



LEEDS
BECKETT
UNIVERSITY

Citation:

Gorbenko, A and Karpenko, A and Tarasyuk, O (2020) Analysis of Trade-offs in Fault-Tolerant Distributed Computing and Replicated Databases. In: 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT). IEEE. ISBN 978-1-7281-9958-0, 978-1-7281-9957-3 DOI: <https://doi.org/10.1109/dessert50317.2020.9125078>

Link to Leeds Beckett Repository record:

<http://eprints.leedsbeckett.ac.uk/id/eprint/7186/>

Document Version:

Book Section

© 2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

The aim of the Leeds Beckett Repository is to provide open access to our research, as required by funder policies and permitted by publishers and copyright law.

The Leeds Beckett repository holds a wide range of publications, each of which has been checked for copyright and the relevant embargo period has been applied by the Research Services team.

We operate on a standard take-down policy. If you are the author or publisher of an output and you would like it removed from the repository, please [contact us](#) and we will investigate on a case-by-case basis.

Each thesis in the repository has been cleared where necessary by the author for third party copyright. If you would like a thesis to be removed from the repository or believe there is an issue with copyright, please contact us on openaccess@leedsbeckett.ac.uk and we will investigate on a case-by-case basis.

Analysis of Trade-offs in Fault-Tolerant Distributed Computing and Replicated Databases

Anatoliy Gorbenko^{1,2}

¹Leeds Beckett University
Leeds, UK

A.Gorbenko@leedsbeckett.ac.uk

Andrii Karpenko

²National Aerospace University
Kharkiv, Ukraine

A.Karpenko@student.csn.khai.edu

Olga Tarasyuk

²National Aerospace University
Kharkiv, Ukraine

O.Tarasyuk@csn.khai.edu

Abstract—This paper examines fundamental trade-offs in fault-tolerant distributed systems and replicated databases built over the Internet. We discuss interplays between consistency, availability, and latency which are in the very nature of globally distributed systems and also analyse their interconnection with durability and energy efficiency. In this paper we put forward an idea that consistency, availability, latency, durability and other properties need to be viewed as more continuous than binary in contrast to the well-known CAP/PACELC theorems. We compare different consistency models and highlight the role of the application timeout, replication factor and other settings that essentially determine the interplay between above properties. Our findings may be of interest to software engineers and system architects who develop Internet-scale distributed computer systems and cloud solutions.

Keywords—distributed system, replication, trade-offs, consistency, availability, latency, durability, energy-efficiency

I. INTRODUCTION

Internet-scale distributed computer systems are now extensively used in business-critical applications and critical infrastructures. In such application domains, system failures affect businesses and people's lives. Thus, ensuring dependability of distributed computer systems is a must, as well as a challenge. However, by their nature, large-scale distributed systems running over the Internet are subject to components failures, packets loss, network disconnections, congestions and other accidents.

High availability requirements for many Internet applications call for the use of data replication and system redundancy. Traditional fault tolerance mechanisms such as N-modular, cold- and hot-spare redundancy usually rely on a synchronous communication between replicated components. This suggests that system replicas are synchronized over a short and well-predicted amount of time [1]. This is a reasonable assumption for embedded applications and for those distributed computer systems which components are located, for instance, within the same data center or in the same local area network. However, this does not apply to globally distributed computer systems, which replicas are deployed across the Internet and their updates cannot be propagated immediately. This circumstance makes it difficult to guarantee consistency across replicas.

The Internet and globally distributed computer systems are characterized by a high level of uncertainty of network delays. This makes it almost impossible to guarantee that network messages will always be delivered between system components within a certain time. It has been previously shown that there is a significant uncertainty of response time in service-oriented systems invoked over clouds and the Internet [2]. Besides, in our practical experiments and as

discussed by other researchers [3, 4, 5], failures occur regularly on the Internet, clouds and in scale-out data center networks.

When system architects employ replication and other fault tolerant techniques for the Internet- and cloud-based systems, they have to care about additional delays and their uncertainty and also understand energy and other overheads. Besides, maintaining consistency between replicas is another important issue that needs to be addressed in fault-tolerant distributed computing and replicated data storages.

Maintaining several system replicas inevitably increases the overall energy, consumed by the system. The bigger the replication factor, the higher availability can be ensured at higher energy costs. Moreover, necessity of data synchronization between replicas to guaranty their consistency causes additional energy overheads spent for parallel requests processing and transferring larger amount of data over the network.

