquel [7] from VMWare/NYU, b) Microsoft Azure SQL [4] and c) Amazon RDS [1]. We pick the first system because it uses optimistic concurrency control, which is aimed at maximizing throughput[2]. The latter systems were selected as they are well-established cloud-based transactional database systems and have publicly available stable APIs of these two systems.

In reality, we spent a total of about 1 man-year merely attempting to get several optimistic concurrency control systems up and running with benchmarks like YCSB+T [15]. We have been in intense communication with the authors, and all were very responsive, however it has been non-trivial running some of the systems in a sustained way. For example, Tapir [30] from U. Washington, and Hyperdex-Warp [17] from Cornell do not run satisfactorily yet.

Our choice of Yesquel, MS Azure SQL, Amazon RDS and our results on these systems, are not meant as a reflection on the performance of these systems. We reiterate that these results are typical and illustrative of the performance of all systems in this space (though systems may be different).

Below, we first illustrate the three specific areas in the tradeoff space outlined in Section 2, namely CA, CT, and AT. Then we illustrate the effect of increasing contention.

4.1 Experimental Setup

For Yesquel, we run our expriments on *EmuLab* cluster [2] with 3 yesquel servers, and up to 7 clients. We wrote a benchmarking tool in C++, based on YCSB+T [15], a transactional extension of YCSB [12]. Like YCSB, our tool has two different phases: a) load phase: database is initiated and 1000 key-values are inserted. b) run phase: transactions are initiated by clients, limited to 1 concurrent transaction per client; a total of 10K transactions were initiated. For experiments running on MS Azure SQL and Amazon RDS, we wrote the same benchmarking tool in C# and Java respectively. Also, to connect with the servers, initiate database and perform queries, we used publicly available APIs.

The length of each transaction is η operations. The selection of keys in the transaction operation was selected based on the Zipf distribution with α parameter. Larger values of transaction length η, Zipf parameter α, and number of concurrent clients, imply a higher rate of conflicts among transactions. We explore a variety of transactions including Write-Only, Read-Only and Read-Write.

4.2 No Concurrent Transactions

Figure 3 shows the changes in abort rate and normalized average TPS by varying the number of clients (TPS is normalized to the 1 client performance case for each system). We varied the number of clients from 1 to 7. The Zipf coefficient α was set to 0.9 and the transaction length η was set to 8. The experiments were performed on Yesquel, Microsoft Azure SQL and Amazon RDS. Our experiments consist of two types of workloads: i) Write Only transactions, ii) 50% read-50% write transactions, where half the transactions are read only and the rest are write only. The trends for the 50%-50% case were similar to Yesquel for MS SQL and Amazon RDS, and are omitted.

In Figure 4 we present the *aggregate throughput* for Yesquel only (MS Azure SQL and Amazon RDS were similar). This shows that as contention (C) rises, so does abort rate (A) and throughput (T) falls. While this is expected behavior, it is important to confirm this relation empirically.

[2] In comparison, pessimistic concurrency control techniques would have lower throughput and are thus not explored.

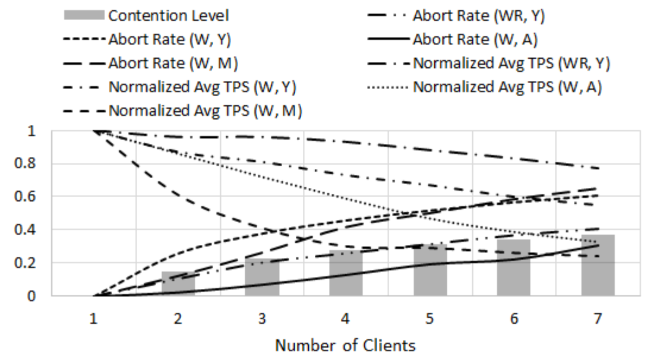

Figure 3: Abort rate (increasing lines) and normalized per client average throughput (decreasing lines) with increasing number of clients ($\alpha = 0.9$, $\eta = 8$, W = Write only transactions, WR = 50% Read-50% Write transactions, Y=Yesquel, A = Amazon RDS, M = MS SQL).

