Retrospective Analysis of Not so Terrific Systems - RANTS

Ashwin Raghav

Abstract

Conferences on Distributed Systems and Operating Systems get increasingly competitive by the year. In an attempt to provide scalable, reliable solutions in the Internet era, often one is forced to test solutions at volumes that are experienced only by a small fraction of the systems architects community. The Googles, Twitters and the Facebooks of our era have defined the way we think about Distributed Systems. The systems that back these organisations not only serve millions of users but have also been successful at academic conferences. Year after year, papers from Industry 'research labs' redefine problems, and at times even reinvent solutions.

In this essay, I wish to perform a retrospective analysis on a few papers from the industry that have been impactful enough to invent their own class of problems in the systems community. This article can be treated as a casual rant with gross generalisations or as an insight into how systems research has been influenced by academic contributions from Internet companies. In section I, we speak about the reasons that make Internet companies successful publishers in Distributed Systems conferences. In sections II, III and IV, we provide some examples of popular contributions from the industry that openly contradict common knowledge. In section V, ideas that may help academicians focus their efforts more effectively on distributed systems is discussed. In section VI, we conclude.

I. The Vicious Cycle

Large Scale Systems are run by two types of organisations: 1) Internet Companies 2) Super Computing Centres [1]. A survey of research papers from OSDI and SOSP indicated that Internet companies have published 4 times as often as Supercomputing Centres in the past 5 years.

Ashwin Raghav is a (struggling) graduate student at the University of Virginia, Charlottesville.

[1]gross generalisation

Internet companies have invented for themselves a new class of systems that deal at scales that are parred only by imagination. At scales that some of these systems are built, faults that are unlikely in smaller systems simply become more probable. Hence, it is no surprise that a large portion of academic contributions coming from the industry deals with fault tolerance and reliability. Also, production pressures are typically what drive the design decisions that go into building such large scale systems. Although these systems were not built keeping in mind the academic value that they contribute, they define the state of the art. They succeeding in defining their own problems and building their own solutions. In this section we examine some of the reasons why this has become largely possible.

A. Building Ecosystems

A large section of Distributed Systems prove their worth by way of empirical data. The value of providing theoretical proof in a heterogeneous environment is tremendously high. However, the sheer difficulty in modelling the different variables that affect a system often pushes researchers to resort to statistical analysis over theoretical guarantees.

Most well known techniques in statistical analysis are trustable only with a large volume of data. As a result, academic contributions to the field face the challenge of being able to test their contributions with a large volume of data. Internet companies that build their own economic ecosystems have easier access to such data than academicians whose goals are rarely of an economic nature.

As a simple example, Amazon Dynamo is made available as a service to Amazon Web Services Customers. Facebook's Notification API is extensively used by organisations like Zynga that are themselves fairly large. Such collaborative API provisions help Internet companies build a dataset that is rarely available to academicians. Not only is such widespread usage valuable but it provides tremendous insights into the pattern of problems that can be proactively solved for the future.

B. Mimicking organisations

The most important aspect that favours industry research in Distributed Systems is the influence they have on other organisations. Work from the academia rarely cuts across organisational boundaries or does so slowly at best. On the other hand, work from the industry cuts across organisations fairly quickly. Unlike the academia, the industry finds no shame in mimicking other organisations that have had similar problems. The positive impact that this has on Industry research is that organisations that copy solutions over are rarely aware that they are copying problems over as well. As a result, solutions to these problems that are later published by the parent companies inherently become solutions to important wide spread problems. A good example of this is the Apache Hadoop project. The project was initially an open source equivalent to the Google File System, Map Reduce and Big Table. But on account of the loose consistency measures that were introduced, other projects like Zookeeper had to be included later as traditional synchronization primitives that worked at the persistence levels were no longer applicable. Such is the effect of having a community using solutions that organisations build. It automatically makes future problems important.

C. Configuration Management of data centres

There has been a lot of traction that the ”Cloud” has received in recent times. The biggest selling point for Platform As a service has been the pro rated costs that companies incur in hardware usage. The elastic nature of the offering guarantees organisations that resources may be acquired or dropped at will. Although this seems incredibly attractive at the offset, there are several problems that remain unsolved.

Configuration Management of these resources is an incredibly hard problem. AWS for example has custom Amazon Machine Instances that can be instantiated on hardware. However, once deployed, it really becomes the prerogative of the acquirer to configure and manage resources. For distributed solutions this is a highly relevant problem.