This paper examines fundamental trade-offs between system latency, durability and energy consumption in addition to Consistency (C), Availability (A) and Partition tolerance (P) properties, as described by the CAP theorem [6]. In this work we put forward an idea that these properties need to be viewed as more continuous than binary. Understanding trade-offs between them will allow systems developers to predict system response time depending on the used replication factor and/or the selected consistency level. Besides, it will enable balancing availability, durability and/or consistency against latency, power consumption and other properties.

The rest of the paper is organized as follows. In Sections 2 we discuss fundamental CAP and PACELC theorems and their qualitative implications. Section 3 discusses different consistency models and examines trade-offs in fault-tolerant distributed computing and replicated databases between CAP properties, latency, durability and energy consumption. Finally, we draw our conclusions in Section 4 where the role of the application timeout, replication factor and other settings that essentially determine the interplay between above properties is also discussed.

II. CAP AND PACELC THEOREMS AND THEIR IMPLICATIONS

The CAP theorem [6], first appeared in 1998-1999, defines a qualitative trade-off between system Consistency, Availability, and Partition tolerance. It declares that the only two of the three properties can be preserved at once in distributed replicated systems.

Gilbert and Lynch [7] consider the CAP theorem as a particular case of a more general trade-off between consistency

and availability in unreliable distributed systems propagating updates eventually over time.

Internet-scale distributed systems cannot avoid partitions happened due to network failures and congestions, arbitrary message losses and components crashes and, hence, have to choose between strong consistency or high availability.

Thus, a distributed system working over the unreliable and unpredictable network environment has to sacrifice one of these properties. Failing to achieve consistency (i.e. receive responses from all replicas) within the specified timeout causes a partition of the replicated systems.

Timeout settings are of great importance in the context of the CAP theorem. If timeout is less than the typical response time, a system will likely enter a partition mode more often [8]. When a system detects a partition (e.g. when one of its replica did not respond before the time out) it has to decide whether to return a possibly stale result to a client or to reply with an exception message notifying about service unavailability. Thus, the CAP theorem suggests the following three types of systems:

- CA, e.g. traditional ACID-oriented RDBMS systems (MySQL, MS SQL, Oracle, PostgreSQL, etc.) which preserve Consistency and Availability;
- AP, NoSQL databases such as Cassandra, Riak, CouchDB, Voldemort, Dynamo which relax Consistency in favor of Availability and Partition tolerance;
- CP, NoSQL databases such as HBase, MongoDB, Redis, BigTable, MemcacheDB, which preserve Consistency when Partitioned.

The PACELC [9] theorem is a further refinement of the CAP. It suggests that that in case of *partitioning* (P) of a distributed computer system, one has to choose between *availability* (A) and *consistency* (C), *else* (E) in the absence of partitions the replicated systems still face a trade-off between *latency* (L) and *consistency* (C). The PACELC theorem defines the following types of distributed replicated systems:

- PC/EC, e.g. BigTable, HBase, VoltDB/H-Store, Megastore;
- PC/EL, e.g. PNUTS;
- PA/EL, e.g. Dynamo, Cassandra, and Riak;
- PA/EC, e.g. MongoDB.

However, a notion of *Partition* is not well defined in the PACELC as well. For instance, PC (PA) does not indicate that the system is fully consistent (available). It should rather be interpreted as ‘*if partitioning happens it causes more consistency (availability) issues than availability (consistency) issues*’.

Though CAP and PACELC theorem helps the developers do understand the system trade-offs between consistency and availability/latency, there are no methods available that allow trading-off consistency against availability and latency in a quantitative way. Besides, the CAP and PACELC theorems do not take into account other fundamental trade-offs, e.g. between durability and latency, energy consumption and consistency, etc.

III. TRADE-OFFS IN FAULT-TOLERANT DISTRIBUTED COMPUTING AND REPLICATED DATABASES

In this paper we put forward an idea that CAP/PACELC properties need to be viewed as more continuous than binary.

Indeed, the *availability* is measured as usual between 0% and 100%. *Latency* (response time) can theoretically vary between zero and infinity. Though, in practice it is restricted from the right by the application timeout and from the left by some minimal response time higher than zero. Consequently, the replica’s timeout defines system *partitioning*.