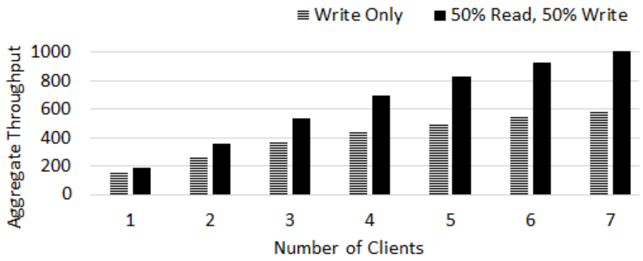

Figure 4: Aggregate throughput (across all clients) with increasing number of clients ($\alpha = 0.9$, $\eta = 8$, transaction type = Write-Only and 50% Read-50% Write, Yesquel).

The left of Figure 3 shows that for any system, when there is no contention C (*number of clients* = 1, thus only 1 transaction at a time), the abort rate A is zero. However, Figure 4 shows that in Yesquel, for both types of transactions (Write-Only and 50% Read - 50% Write) the aggregate throughput T at single client is low compared to when there is more concurrency. This single client case illustrates the CA scenario outlined earlier (Section 2).

4.3 Concurrent Transactions

The traditional way to increase throughput T is to increase the concurrency. Figure 4 shows that, in Yesquel, with an increase in number of clients, the aggregate throughput does indeed increase, however, this comes at the expense of an increased abort rate A (Figure 3). Figure 3 also depicts that, as number of client increases, the Contention Level increases. With 6 clients, the Contention Level was 0.34 for Write-Only transactions. The abort rate of the 50% Read-50% Write transaction was as high as 36% and for Write-Only transaction it was 58%. This illustrates our second scenario (CT) outlined in Section 2.

We also noticed that while the *aggregate* throughput increased with concurrency (Figure 4), the *per-client* throughput actually *decreased* as the number of client was increased (Figure 3). This illustrates a subtle behavior that we believe may be true of many transaction systems.

4.4 Zero Contention

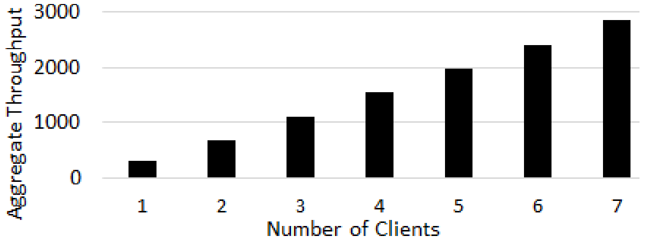

Figure 5: Aggregate throughput with increasing number of clients ($\alpha = 0.9$, $\eta = 8$, transaction type = Read-Only, Yesquel).

Figure 5 illustrates the AT scenario of the tradeoff space. When all transactions are Read-Only, there are no contentions, hence the abort rate is zero. Further, the throughput is significantly higher than when there were writes. For instance, for Yesquel, at 7 clients, the throughput is almost 3000 TPS, which is significantly higher than the 600 TPS from 7 clients when transactions contain write operations only (Figure 4). To illustrate this further, Figure 6 shows the ratio of the throughput in the Write-Only case to the Read-Only case of Yesquel–as contention increases (due to concurrency), the Write-Only transaction scenario degrades significantly compared to the Read-Only transaction scenario. We also tried Read-Write transaction mixes, and the conclusions were similar.

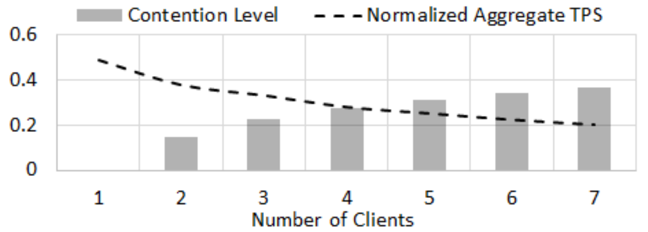

Figure 6: Aggregate throughput of Write-Only transactions (normalized to Read-Only transactions) with increasing number of clients ($\alpha = 0.9$, $\eta = 8$, Yesquel).