Consider the following scenario. If you wish to acquire a large number of instances from a cloud provider to test your scalable, distributed software, there is no easy way to essentially install these solutions onto a large number of systems. Installations contain dependencies. They need to be updated with patches. They may need restarts. All these need to be incrementally available even after deployment. Some tools like Zookeeper, Chef and Puppet are solutions for such configuration state management. Such tools involve a steep learning curve and leave room for bugs in deployment. Internet companies that have large teams working on making deployment easier have the advantage of being able to deploy, update and patch software at will. This is a practical problem that remains largely relevant to building Distributed Systems in academia.

D. Economics of Scale

Many Internet companies spend time building solutions that need to be retrofit into a problem that they already face. As a first step to any scalability issues, Internet companies are always willing to throw more hardware at the problem if that suffices. The sheer economic flexibility that such environments have outranks the potential to acquire new resources in the academia. I have personally had a chance to work in a large Internet company before and have seen this happen many a time. By the time an efficient solution is in place for a problem, there is already a large amount of money invested in temporarily solving the problem.

E. Bleeding into Subsystems

Traditional Distributed Systems literature suggests that systems must be decoupled and autonomous in their behaviour. In reality, the multi tenancy of these systems results in systems that are inter dependent. And the effect of using a generalised solution can have a telling impact on the results. In such cases, organisations that have solved all active problems stand to benefit.

As a simple example, most systems that guarantee reliability will resort to disk level persistence of some sort. This typically bleeds the scope of a project into filesystems or databases. Like any other part of a system, databases and file systems can be tuned to accommodate specific workloads. Google as an organisation has benefited tremendously out of the Google File System and the Big Table subsystems. Thialfi for example is a notification system that guarantees reliability of notifications. It is widely used by Google Applications like Calendar, Google Plus etc. The data model of Thialfi is tightly coupled to the storage API that is most appropriate for Big Table. As evident from some of their own results, batching writes to Big Table is the process that contributes the most towards notification latency. As such, this paper has become the benchmark for providing scalable notification delivery and it is unlikely that we come across a paper that can better its performance in the near future.

II. Thialfi[1]

A. Paper Summary

Ensuring the freshness of client data is a fundamental problem for applications that rely on cloud infrastructure to store data and mediate sharing. Thialfi is a notification service developed at Google to simplify this task. Thialfi supports applications written in multiple programming languages and running on multiple platforms, e.g., browsers, phones, and desktops. Applications register their interest in a set of shared objects and receive notifications when those objects change. Thialfi servers run in multiple Google data centers for availability and replicate their state asynchronously. Thialfi's approach to recovery emphasizes simplicity. All server state is soft, and clients drive recovery and assist in replication. A principal goal is to provide a straightforward API and good semantics despite a variety of failures, including server crashes, communication failures, storage unavailability, and data center failures.

In the sections that follow, some important aspects that are sure to bring in a new set of problems for google to solve in the future are discussed.

B. Asynchronous Replication of State

Thialfi has multiple representations of client registration and notification state. This is to be differentiated from the general notion of replication that one finds on systems to improve performance and availability. This is done in order to improve io throughout. My claim is that a large portion of this was done to improve access via Big Table API that parameterizes row, column and version number. In order to maintain such duplicated state, the paper just claims to carry out asynchronous copying. There is no details on how any inconsistency is tolerated between the components that accepts registration and those that carry out notification.

C. Dependancy on Big Table

In order to guarantee that failed notifications can be recovered, notifications are persisted onto disk. This has been a problem is several pub sub systems. However, the fact that Google's Big

Table schema so cleanly fits into being able to store client-id, object-id, version as row, column and version greatly mitigates the latency involved in disk writes. Such wholesome solutions are hard to recreate.

D. Lack of Message Ordering

Thialfi claims that notifications have not really required ordering in their applications. Traditionally this has been one of the biggest problems solved by pub sub systems. As a simple example, a file system that publishes file open, read and write notifications needs that these are delivered before a delete notificaiton. Any change in this partial ordering can confuse the application semantics. Such a gross generalisation that is highly specific to a set of applications brings in scope to solve ordering problems in the future with the necessary eco system already in place.

E. Unsupported partial state loss

Thialfi claims that any loss of state in a single registration server requires restarting all servers within a datacenter. Although their client side semantics support a client driven recovery, it does not support incremental/partial recovery. This requires that their entire set of servers are restarted. In my opinion, this cannot be called a ”mechanism” to support server failures.

F. Flood of polling requests to client

When server state is lost or when data migrations are lost, all clients are forced to poll for the latest state of subscribed objects in order to simplify implementation. This is sure to cause a sudden spike in polling traffic at the concerned applications. This spike would be newly introduced by Thialfi thereby forcing applications to scale to a larger number of concurrent services than needed otherwise.

