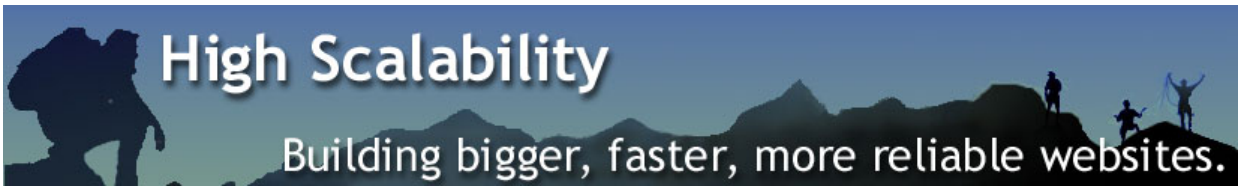


→

RECENT POSTS

- Engineering dependability and fault tolerance in a distributed system
- Sponsored Post: 3T, Bridgecrew, Toptal, IP2Location, Ipdata, StackHawk, InterviewCamp.io, Educative, Triplebyte, Stream, Fauna
- Benchmark (YCSB) numbers for Redis, MongoDB, Couchbase2, Yugabyte and BangDB
- Sponsored Post: 3T, Bridgecrew, Toptal, IP2Location, Ipdata, StackHawk, InterviewCamp.io, Educative, Triplebyte,

- Login
- Register

- High Scalability RSS
- High Scalability Comments RSS

Stream, Fauna

- Stuff The Internet Says On Scalability For February 1st, 2021
- Sponsored Post: 3T, Bridgecrew, Toptal, IP2Location, Ipdata, StackHawk, InterviewCamp.io, Educative, Triplebyte, Stream, Fauna
- Sponsored Post: Bridgecrew, Toptal, IP2Location, Ipdata, StackHawk, InterviewCamp.io, Educative, Triplebyte, Stream, Fauna
- The Read Aloud Cloud: An Interview With Forrest Brazeal On His New Book
- Sponsored Post: Toptal, IP2Location, Ipdata,

StackHawk,
InterviewCamp.io,
Educative,
Triplebyte,
Stream,
Fauna

- Stuff The Internet Says On Scalability For December 19th, 2020

Friday
Feb192021

Engineering Dependability And Fault Tolerance In A Distributed System

FRIDAY, FEBRUARY 19, 2021 AT 9:09AM

This is a guest post by Paddy Byers, Co-founder and CTO at Ably, a realtime data delivery platform. You can view the original article on Ably's blog.

Users need to know that they can depend on the service that is provided to them. In practice, because from time to time individual elements will inevitably fail, this means you have to be able to continue in spite of those failures.

In this article, we discuss the concepts of dependability and fault tolerance in detail and explain how the Ably platform is designed with fault tolerant approaches to uphold its dependability guarantees.

As a basis for that discussion, first some definitions:

> ***Dependability***
> *The degree to which a product or service can be relied upon.* ***Availability*** *and* ***Reliability*** *are forms of dependability.*
>
> ***Availability***
> *The degree to which a product or service is* ***available*** *for use when required. This often boils down to provisioning sufficient*

> *redundancy of resources with* statistically independent failures.
>
> ***Reliability***
> *The degree to which the product or service* ***conforms to its specification*** *when in use. This means a system that is not merely available but is also engineered with extensive redundant measures to* ***continue to work as its users expect****.*
>
> ***Fault tolerance***
> *The ability of a system to continue to be* ***dependable*** *(both available and reliable) in the presence of certain component or subsystem failures.*

Fault tolerant systems tolerate faults: they're designed to mitigate the impact of adverse circumstances and ensure the system remains dependable to the end-user. Fault-tolerance techniques can be used to improve both availability and reliability.

Availability can be loosely thought of as the assurance of uptime; reliability can be thought of as the quality of that uptime — that is, assurance that functionality and user experience and preserved as effectively as possible in spite of adversity.