4.5 Effect of Contention due to Transaction Overlap

Contention can be increased both by increasing concurrency and by increasing the "overlap" among transactions. We explored the former earlier (Figure 3). Here, we hold the concurrency constant and instead increase the overlap among transactions.

Figure 7 shows the changes in abort rate and normalized average TPS (normalized to $\alpha = 0.0$ for each system) by varying the Zipf coefficient α from 0.0 to 1.0. There were a total of 4 clients in the system, $\eta = 8$ and all transaction were Write-Only. As the Zipf coefficient increased, it selected the most popular keys with high probability, as a result the Contention Level in the system increased (represented by the increasing bars in Figure 7). This leads to a higher abort rate and decreases the TPS.

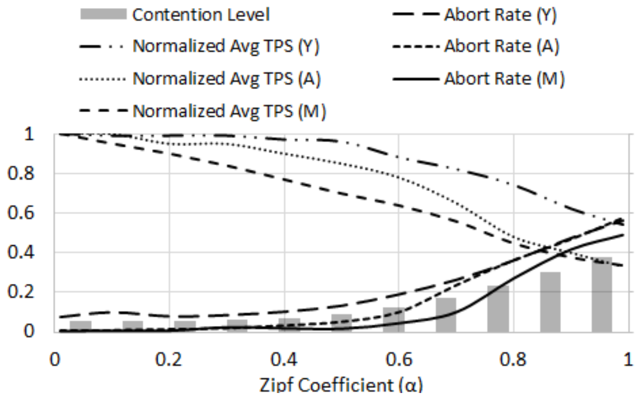

Figure 7: Abort rate and normalized aggregate throughput with increasing Zipf coefficient α (4 clients, η = 8, transaction type = Write-Only, Y=Yesquel, A = Amazon RDS, M = MS SQL).

4.6 Effect of Transaction Length η

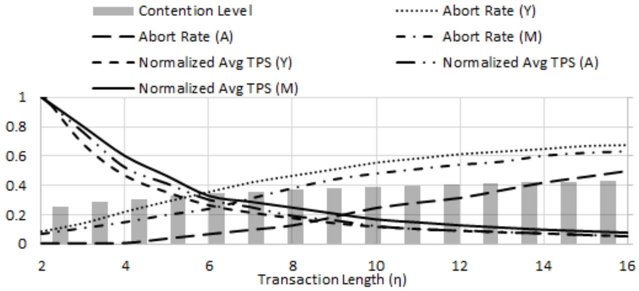

Figure 8: Abort rate and normalized average transaction per second with increasing transaction length η (4 clients, α = 0.9, Write-Only transactions, Y=Yesquel, A = Amazon RDS, M = MS SQL).

Figure 8 shows the changes in abort rate and normalized average TPS (normalized to $\eta = 2$ for each system) by varying the transaction length η from 2 to 16. There were a total of 4 clients in the system, $\alpha = 0.9$, and all transactions were Write-Only. As expected, increased transaction length leads to significantly higher abort rates. The aggregate and average throughput drops because transactions take longer to complete, and thus fewer transactions are being injected into the system (with a fixed number of clients).

5. RELATED WORK

The CAP theorem was proposed by Eric Brewer [10, 11], and later formally proved by Gilbert and Lynch [19]. After over a decade since its introduction, in [9] Brewer argued that the “2 of 3” formulation of CAP was always misleading. First, as partitions are rare in modern systems, there is little reason to forfeit C or A when the system is not partitioned. Second, in a system the choice between C and A can vary–the choice may be different for different sub-systems, for different operations, or even for a specific data or user involved. Our three CAT properties are more continuous than CAP, which tends to be a binary choice.

