# Report from the Cloud and Big-Data Systems Track

Aaron Blankstein, Natacha Crooks, Xianzheng Dou, Ayush Dubey, Peng Huang, Kai Mast, Radhika Mittal, Chunzhi Su, Matthias Vallentin and Irene Zhang

## 1 Consistency vs. Performance in BigData Systems

A recurring discussion theme during the cloud and big data systems working group was data consistency. There has been much research related to characterizing and tweaking the level of consistency in data storage and processing systems. However, big data poses novel challenges which make questions of consistency especially tricky in contemporary systems. First, building strongly consistency storage systems is difficult due to the velocity of data generated by modern big data applications. For example, Facebook generates 500 TB of data per day at multiple datacenters around the world [1]. It is non-trivial to store all such data and guarantee that applications will see a consistent timeline of data accesses. Second, the data in such modern applications is varied, and the variety poses challenges in coming up with a uniform schema that supports strong consistency. Third, the amount of data, by definition, is huge, and traditional database techniques for consistently storing and processing data do not serve such a large volume of data with acceptable performance. There was consensus among all present that we need to solve the tension between strong consistency and high performance in big data systems.

The root cause of this tension in big data systems is that in order to support the large volume and velocity of data and not perform at a snail's pace, such systems have to inevitably be either multicore or distributed or both. Because the data can be manipulated by multiple threads, at multiple machines, and often in multiple datacenters, ensuring consistency is a huge challenge. The rich body of traditional database literature does not serve such data well. For example, executing a database transaction by performing two-phase locking across datacenters will lead to very high commit latency.

Of course, enabling fast and consistent systems is a classic problem for systems researchers. Recent work has attempted to weaken the consistency guarantees of the system in order to achieve higher performance. Most of these systems have been advised by the so-called CAP theorem [3]. Essentially, the CAP theorem explains that in an environment that permits unconstrained system crash failures and network partitions, it is impossible to build a strongly consistent system. The CAP theorem is true, but not particularly interesting because in practice we can make stronger assertions about modern networks and distributed systems. Unfortunately, it has been interpreted by the designers of many contemporary systems to mean that we necessarily need to abandon one out of three desirable system properties—consistency, availability, and partition-tolerance. Thus, many storage systems such as MongoDB have preemptively given up on consistency in order to achieve better performance. Other systems have posited progressive weakening of the consistency guarantee, leading to a whole host of consistency levels with different definitions [11]. It is very difficult for application programmers to understand a variety of consistency models and write bug-free code against such systems.

The working group on cloud and big data systems was of the opinion that we need to build systems that offer strong consistency from the ground up. By incorporating strong consistency as a basic system design goal, we will ensure that the users of such big data storage and processing systems can easily understand the system semantics. This will minimize the number of bugs due to overestimation of the system capabilities. For example, a database that provides a transactional programming interface is much easier for programmers to understand.

The key challenge in ensuring that such strongly consistent systems also provide high performance, in terms of latency, throughput, scalability, etc. A lot of recent systems research has been focusing on new techniques for enabling strong properties without weakening system guarantees. HyperDex [5] is a document store that provides a transactional interface while enabling better performance than MongoDB, an eventually consistent database. FaRM [4] leverages new advances in hardware, such as RDMA and non-volatile DRAM to give high throughput database transactions. TAPIR [13] and Callas [12] implement novel transaction processing protocols to provide improved performance compared to conventional database systems on large volume of data. Combined with many other recent research works, these novel techniques show that we can achieve our goal of strong consistency on big data.

While building strongly consistent data systems within a datacenter is challenging and has been the focus of a lot of recent research, the working group also accepted that we do not yet have a panacea and there are many open problems. Providing strongly consistent operations across datacenters still remains challenging. Intelligent transaction protocol design only takes us so far in geo-replicated applications, since the network latency poses a fundamental barrier to the overall system performance. Even within a datacenter, a single application can often be served by a number of different databases, and consistently coordinating updates and reads across the different databases is an open problem. For such applications, which access data across multiple database instances, potentially across many datacenters, we need to build better protocols in concert with novel techniques across the stack such as better network primitives and intelligent scheduling algorithms.

## 2 Towards better performance guarantees for distributed systems

Over the last decade there has been tremendous progress in the scalability and reliability of distributed systems. New techniques for distributed transactions have allowed datastores to scale across multiple servers [2], data centers, and even continents [9]. At the same time, improved replication techniques [10] have allowed for better reliability and fault-tolerant centralized services.

However, there is still a disconnect between the proven correctness of these techniques and their actual implementation. While projects like IronFleet [8] allow formal verification of simple distributed systems, there is still a long road ahead until such an approach can be applied to large-scale systems. One of the key reasons for this is that, while properties such as liveness, safety and eventual consistencies have been formally proven [7], limited work has been done on providing any form of formal guarantees about the *performance* of a system. It is also hard to provide any empirical guarantees on performance in large-scale distributed systems, which are difficult to debug and reason about.