G. Unreliable writes

The solution that Thialfi has to combat Big Table unavailability in such a high throughout system is to retry the writes at pre determined fixed locations and run background jobs to bring consistency later. Such a solution fits better in an edition of ”Pragmatic Programmers” than on an academic paper. Once again, no details are provided on how the system tolerated inconsistencies in the intermittent period.

III. DYNAMO[3]

A. *Paper Summary*

Amazon's nature of business requires that some applications be provided with a service that offers 'always-write' semantics. The nature of shopping carts and ecommerce is closely tied to this philosophy. Such semantics have proven to be very hard to provide in the presence of reliability and scalability requirements. Also Amazon's Service Oriented Architecture requires that systems gurantee a Service Level Agreement to the 99th percentile. By sacrificing consistency, Amazon's Dynamo satisfies the required 'always-on' semantics by providing a Key-Value store that does not provide the rich semantics of traditional RDBMS systems.

B. *Single Hope DHT*

Several papers in Distributed System have laid emphasis on the importance of handling meta data about node locations. Traditionally it has been shown by projects like Chord[7] and Scribe[6] that Distributed Hash Tables are key to avoid scalability issues. On the other hand, Dynamo openly states that a single hope DHT without any establishment of hierarchical structure is required to keep up the Service Level Agreements. In order to combat this, Amazon recommends that each application deploy a separate instance of dynamo without sharing a single deployment across applications. This seems like an acceptable solution. The paper does admit that such methods of meta-data maintenance will not scale to over few hundred nodes.

C. *Arbitrary Version Truncation*

Dynamo uses data versioning to provide eventual consistency of data. In a system that contains N nodes, Dynamo can be tuned to guarantee atleast W nodes write a value and reads are consolidations from R nodes. When R+W is greater than N, a simple quorum protocol is sufficient to resolve inconsistencies. In other cases, applications are responsible for conflict resolution or the system adopts strategies like "last write wins". Vector Clocks are used as opposed to time stamps to track versions.

In cases where versions diverge, versions are not maintained infinitely until resolved, Instead, once a time span expires, the clock is truncated starting at the oldest entry. Such truncation can lead to severe problems when versions are reconciled. However, this was supposedly never a problem in practice. There is no information on how clients handling the data from this truncated version are supposed to react. Once this happens, no quorum can be used to resolve version inconsistencies.

D. Hinted Handoff Inconsistencies

Dynamo gives clients the guarantee of writing to atleast W clients before returning. When one of the W clients in the particular key' s preferred list is unavailable, Dynamo will simply write to any other available store marking the record as pending. This method is called a hinted handoff. It is unclear how reads are resolved in such cases where any node coordinating a read is expecting R nodes to return values.

E. Global Knowledge of Failed Nodes

As per Dynamo's semantics, any node that is part of the deployment can be a coordinator for key reads. When a read is carried out, the coordinator gathers values from R nodes and resolves their versions if possible or returns all versions to the application for conflict resolution. these R nodes picked are the the first R healthy nodes of a key's preference List. Such a strategy requires that a node be aware of the health of all other nodes of a system. Once again, this may be acceptable in practice. But an academic paper is sure to receive criticism for such short comings on account of having no 'production' data.

IV. Zookeeper[5]

A. Paper Summary

Zookeeper is a distributed process coordination service that was developed by Yahoo. Systems like the Amazon Queueue Service are useful for specific coordination paradigms like queueing. However a generalised API abstraction that can be used to build synchronization techniques was unavailable at that point. By providing a wait free coordination service that can be used as a general API by all applications, Zookeeper becomes a system that cuts across application semantics and solves problems at the coordination level. The API creates hierarchical node like virtual data structure that helps applications implement locks, barriers, rendezvous etc with the additional guarantees that replication provides along with the semantic changes that relaxed consistency measures introduce.

B. FIFO client order

One of the crucial properties that make it possible to implement Locks with Zookeeper is the fact that it provides ordering guarantees to a single client. For example when a client creates the root of two trees in a particular order, it is guaranteed that any client that has a 'watch' on the trees will receive the notifications for the root creations in the actual order of creation. There is no details on how this is done since several papers like Bauyeaux [8], Scribe [6] and Siena [2] have shown that ordering guarantees are an expensive proposition. Zookeeper on the other hand has not information about how such ordering is provided or what kind of time synchronization such ordering entails.