If the service isn't available to be used at the time it is needed, that's a shortfall in availability. If the service is available, but deviates from its expected behavior when you use it, that's a shortfall in reliability. Fault tolerant design approaches address these shortfalls to provide continuity both to business and to the user experience.

Availability, Reliability, And State

In most cases, the primary basis for fault tolerant design is **redundancy**: having more than the minimum number of components or capacity required to deliver service. The key questions relate to what form that redundancy takes and how it is managed.

In the physical world there is classically a distinction between

- **availability** situations, where it is acceptable to stop a service and then resume it, such as stopping to change a car tire; and
- **reliability** situations, where continuity of service is essential, with redundant elements continuously in-service, such as with airplane engines.

The nature of the continuity requirement impacts the way that redundant capacity needs to be provided.

In the context of distributed systems, at Ably we think of an analogous distinction between components that are **stateless** and those that are **stateful**, respectively.

Stateless components fulfil their function without any dependency on long-lived state. Each invocation of service can be performed independently of any previous invocation. Fault tolerant design for these components is comparatively straightforward: have sufficient resources **available** so that any individual invocation can be handled even if some of the resources have failed.

Stateful components have an intrinsic dependency on state to provide service. Having state implicitly links an invocation of the service to past and future invocations. The essence of fault tolerance for these components is, as with an airplane's engines, about being able to provide continuity of operation — specifically, continuity of the state that the service depends on. This ensures **reliability**.

In the remainder of this article we will give examples of each of these situations and explain the engineering challenges encountered in achieving fault tolerance in practice.

Treating Failure As A Matter Of Course

Fault tolerant designs treat failures as routine. In large-scale systems, the assumption has to be that component failures will happen sooner or later. Any individual failure must be assumed as imminent and, collectively, component failures must be expected to be occurring continuously.

By contrast with the physical world, failures in digital systems are typically non-binary. The classical measures of component reliability (e.g. Mean Time Between Failures, or MTBF) do not apply; services degrade along a gradient of failure. Byzantine faults are a classic example.

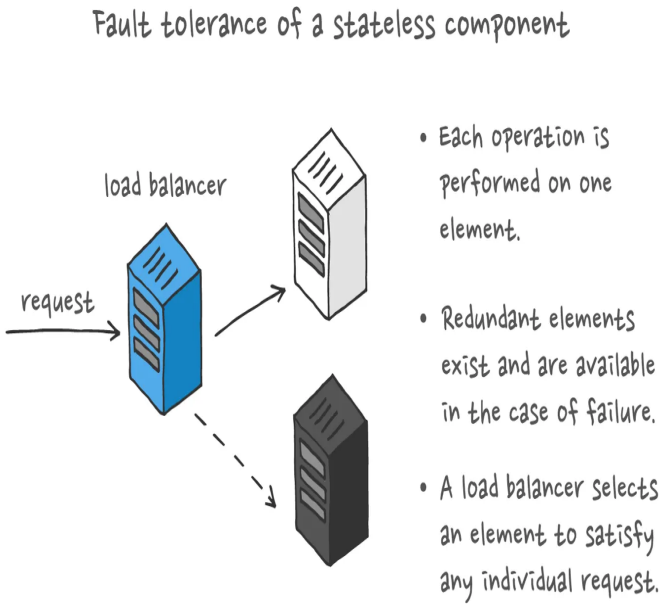
For example, a system component might work intermittently, or produce misleading output. Or you might be dependent on external partners who don’t notify you of a failure until it becomes serious on their end, making your work more difficult.

Being tolerant to non-binary failures requires a lot of thought, engineering and, sometimes, human intervention. Each potential failure must be identified and classified, and must then be capable of being remediated rapidly, or avoided through extensive testing and robust design decisions. The core challenge of designing fault-tolerant systems is understanding the nature of failures and how they can be detected and remediated, especially when partial or intermittent, to continue to provide the service as effectively as possible for users.