Consistency is also a continuum, ranging from weak consistency at one extreme to strong consistency on the other, with varying points of eventual consistency in between.

In this section we describe different consistency models and discuss trade-offs between core properties of distributed storage systems.

A. Consistency Models and Levels

Data consistency models define a contract between a replicated data store and its users, in which the data store specifies guarantees on the results of read and write operations in the presence of concurrent users’ requests.

There are two major groups of consistency models [10]: the first one guarantees data store state for all users (data-centric models); the second one provides guarantees only for an individual user (client-centric models) while the data seen can be varying from users to users. There is a variety of different consistency models used in distributed computing and storage systems [10] which are structured in Fig. 1.

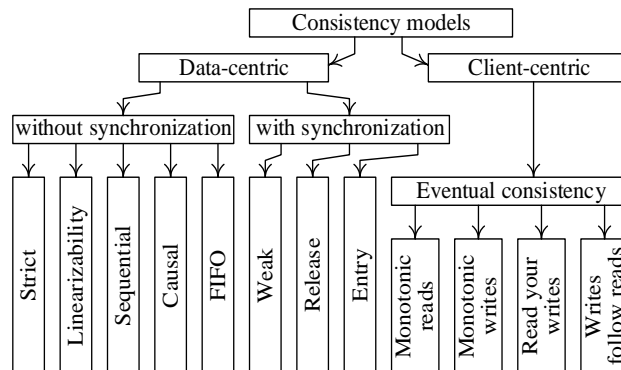


Fig. 1. The CAP trade-offs

The strong consistency cannot be efficiently achieved in replicated systems. Thus, many distributed database systems implements different kinds of relaxed/eventual consistency models.

Unfortunately, there are no general rules for relaxing consistency. As a result, different vendors implement different consistency models, which are hardly compatible and are difficult to match.

For example, *Apache Cassandra* defines a discrete set of consistency levels specifying a number of replicas queried in each read and write operation (see Table I). The strong consistency guaranteeing that a read will always reflect the most recent write is achieved when a sum of nodes written and nodes read is higher than data replication factor.

TABLE I. APACHE CASSANDRA CONSISTENCY MODEL

Consistency level	Definition and consistency guarantee
ONE	Data must be written to the commit log and <i>memtable</i> of at least one replica node before acknowledging the write operations to a client; when reading data, Cassandra queries and returns a response from a single replica (the nearest replica with the least network latency); the strong consistency is guaranteed when a sum of nodes written and nodes read is higher than data replication factor.
TWO	Data must be written to at least two replica nodes before being acknowledged; read operations will return the most recent record from two of the closest replicas (the most recent data is determined by comparing timestamps of records returned by those two replica)
THREE	Similar to TWO but for three replicas
QUORUM	A quorum of nodes needs to acknowledge the write or to return a response for a read request; a quorum is calculated by rounding down to a whole number the following estimate: $replication_factor/2+1$
ALL	Data must be written to all replica nodes in a cluster before being acknowledged; read requests return the most recent record after all replicas have responded. The read operation will fail even if a single replica does not respond
EACH_QUORUM, LOCAL_QUORUM, LOCAL_ONE	Additional consistency levels which become available if Cassandra runs across multiple data centres

TABLE II. MONGODB CONSISTENCY MODEL

Consistency level	Definition and consistency guarantee
w:0, j:false	The weakest consistency setting (writes can be lost even without partition) which provides the lowest latency
w:1, j:false	Writes are guaranteed onto disk of the primary replica; this provides very low latency but very weak consistency
w:2, j:false	Writes are guaranteed on the primary replica's disk and in the memory of one of the secondary replicas; this provides low latency and low consistency
w:2, j: true	Writes are guaranteed on the disks of primary replica and one of the secondary replicas; this provides medium latency and consistency
w:majority, j: false	Writes are guaranteed on the primary replica's disk and in the memory of a majority of secondary replicas; this provides medium latency and consistency
w:majority, j: true	Writes are guaranteed on the disks of primary replica and a majority of secondary replicas; this provides high latency and consistency