C. Lack of Partition Tolerance

When a lock is created using ephemeral nodes in Zookeeper, when a client dies for some reason, the ephemeral nature of the lock will destroy the lock automatically. This is detected using a session timeout when no heartbeats are received from the application. From an application developer's point of view, when an application is partitioned from a Zookeeper node, its session times out and the activity being performed after the lock has been acquired must be stopped. This seems like a complicated problem since in several cases one may have to design complex rollback mechanisms when a Zookeeper coordinator declares a client dead as a result of network partition.

D. Atomic Broadcasts

In order to increase the availability of the system, all nodes that contain the hierarchical data structure representation that Zookeeper proposes are replicated. Reads are mostly completed from the node where the request arrives. Writes on the other hand entail conversion to an idempotent form. It is unclear how any function can be made idempotent. After this, an atomic broadcast to all replicas brings in a universal ordering between them. Once again this is a very expensive proposition in terms of performance penalties and it cannot tolerate network partitions.

E. Invalidating Caches

In order to improve the performance of reads, all information that is read is cached on the client. As a result when a node that is being watched is updated, the write must wait until all clients who are currently watching the node are informed about the mutation and the client side cache is invalidated. Such a straight forward cache invalidation scheme will neither scale in terms of write throughout nor in terms of client cardinality.

V. Retrospective

Although a large part of distributed systems conferences continue to be academic in nature, papers from the industry become proverbially more popular. In this section I briefly would like to provide some take aways on what can be done to change the impact academic distributed systems papers have.

Build and put systems out there: In order to compete with systems of the larger companies, academic systems groups must build end to end systems and provide it for public use. Projects like Scribe have been tested on simulators. On the other hand Coral CDN [4] and Chord[7] have been deployed onto real world distributed systems and have received significant credit for doing so.

Investing in academic Data Centres: A large portion of the funding that goes into academic systems research is allocated to building super computing centres. In the future, apportioning a part of this to building data centres that academicians can use can be of help .Although the cloud provides a virtualized environment, performance guarantees are non isolated. Academicians must focus on providing isolated performance results by using shared data centres as test beds. This itself proves to be an interesting area that remains largely unexplored.

Configuration Management: As previously mentioned, clean configuration management tools have a steep learning curve. In order to use the cloud to test large systems, better tools for this purpose with cleaner scaling mechanisms need to be in place.

Entering the Mobile Space: In the past three years two out of the three best papers of sosp were academic papers. Both these papers belonged to the mobile space. With the large bed of open source mobile operating systems and tools available, systems teams from the academia must move on to focussing on thin mobile clients. Typically such endeavours do not involve significant infrastructure costs.

Conference tracks for mid sized systems: Systems that are deployed to the scale of Internet giants form only a very small percentage of actual systems in production. Several problems that smaller systems face can be solved more elegantly and efficiently by compromising on the reliability guarantees that larger systems need to provide. In most of these cases, such large scale solutions are overkills. Conferences that focus on performance of mid sized systems can be adopted in several situations. This will truly add value to the conferences themselves.

VI. Conclusion

The contributions from the industry in Distributed Systems conferences have a large impact on the progress of research in the area. The academia in many ways is thankful to some of these firms for open sourcing solutions. As an overarching thought, academicians must however learn to design and test systems that deviate from academic ’common-knowledge’ and focus on systems that work. To test such systems we need to either have shared academic data centres or learn to appreciate the value of having mid sized distributed systems solutions. This paper is meant to be a starting point by being a retrospection on distributed systems contributions from the industry. We provide some premature but useful take aways on how academicians can move to a paradigm that fits into the current scheme of matters. It is my sincere hope that we continue retrospecting on the state of Distributed Systems and continuously make an effort to understand how the state of the art can be improved by an academician. It would truly enhance the sheer pleasure of building systems and seeing them work.

References

[1] A. Adya, G. Cooper, D. Myers, and M. Piatek. Thialfi: a client notification service for internet-scale applications. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP ’11, pages 129–142, New York, NY, USA, 2011. ACM.

[2] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, 19(3):332–383, Aug. 2001.

[3] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon’s highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP ’07, pages 205–220, New York, NY, USA, 2007. ACM.

[4] M. J. Freedman. Experiences with coralcdn: A five-year operational view. In *In Proc NSDI*, 2010.

[5] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. Zookeeper: wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIXATC'10, pages 11–11, Berkeley, CA, USA, 2010. USENIX Association.

[6] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructure. In *Proceedings of the Third International COST264 Workshop on Networked Group Communication*, NGC '01, pages 30–43, London, UK, UK, 2001. Springer-Verlag.

[7] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, Feb. 2003.

[8] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '01, pages 11–20, New York, NY, USA, 2001. ACM.