Stateless Services

Service layers that are **stateless** do not have a primary requirement for continuity of service for any individual component. The availability of resources directly translates into availability of the layer as a whole. Having access to extra resources whose failures are statistically independent is key to keeping the system going. Wherever possible, layers are designed to be stateless as a key enabler not only of availability, but also of scalability.

For stateless objects, it suffices to have multiple and independently available components to continue to provide service. Without state, durability of any single component is not a concern.



However, simply having extra resources is not enough: you also have to use them effectively. You have to have a way of detecting resource availability, and to load-balance among redundant resources.

As such, the questions that you have to answer are:

- How do you survive different kinds of failure?
- What level of redundancy is possible?
- What is the resource and performance cost of maintaining those levels of redundancy?
- What is the operational cost of managing those levels of redundancy?

The consequent trade-offs are among the following:

- Customer requirements for achieving high availability
- Business operational cost

- Real world engineering practicality of actually making it possible

Redundant components, and in turn their dependencies, need to be engineered, configured and operated in a way that ensures that any failures are statistically independent. The simple math is: statistically independent failures render your chances of a catastrophic failure exponentially lower as you increase the level of redundancy. If the failures are set up to occur in statistical silos, then there is no cumulative effect, and you decrease the likelihood of complete failure by a whole order of magnitude with each additional redundant resource.

At Ably, to increase statistical independence of failures, we place/distribute capacity in multiple availability zones and in multiple regions. Offering service from multiple availability zones within the same region is relatively simple: AWS enables this with very little effort. Availability zones generally have a good track record of failing independently, so this step by itself enables the existence of sufficient redundancy to support very high levels of availability.

However, this isn't the whole story. It isn't sufficient to rely on any specific region for multiple reasons – sometimes multiple availability zones (AZs) do fail at the same time; sometimes there might be local connectivity issues making the region unreachable; and sometimes there might simply be capacity limitations in a region that prevent all services from being supportable there. As a result, we also promote service availability by providing service in multiple regions. This is the ultimate way of ensuring statistical independence of failures.

Establishing redundancy that spans multiple regions is not as straightforward as supporting multiple AZs. For example, it doesn't make sense simply to have a load balancer distributing requests among regions; the load balancer itself exists in some region and could become unavailable.

Instead, we use a combination of measures to ensure that client requests can at all times be routed to a region that is believed to be healthy and have service available. This routing will preferably be to the nearest region, but routing to non-local regions must be possible when the nearest region is unable to provide service.

Stateful Services

At Ably, reliability means business continuity of *stateful* services, and it is a substantially more complicated problem to solve than just availability.

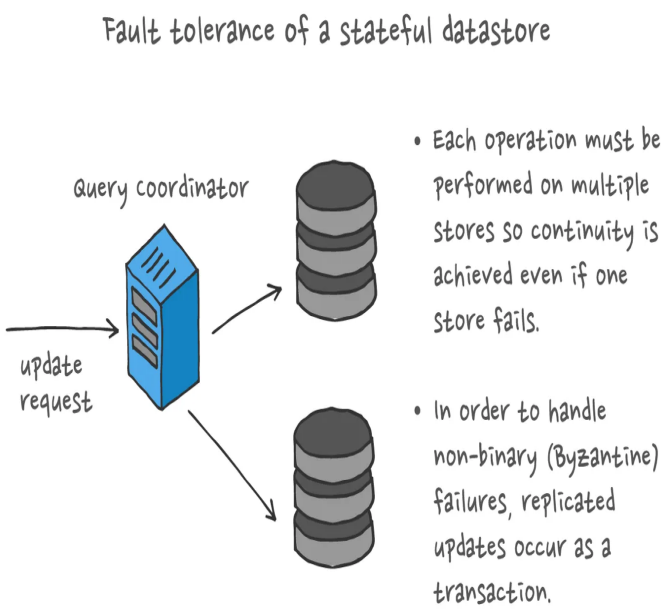
Stateful services have an intrinsic dependency on state that survives each individual invocation of service. Continuity of that state translates into correctness of the

service provided by the layer as a whole. That requirement for continuity means that fault tolerance for these services is achieved by thinking of them in classic reliability terms. Redundancy needs to be continuously in use, in order for the state not to become lost in the case of failure. Fault detection and remediation needs to address the possible Byzantine failure modes via consensus formation mechanisms.