TABLE III. AZURE COSMOS DB CONSISTENCY MODEL

Consistency level	Definition and consistency guarantee
STRONG (Reads: local minority; Writes: global majority)	Strong consistency offers a linearizability guarantee, e.g. the reads are guaranteed to return the most recent committed writes with a zero staleness window.
BOUNDED STALENESS (Reads: local minority; Writes: local majority)	It is guaranteed that reads never see out-of-order writes. Though, reads might lag behind writes by at most K updates of a record or by T time interval (i.e. a staleness window) whichever is less.
SESSION (Reads: single replica with session token; Writes: local majority)	Within a single client session it is guaranteed that reads never see out-of-order writes; monotonic reads, writes, write-follows-reads and read-your-writes are also guaranteed; For other client sessions
CONSISTENT PREFIX (Reads: single replica; Writes: local majority)	It is guaranteed that read never see out-of-order writes or writes with gaps; e.g. it is guaranteed to observe an ordered sequence of writes (starting with the first one) that stored at the master replica at some time in the past; more recent writes can be missed.
EVENTUAL (Reads: Single replica; Writes: local majority)	There is no ordering guarantee for reads. In the absence of any further writes, the replicas eventually converge; users may read the values that are older than the ones it had read before.

MongoDB's consistency model is based on tuning w (write concern) and j (journaling) parameters as shown in Table II. Writes are always replicated asynchronously from primary to secondary replicas. By default, reads are done on primary replica. Enabling reads from secondaries makes MongoDB always eventually consistent.

Azure Cosmos DB is a cloud-based storage system which can be globally distributed across multiple Azure regions. It can be configured for a single- (single write region; by default) or multi-master replication and supports five consistency levels described in Table III.

B. Trade-offs Between Consistency, Availability and Latency

System consistency, availability, partition and latency are tightly connected. A system is considered as partitioned when some of its part does not respond until timeout due to arbitrary message loss or delay or replica failure. Availability can be interpreted as a probability that each request eventually receives a response. Though, in many real systems a response that is too late (i.e. beyond the application timeout) is treated as a failure.

High latency has an undesirable effect for many interactive web applications. In [11] authors showed that if a response time increases even as small as 100 ms it dramatically reduces the probability that a customer will continue to use the system.

A system or its replica can be considered as unavailable if the actual response time exceeds the application time out. I.e. a partition can be considered as a time bound on replica's response time. Thus, in term of CAP a slow network connection or slow responding replica can cause a decision that the system is partitioned.

Nowadays, architects of distributed database management systems and large-scale web applications like Facebook, Twitter, etc. often decide to relax consistency requirements by introducing asynchronous data updates in favour of high system availability and low response time. However, the most promising approach is to try to balance these properties with regards to the desired latency and required consistency.

Our interpretation of the CAP and PACELC theorems and the trade-offs resulting from them is depicted in Fig. 2. The consistency level in this model determines the number of replicas which are invoked simultaneously to return the adjudicated (consistent) result to a client application (pretty

similar to Cassandra consistency model; see Table I). In particular, the following settings are possible:

- ONE (equivalent of a hot-spare redundancy) – when the FASTEST response is received the system forwards it to the client. This is the weakest consistency level though it guarantees the minimal latency.
- ALL (equivalent of a N-modular redundancy) – the system must wait until ALL replicas return their responses. In this case the response time is constrained by the slowest replica though the strongest consistency is provided.
- QUORUM (equivalent of two-out-of-three or majority systems) – the system must wait for the responses from a QUORUM of replica web services. It provides a compromise between the ONE and ALL options trading off latency versus consistency.

If minimum latency is the top priority, users should consider a weaker consistency level, e.g. ONE when the only one replica node is requested to respond to a read or write request. If one intentionally decides to give up system consistency its latency could be further improved by requesting more replicas at once than it is necessary. For instance, all replica nodes can be requested but only the fastest response will be returned to a client without waiting for other replica responses that should be ignored.

If consistency is the top priority, users should opt for a stronger consistency setting, which, however, will worsen system latency. One can ensure the strong consistency when a read will always reflect the most recent write by using the following equation:

$$(nodes_written + nodes_read) > replication_factor \quad (1)$$

For example, QUORUM consistency level used for both write and read operations always ensures the strong read consistency, trading off between reads and writes latencies. The strong read consistency is also provided when the ALL consistency level is used to read data while the ONE consistency level is used to write them and vice versa. In the first case the preference is given to minimizing writes latency, while in the second case the minimum reads latency is ensured.