In [20] the authors review the CAP theorem and situate it within the broader context of distributed computing theory. The authors argue that traditionally there are two types of systems: i) systems that guarantee strong consistency and provide best effort availability, and ii) systems that guarantee availability and provide best effort consistency. The paper argues that there exists a third type of system which sacrifices both consistency and availability and thus they may achieve a trade-off better suited for the application at hand.

In [8] authors present the notion of highly available transactions (HAT) where they show that its possible to achieve many of the transactional guarantees of today’s database without sacrificing high availability and low latency.

In [18], the authors propose the existence of the three way fairness-isolation-throughput (FIT) trade-off in distributed database systems. Here fairness ensures that all transactions will be treated equally and the system will not prioritize or delay certain transactions. According to this trade-off, a distributed database system can pick only two of these three properties. Here the notion of the throughput is the same as in our proposed CAT theorem.

In [6], authors explain how the well known CAP theorem has become increasingly misunderstood and misapplied potentially causing significant harm. Even in the course of normal operation, a system that strictly follows the CAP theorem imposes unnecessary constraints. In the paper authors unify CAP and consistency/latency tradeoff into a single formulation *PACELC*: “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 tradeoff latency (L) and consistency (C)?”

6. ACKNOWLEDGMENTS

We thank Marcos K. Aguilera, Irene Zhang, Naveen Kr. Sharma, Robert Escriva and Emin Gün Sirer, the authors of Yesquel, Tapir and Hyperdex systems, for their quick and helpful responses. This work was supported in part by the following grants: NSF CNS 1319527, NSF CCF 0964471, AFOSR/AFRL FA8750-11-2-0084, and a generous gift from Microsoft.

7. AVAILABILITY

Our benchmarking tool is siliar to YCSB+T and written in C++, C# and Java for Yesquel, MS Azure SQL and Amazon RDS respectively. All the benchmarking tools can be downloaded from the following link:
https://gitlab.com/shegufta/CAT_PROJECT_PUBLIC

8. CONCLUSION

Transaction-based database systems require a practical version of an impossibility theorem. In this paper we proposed a new impossibility theorem called CAT (contention-abort-throughput), which shows that in a transaction-based database system, it is impossible to satisfy the three properties at the same time. We characterized the behavior of both a recently proposed transactional database system and two cloud-based transactional database systems in the C-A-T trade-off space. Our core results are inspired by Jim Gray’s work, but we believe this theorem and its implications for developers and programmers, need to be stated explicitly.

We hope that our paper encourages researchers to envision CAP-like variants for transactional systems. The interplay of CAT and CAP theorems is also an interesting topic to explore. The CAT theorem might also have applications in transactional shared-memory models.

9. REFERENCES

[1] Amazon RDS MySQL. https://aws.amazon.com/rds/mysql/. Last accessed May 2016.

[2] Emulab. https://www.emulab.net/. Last accessed May 2016.

[3] MongoDB. https://www.mongodb.org/. Last accessed May 2016.

[4] Ms azure SQL. https://azure.microsoft.com/en-us/services/sql-database/. Last accessed May 2016.

[5] Riak. http://basho.com/products/. Last accessed May 2016.

[6] Abadi, D. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *IEEE Computer 45*, 2 (2012), 37–42.

[7] Aguilera, M. K., Leners, J. B., and Walfish, M. Yesquel: Scalable SQL storage for web applications. In *Proceedings of the 25th Symposium on Operating Systems Principles* (2015), SOSP '15.

[8] Bailis, P., Davidson, A., Fekete, A., Ghodsi, A., Hellerstein, J. M., and Stoica, I. Highly available transactions: Virtues and limitations. *Proc. VLDB Endow. 7*, 3 (Nov. 2013), 181–192.

[9] Brewer, E. CAP twelve years later: How the "rules" have changed. *IEEE Computer 45*, 2 (2012), 23–29.

[10] Brewer, E. A. A certain freedom: thoughts on the CAP theorem. In *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC'10*.

[11] Brewer, E. A. Towards robust distributed systems (invited talk). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing* (2000), PODC '00, ACM.