The most simplistic analogy is with airplane safety. An airplane crash is catastrophic because you – and your state – are on a *specific* airplane; it is *that* airplane which must provide continuous service. If it fails to do so, state is lost and you are afforded no opportunity to continue by migrating to a different airplane.

With anything that relies on state, when an alternate resource is selected, the requirement is to be able to carry on with the new resource where the previous resource left off. State is thus a requirement, and in those cases availability alone is insufficient.

At Ably, we provision enough reserve capacity for stateless resources to support all of our customers' availability requirements. However, for stateful resources, not only do we need redundant resources, but also explicit mechanisms to make use of that redundancy, in order to support our service guarantees of *functional continuity*.



For example, if processing for a particular channel is taking place on a particular instance within the cluster, and that instance fails (forcing that channel role to move), mechanisms must be in place to ensure that things can continue.

This operates at several levels. At one level, a mechanism must exist to ensure that that channel's processing is reassigned to a different, healthy, resource. At another level, there needs to be assurance that the reassigned resource continues at exactly the point that processing was halted in the previous location. Further, each of these mechanisms is itself implemented and operated with a level of redundancy to meet the overall assurance requirements for the service.

The effectiveness of these continuity mechanisms directly translates into the behavior, and assurance of that behavior, at the service provision boundary. Taking one specific issue in the scenario above: for any given message, you need to know with certainty whether or not processing for that message has been completed.

When a client submits a message to Ably for publication, and the service accepts the message for publication, it acknowledges that attempt as successful or unsuccessful. At this point, the principal *availability* question is:

> *What fraction of the time does the service accept (and then process) the message, versus rejecting it?*

The minimum aim in this case is four, five, or even six 9s.

If you attempt to publish and we let you know we weren't able to do that, then that's merely an **availability shortcoming**. It's not great, but you end up with an awareness of the situation.

However, if we respond with success — "yes, we've received your message" — but then we fail to do all the onward processing of it, then that's a different kind of failure. That's a failure to uphold our functional service guarantee. That's a **reliability shortcoming**, and it's a far more complicated problem to solve in a distributed system — there is significant engineering effort and complexity devoted to meeting this requirement.

Architectural Approaches To Achieve Reliability

The following are two illustrations of the architectural approaches we adopt at Ably to make optimum use of redundancy within our message processing core.

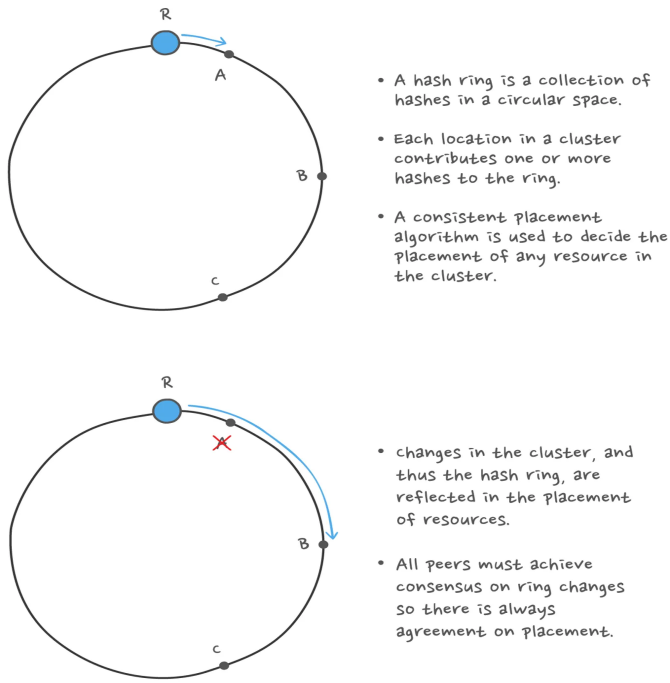
Stateful Role Placement

In general, horizontal scalability is achieved by distributing work across a scalable cluster of processing resources. As far as entities that perform **stateless** processing are concerned, they can be distributed across available resources with **few constraints** on their placement: the location of any given operation can be decided based on load balancing, proximity, or other optimization considerations.