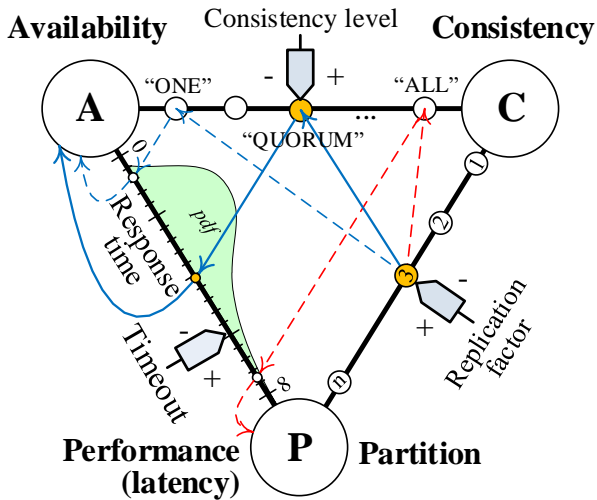


Fig. 2. The CAP/PACELC trade-offs

Application timeout is considered as a bound between system availability and partition/performance (in term of latency or response time) [12]. Thus, system designers should be able to set up timeouts according to the desired system response time also keeping in mind a choice between consistency and availability.

C. Trade-offs Between Performance, Consistency and Durability

Durability is the ability of a system to keep the committed data consistent after crashes, drive failures, power outages, or other forms of corruption. Storing data in memory reduce querying time and provides more predictable and faster performance than storing data on a hard or solid state drive.

Many NoSQL databases store data in memory which improves performance giving up durability. Even traditional RDBMS like PostgreSQL, Oracle, MS SQL or MySQL can be configured to enable periodic durable commits or in-memory data storage.

Durability of such systems is ensured by *transaction/commit logging*, which records changes to the database in a journal file. It is used for in-memory data recovery in case of power loss or rollback operation. However, a database can be configured to store the commit/transaction log in memory and flush it to disk only periodically which improves performance however creates so called ‘data loss window’ [13].

For example, Cassandra NoSQL flushes commit log to disk every 10 seconds by default. Thus, if a power outage happens right before writing the commit log to disk a system could potentially lose up to ten seconds of data.

Nevertheless, *transaction/commit logging* does not prevent data loss in the event of a disk failure. If this happens, lost data could be restored on the crashed node (when it is being repaired) only if the system replicates data across multiple nodes (e.g. a replication factor is above one).

Thus, a higher replication factor improves durability property of a system. However, it affects system consistency and latency depending on the chosen consistency level, as it is shown in Fig. 3.

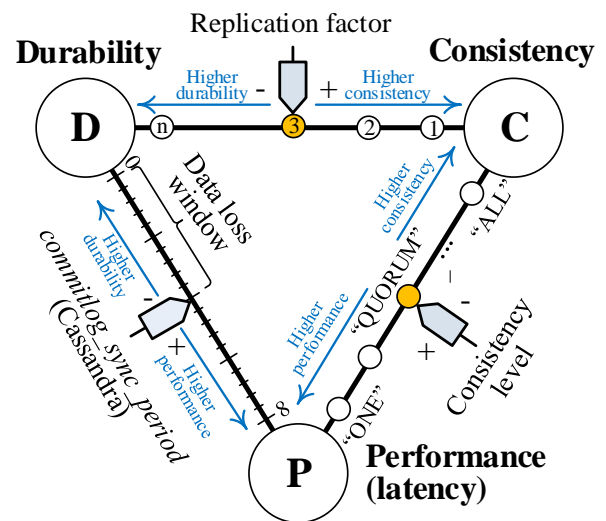


Fig. 3. Trade-offs between Durability, Performance and Consistency

D. Trade-offs Between Energy Consumption and CAP properties

The global information and communications technology industry, according to Gartner's investigation [14], is responsible for approximately 2 percent of global CO₂ emissions that is, for example, equivalent to the CO₂ emissions of the whole aviation sector of economy. There are several more facts highlighting the importance of IT contribution into the world's energy consumption and CO₂ emission.

Alex Wissner-Gross investigated [15] that a typical Google search request generates about 7g of CO₂ (Google contradicts that it is only about 0.2 g) that is half as much as boiling a kettle, while visiting a Web site takes on average about 20 milligrams of CO₂ per second.