[12] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. Benchmarking cloud serving systems with YCSB. In *Proceedings of the First ACM Symposium on Cloud Computing* (2010), SoCC '10, ACM, pp. 143–154.

[13] Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. *Distributed Systems: Concepts and Design*, 5th ed. Addison-Wesley Publishing Company, 2011.

[14] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. Dynamo: Amazon's highly available key-value store. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007*.

[15] Dey, A., Fekete, A., Nambiar, R., and Röhm, U. YCSB+T: benchmarking web-scale transactional databases. In *Workshops Proceedings of the Thirtieth International Conference on Data Engineering Workshops, Chicago, IL, USA, March 31 - April 4* (2014), ICDE '14, pp. 223–230.

[16] Dragojević, A., Narayanan, D., Nightingale, E. B., Renzelmann, M., Shamis, A., Badam, A., and Castro, M. No compromises: Distributed transactions with consistency, availability, and performance. In *Proceedings of the 25th Symposium on Operating Systems Principles* (2015), SOSP '15.

[17] Escriva, R., Wong, B., and Sirer, E. G. Warp: Lightweight multi-key transactions for key-value stores. *Computing Research Repository*.

[18] Faleiro, J. M., and Abadi, D. J. FIT: A distributed database performance tradeoff. *IEEE Data Eng. Bulletin 38*, 1 (2015), 10–17.

[19] Gilbert, S., and Lynch, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM Special Interest Group on Algorithms and Computation Theory 33*, 2 (June 2002), 51–59.

[20] Gilbert, S., and Lynch, N. A. Perspectives on the CAP theorem. *IEEE Computer 45*, 2 (2012), 30–36.

[21] Gray, J., Helland, P., O'Neil, P. E., and Shasha, D. The dangers of replication and a solution. In *Proceedings of the 1996 ACM Special Interest Group on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996.* (1996), SIGMOD '96, pp. 173–182.

[22] Kung, H. T., and Robinson, J. T. On optimistic methods for concurrency control. *ACM Transaction on Database Systems 6*, 2 (1981), 213–226.

[23] Lakshman, A., and Malik, P. Cassandra: a decentralized structured storage system. *Operating Systems Review 44*, 2 (2010), 35–40.

[24] Lee, C., Park, S. J., Kejriwal, A., Matsushita, S., and Ousterhout, J. Implementing linearizability at large scale and low latency. In *Proceedings of the 25th Symposium on Operating Systems Principles* (2015), SOSP '15, ACM, pp. 71–86.

[25] Li, C., Porto, D., Clement, A., Gehrke, J., Preguiça, N., and Rodrigues, R. Making geo-replicated systems fast as possible, consistent when necessary. In *Proceedings of the Tenth USENIX Conference on Operating Systems Design and Implementation* (2012), OSDI'12.

[26] Lloyd, W., Freedman, M. J., Kaminsky, M., and Andersen, D. G. Don't settle for eventual: Scalable causal consistency for wide-area storage with COPS. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (2011), SOSP '11, ACM, pp. 401–416.

[27] Sirer, E. G. Why CAP is flawed and what this means for next generation data stores. LADIS, 2015.

[28] Wei, X., Shi, J., Chen, Y., Chen, R., and Chen, H. Fast in-memory transaction processing using RDMA and HTM. In *Proceedings of the 25th Symposium on Operating Systems Principles* (2015), SOSP '15, ACM, pp. 87–104.

[29] Xie, C., Su, C., Littley, C., Alvisi, L., Kapritsos, M., and Wang, Y. High-performance ACID via modular concurrency control. In *Proceedings of the 25th Symposium on Operating Systems Principles* (2015), SOSP '15, ACM, pp. 279–294.

[30] Zhang, I., Sharma, N. K., Szekeres, A., Krishnamurthy, A., and Ports, D. R. K. Building consistent transactions with inconsistent replication. In *Proceedings of the 25th Symposium on Operating Systems Principles* (2015), SOSP '15.