Meanwhile, in the case of **stateful** operations, the placement of processing roles must take their stateful nature into account such that, for example, all concerned entities can agree on the specific location of any given role.

A specific example is channel message processing: whenever a channel is active it gets assigned a resource that processes it. Being a stateful process, it is possible to achieve greater performance: we know more about the message at the time of processing, so we don't have to look it up — we can just process it and send it on.

In order to distribute all the channels across all the available resources as uniformly as possible, we use consistent hashing, with the underlying cluster discovery service providing consensus on both node health and hashring membership.



Achieving consensus via consistent placement algorithm in a hash ring

The placement mechanism is required not only to determine the initial placement of a role, but also to relocate the role whenever there is an event, such as node failure, that causes the role to move. Therefore, this dynamic placement mechanism is a core functionality in support of service continuity and reliability.

Detect, Hash, Resume

The first step in mitigating failure is detecting it. As discussed above, this is classically difficult because of the need to achieve near-simultaneous consensus between distributed entities. Once detected, the updated hashring state implies the new location of that resource, and from that point onward the channel and the state of

the failed resource must resume in the new location with continuity.

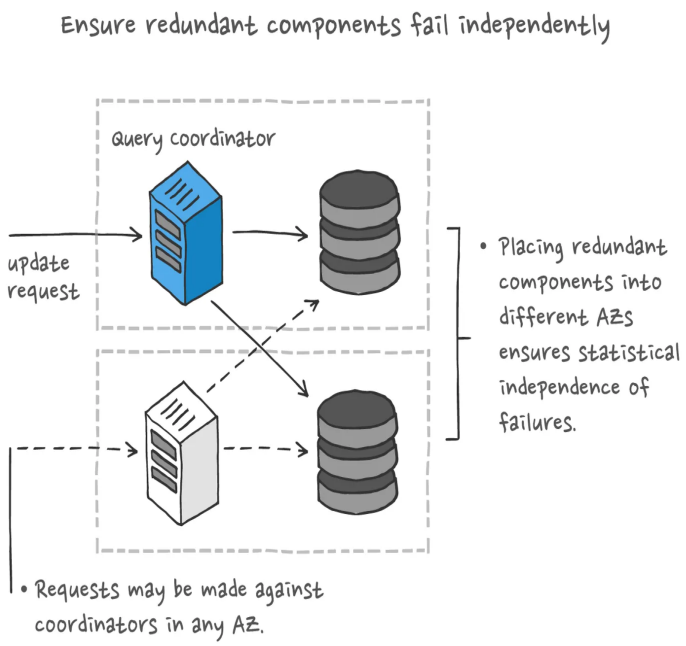
Even though the role failed and there was resulting loss of state, there needs to be sufficient state persisted (and with sufficient redundancy) that resumption of the role is possible with continuity. Resumption with continuity is what enables the service to be reliable in the presence of this kind of failure; the state of each in-flight message is preserved across the role relocation. If we were unable to do that, and simply re-established the role without continuity of state, we could ensure service availability — but not *reliability*.

Channel Persistence Layer

When a message is published, we perform some processing, decide on success or failure, then respond to the call. The reliability guarantee means having certainty that once a message is acknowledged, all onward transmission implied by that will in fact take place. This in turn means that we can only acknowledge a message once we know for a fact that it is durably persisted, with sufficient redundancy that it cannot subsequently be lost.