The total power supply of the all computing and communication equipment in the world accounts for 160 GW per year that is about 8% of the total generated energy in the world [16]. It is also worth to note that 1W of application computing requires 27W of data center power and the aggregated energy loss can reach up to 97% [17].

Thus, enhancing power effectiveness of information and communication equipment, servers and data centers is one of the key issues in modern IT industry.

The interplay between Energy Consumption (EC), Fault-Tolerance (FT) and other CAP properties is shown in Fig. 2. Replication factor defines the main trade-off between fault tolerance and energy consumption which is proportional to the number of replicas.

Replication (i.e. redundancy) is introduced to the distributed computer systems with the two main purposes. Firstly, it is an effective approach to tolerate errors, failures and other abnormal situations, occurred in such systems. Secondly, replication increases performance of high loaded client-server systems by balancing users' requests between server's replicas.

At the same time, the high degree of redundancy (large replication factor) which assumes better fault-tolerance does not necessary ensure the high availability, which can be treated as a probability of a system to return response before the time-out (see Fig. 4).

The second important trade-off is the consistency level, defining a number of replicas invoked simultaneously during the execution of a particular read or write request. Higher consistency level increases system latency especially if replicas are distributed over the Internet (hosting all replicas in the same data center, in general case, reduces the deviation between their response times; though the probability of common-mode failure is increased).

Concurrent execution of redundant replicas additionally increases the overall power consumption.

Besides, in the global sustainability context the amount of energy spending on network transfer of the increased amount of data is also need to be accounted.

If one of the replicas returns its response beyond the specified application timeout the system enters a partition mode, causing timeout exception, or returns possibly incorrect (inconsistent) response to the client.

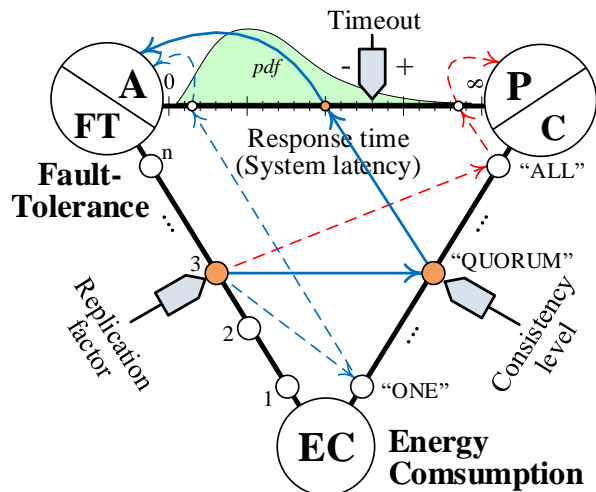


Fig. 4. Trade-offs between energy consumption and CAP properties

Thus the replication factor and consistency level contribute together to the overall energy consumption. We can consider the replication factor as the dominating factor in energy increase, while the consistency level is adding a variable component.

IV. CONCLUSION

When employing replication and other fault-tolerance techniques over the Internet and clouds, engineers have to deal with delays, their uncertainty, timeouts, adjudication of asynchronous replies from replicas, and other specific issues involved in global distributed systems. The overall aim of this work was to study fundamental trade-offs in distributed database systems and fault tolerant Internet computing and also between CAP properties, durability and energy consumption.

Distributed nature of modern computer application increases the failure probability. Replication helps to ensure system fault tolerance and to increase its performance by load balancing. At the same time, running several replicas proportionally increase energy consumption. Besides, replicated system cause the consistency issue. Necessity to provide better consistency requires concurrent invocation of several replicas that additionally increases the overall energy consumption. Consequently, stronger data consistency worsens system latency as well as increases energy consumption. This finding confirms one of the generally adopted qualitative implications of the CAP theorem [6].

In the paper we discussed key system settings that can be used to interplay CAP properties, latency, durability and energy consumption. They include:

- Replication factor (number of system replicas).
- Consistency level (in terms of consistency model used by the system).
- Time-out settings (e.g. how long a client should wait for read/write operations to complete; how long the system should wait for replicas responses before it is considered partitioned).
- Commit log synchronization time (e.g. how often the commit log buffer stored in memory is synchronized to disk).

One should note that these settings are tightly connected and should not be considered individually. For instance, a replication

factor higher than one always causes consistency issue. However, depending on the consistency settings higher replication factor can further increase system latency (if a system is configured to provide more strict consistency guarantees) or reduce it (if consistency is more relaxed).