Given how important performance is, both for user satisfaction and to the service providers for generating revenues, we believe that the lack of such theoretical or empirical performance guarantees is a gaping hole in the distributed systems literature. As a preliminary attempt towards filling this hole, we discussed two proposals: the first one deals with the theoretical aspects of the problem and the second one deals with it empirically. We elaborate both of these proposals below.

**Theoretical - Providing Formal Performance Guarantees:** An interesting approach would be to extend formal methods to include guarantees about performance. A feasible method to achieve this could be to estimate the number of iterations it takes for a system to converge, either using formal verification or mathematical modeling. However, as per the FLP [6] result, this would only be possible, if stronger guarantees are already given about the network performance. The work by Hawbitzel et al. [8] already built there liveness proofs on such bounds. Arguing with such worstcase bounds, however, would yield in very weak guarantees (e.g. high latency bounds). This means, we need to improve this practice or we will need to find another way to argue about time-bounds in a formally verified system.

**Empirical - Benchmarking a distributed system:** We believe that there is a need for easier benchmarking and testing of a distributed system. Automated testing is already prevalent on a single machine. A developer can just run ”make check” or similar tools to ensure their changes did not introduce any new bugs. However, for large-scale distributed systems, this is not as trivial. What if multiple concurrent failures occur? How does the system behave if a network link go down? Further, it is often nontrivial to estimate how a deployment affects performance. It might not be clear for an administrator how (and if) to shard data.

We believe that it might be possible to use existing techniques for virtualization and software-defined networks (SDNs), to deploy and test a distributed system. This would also make it easier for the research community to set up and repeat experiments. One might, for example, want a set of scripts that automatically creates multiple containers running instances of MongoDB. They can then introduce failure of several of the nodes at a specific point in time, to stress test the system. Without the use of automation, this would be hard to trigger at the exact intended time and even harder to reproduce. In this case using virtual machines will make it much easier to introduce such a failure deterministically. SDNs, on the other hand, can aid in simulating link failures. This can, thus, lead to a common set of benchmarks and evaluation suites, changing the way we conduct systems research significantly. One of the key challenges here would be to ensure that such a virtualized setup adds minimal overheads to the system performance and does not introduce any skew in the results.

## 3 Analyzing Geo-distributed Datasets

It is our belief that geo-distribution of cloud data will continue to be a major difficulty for big data systems. These difficulties can be reduced to scheduling problems: how do we schedule resources and place data to satisfy latency requirements and balance processing on less costly or loaded data centers? Furthermore, even once this problem is conceptualized as a set of constraints, how easy is it to solve these constraints? Do we need approximate solutions, and how well will they work?

Secondly, we argue that the problem of geodistribution is really a special case (multiple locations under single domain of control) of the more general problem of data management and analysis across multiple datacenters with multiple domains of control. When there are multiple domains of control, additional constraints must be enforced due to legal or organizational barriers which prevent data from moving. The same basic questions arise in this situation, but there are new ones, including concerns about data privacy and security. When is it safe to share analysis results or share processing? How can one domain trust the analysis of another?

## References

[1] Facebook processes more than 500 TB of data daily. `http://www.cnet.com/news/facebook-processes-more-than-500-tb-of-data-daily/`.

[2] M. K. Aguilera, A. Merchant, M. Shah, A. Veitch, and C. Karamanolis. Sinfonia: A New Paradigm for Building Scalable Distributed Systems. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 159–174. ACM, 2007.

[3] E. A. Brewer. Towards Robust Distributed Systems. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, 2004.

[4] A. Dragojevic, D. Narayanan, E. B. Nightingale, M. Renzelmann, A. Shamis, A. Badam, and M. Castro. No compromises: distributed transactions with consistency, availability, and performance. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 54–70, 2015.

[5] R. Escriva, B. Wong, and E. G. Sirer. HyperDex: A Distributed, Searchable Key-Value Store. In *Proceedings of the ACM SIGCOMM Conference*, pages 25–36, 2012.

[6] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[7] R. W. Floyd. Assigning meanings to programs. *Mathematical aspects of computer science*, 19(19-32):1, 1967.

[8] C. Hawblitzel, J. Howell, M. Kapritsos, J. R. Lorch, B. Parno, M. L. Roberts, S. Setty, and B. Zill. IronFleet: Proving Practical Distributed Systems Correct. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 1–17. ACM, 2015.

[9] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. 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*, SOSP '11, pages 401–416, New York, NY, USA, 2011. ACM.

[10] R. Van Renesse and F. B. Schneider. Chain Replication for Supporting High Throughput and Availability. In *OSDI*, volume 4, pages 91–104, 2004.

[11] W. Vogels. Eventually Consistent. *Communications of the ACM*, 52(1):40–44, 2009.

[12] C. Xie, C. Su, C. Littley, L. Alvisi, M. Kapritsos, and Y. Wang. High-Performance ACID via Modular Concurrency Control. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 279–294, 2015.

[13] I. Zhang, N. K. Sharma, A. Szekeres, A. Krishnamurthy, and D. R. K. Ports. Building Consistent Transactions with Inconsistent Replication. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 263–278, 2015.