First, we record the receipt of the message in **at least two different availability zones** (AZs). Then we have the same multiple-AZ redundancy requirement for the onward processing itself. This is the core of the persistence layer at Ably: we write a message to multiple locations, and we also make sure the process of writing it is transactional. You come away knowing the writing of the message was either *not successful* or *unequivocally successful*. With assurance of that, subsequent processing can be guaranteed to occur eventually, even if there are failures in the roles responsible for that processing.

Ensuring that messages are persisted in multiple AZs enables us to assume that failures in those zones are independent, so a single event or cause cannot lead to loss of data. Arranging for this placement requires AZ-aware orchestration, and ensuring that writes to multiple locations are actually transactional requires distributed consensus in the message persistence layer.



Structuring things this way allows us to start to quantify, probabilistically, the level of assurance. A service failure can only arise if there is a compound failure — that is, a failure in one AZ and, before that failure has been remediated, a failure in a second AZ.

In our mathematical model, when one node fails, we know how long it takes to detect and achieve consensus on the failure, and how long it takes subsequently to relocate the role. Knowing this, together with the failure rate of each AZ, it is possible to model the probability of an occurrence of a compound failure that results in loss of continuity of state. This is the basis on which we are able to provide our guarantee of eight 9s of reliability.

Implementation Considerations

Even once you have a theoretical approach to achieving a particular aspect of fault tolerance, there are numerous practical and systems engineering aspects to consider in the wider context of the system as a whole. We give some examples below. To explore this topic further, read and/or watch our deep-dive on the topic, Hidden scaling issues of distributed systems — system design in the real world.

Consensus Formation In Globally-Distributed Systems

The mechanisms described above — such as the role placement algorithm — can only be effective when all of the participating entities are in agreement on the topology of the cluster together with the status and health of each node.

This is a classical consensus formation problem, where the members of a cluster, who might themselves be

subject to failure, must agree on the status of one of their members. Consensus formation protocols such as Raft/Paxos are widely understood and have strong theoretical guarantees, but also have practical limitations in terms of scalability and bandwidth. In particular, they are not effective in networks spanning multiple regions because their efficiency breaks down if the latency becomes too high when communicating among peers.

Instead, our peers use the Gossip protocol, which is eventually-consistent, fault-tolerant, and can work across regions. Gossip is used among regions to share topology information as well as to construct a network map, and this broad cluster state consensus is then used as the basis for sharing various details cluster-wide.

Health Is Not Binary

The classical theory that led to the development of Paxos and Raft originated from the recognition that failed or unhealthy entities oftentimes do not simply crash or stop responding. Instead, failing elements can exhibit partly-working behavior such as delayed responses, higher error rates or, in principle, arbitrary confusing or misleading behavior (see Byzantine fault). This issue is pervasive and even extends to the way a client interacts with the service.

When a client attempts connection to an endpoint in a given region, it is possible that the region is not available. If it's entirely unavailable, then that's a simple failure, and we would be able to detect that and redirect the client to greener pastures within the infrastructure. But what happens in practice is that regions can often be in a partially degraded state where they're working *some* of the time. This means that the client itself has the problem of handling the failure — knowing where to redirect to obtain service, and knowing when to retry getting service from the default endpoint.

This is another example of the general fault tolerance problem: with many moving pieces, every single thing introduces additional complexity. If this part breaks, or that thing changes, how do you establish consensus on the *existence* of the change, the *nature* of the change, and the consequent *plan of action*?

Resource Availability Issues

At a very simple level, it is possible to provide redundant capacity only if the resources required are available. Occasionally there are situations where resources are simply unavailable at the moment they are demanded in a region, and it is necessary to offload demand to another region.

However, the resource availability problem also surfaces in a more challenging way when you realize that fault

tolerance mechanisms themselves require resources to operate.

For example, there is a mechanism to manage the relocation of roles when the topology changes. This functionality itself requires resources in order to operate, such as CPU and memory on the affected instances.