Though existing researches help to define a series of useful qualitative implications of the CAP/PACELC theorems (e.g. ‘*better consistency worsens system availability and latency*’), developers have not provided yet with adequate quantitative models helping to predict system response time corresponding to the chosen consistency level and to precisely trade-off between them.

Estimation of the system worst-case execution time still remains a common practice for many applications (e.g. embedded computer systems, server fault-tolerance solutions, like STRATUS, etc.). However this approach is no longer a viable solution for the large-scale computer systems which replicas/components are globally distributed across the Internet.

In our previous works we demonstrated that unpredictable extreme delays exceeding the value of ten average response times could happen in such system quite often. Thus, system architects need novel analytical models providing a quantitative basis for the system response time prediction depending on the consistency level provided to (or requested by) clients.

REFERENCES

- [1] P. Lee and T. Anderson, *Fault Tolerance. Principles and Practice*, Wien - New-York: Springer-Verlag, 1990, p. 320.
- [2] A. Gorbenko, A. Romanovsky, O. Tarasyuk, V. Kharchenko and S. Mamutov, "Exploring Uncertainty of Delays as a Factor in End-to-End Cloud Response Time," in *9th European Dependable Computing Conference (EDCC'2012)*, Sibiu, Romania, 2012.
- [3] R. Potharaju and N. Jain, "When the Network Crumbles: An Empirical Study of Cloud Network Failures and their Impact on Services," in *4th ACM Symposium on Cloud Computing (SOCC'2013)*, Santa Clara, CA, 2013.
- [4] C. Scott, D. Choffnes, I. Cunha and e. al, "LIFEGUARD: practical repair of persistent route failures," in *ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, New York, USA, 2012.
- [5] A. Gorbenko, V. Kharchenko, O. Tarasyuk, Y. Chen and A. Romanovsky, "The threat of uncertainty in Service-Oriented Architecture," in *RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems, SERENE'08*, Newcastle, 2008.
- [6] E. Brewer, "Towards Robust Distributed Systems," in *19th Annual ACM Symposium on Principles of Distributed Computing*, Portland, USA, 2000.
- [7] S. Gilbert and N. Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services," *ACM SIGACT News*, vol. 33, no. 2, pp. 51-59, 2002.
- [8] A. Gorbenko, A. Romanovsky and O. Tarasyuk, "Fault tolerant internet computing: Benchmarking and modelling trade-offs between availability, latency and consistency," *Journal of Network and Computer Applications*, vol. 146, p. 102412, 2019.
- [9] D. Abadi, "Consistency Tradeoffs in Modern Distributed Database System Design," *IEEE Computer*, vol. 45, no. 2, pp. 37-42, 2012.
- [10] A. Tanenbaum and M. Van Steen, *Distributed systems: Principles and Paradigms*, Pearson Prentice Hall, 2006, p. 704.
- [11] J. Brutlag, "Speed Matters for Google Web Search," Google, Inc., 22 June 2009. [Online]. Available: http://services.google.com/fh/files/blogs/google_delayexp.pdf. [Accessed 01 07 2019].
- [12] A. Gorbenko and A. Romanovsky, "Time-outing Internet Services," *IEEE Security & Privacy*, vol. 11, no. 2, pp. 68-71, 2013.
- [13] R. Alagappan, A. Ganesan, Y. Patel, T. S. Pillai, A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau, "Correlated Crash Vulnerabilities," in *12th USENIX Symposium on Operating Systems Design and Implementation*, Savannah, USA, 2016.
- [14] Gartner, Inc. , "Gartner Estimates ICT Industry Accounts for 2 Percent of Global CO2 Emissions," 2007. [Online]. Available: <http://www.gartner.com/newsroom/id/503867>.
- [15] J. Leake and R. Woods, "Revealed: the environmental impact of Google searches," 11 January 2009. [Online]. Available: <https://www.thetimes.co.uk/article/revealed-the-environmental-impact-of-google-searches-xvcc72q8t2z>.
- [16] T. Jörg, "Efficiency starts with the power supply," *Journal of network solutions/LAN*, vol. 4, 2013.
- [17] D. Neilson, "IBM and Dynamic Infrastructure," IBM Systems Group, March 2009. [Online]. Available: www.nesc.ac.uk/talks/968/NESC_Neilson.ppt.