But what if your disruption has come about precisely because your CPU or memory ran out? Now you're attempting to handle those failures, but to do so you need… *CPU and memory*. This means that there are multiple dimensions in which you need to ensure that there exists a resource capacity margin, so that fault tolerance measures can be enacted at the time they are needed.

Resource Scalability Issues

Further to the point above, it's not just about availability of resources, but also about the rate at which demand for them scales. In the steady, healthy state you might have *N* channels, *N* connections, and *N* messages with *N* capacity to deal with it all. Now imagine there is some failure and ensuing disruption for that cluster of *N* instances. If the amount of work required to compensate for the disruption is of size N^2, then maintaining the capacity margin becomes unsustainable, and the only available remediation is to fail over to an undisrupted region or cluster.

Simplistic fault tolerance mechanisms can exhibit this kind of $O(N^2)$ behavior or worse, so approaches need to be analyzed with this in mind. It's another reminder that while things can fail just because they're broken in some way, it's also possible that they can go wrong because they have an unforeseen scale or complexity or some other unsustainable resource implication.

Conclusion

Fault tolerance is an approach to building systems able to withstand and mitigate adverse events and operating conditions in order to dependably continue delivering the level of service expected by the users of the system.

Dependability engineering in the physical world classically makes the distinction between **availability** and **reliability** and there are well-understood formulas for how to achieve each of these through fault tolerance and redundancy. At Ably we make the analogous distinction in the systems world between components that are **stateless** and those that are **stateful**.

Fault tolerance for *stateless* components is achieved in the same way as for availability in the physical world: through provision of redundant elements whose failure is statistically independent. Fault tolerance for *stateful* components can be compared to the physical reliability

problem: it is the assurance of service without interruption. *Continuity of state* is essential in order that failures can be tolerated whilst preserving *correctness and continuity of the provided service*.

To be fault tolerant, a system must treat failures not as exceptional events, but as something expectedly routine. Unlike theoretical models of success and failure, threats to a system's health in the real world are not binary and require complex theoretical and practical approaches to mitigate.

The Ably service is engineered with multiple layers that each make use of a wide range of fault tolerance mechanisms. In particular, we have confronted the hard engineering problems that arise which include stateful role placement, detection, hashing, and graceful resumption of service, among others. We've also articulated, and provide assurance of, the service guarantee at the channel persistence layer, such as assured onward processing once we acknowledge that we've received a message.

Beyond theoretical approaches, designing fault tolerant systems involves numerous real-world systems engineering challenges. This includes infrastructure availability and scalability issues, as well as dealing with consensus formation and orchestration of ever-fluctuating topologies of all clusters and nodes in the globally-distributed system, with unpredictable/hard-to-detect health status of any given entity on the network.

The Ably platform was designed from the ground up with these principles in mind, with the goal of delivering the best-in-class enterprise solution. This is why we can confidently provide our service-level guarantees of both availability and reliability – and thus dependability and fault tolerance.

Get in touch *to learn more about Ably and how we can help you deliver seamless realtime experiences to your customers.*

Todd Hoff | Post a Comment | Permalink | Share Article Print Article Email Article

in Fault-tolerance

Wednesday
Feb172021

Sponsored Post: 3T, Bridgecrew, Toptal, IP2Location, Ipdata, StackHawk,

InterviewCamp.Io, Educative, Triplebyte, Stream, Fauna

WEDNESDAY, FEBRUARY 17, 2021 AT 10:08AM

Who's Hiring?

- Looking to rapidly **hire Top Software Developers**? Get Started with Toptal. **Toptal** will match you with top-quality, pre-screened freelance software developers that meet your project requirements. All in under 48 hours. Get started right away with a no risk trial.

- InterviewCamp.io has hours of system design content. They also do live system design discussions every week. They break down interview prep into fundamental building blocks. Try out their platform.

- **Triplebyte** lets exceptional software engineers skip screening steps at hundreds of top tech companies like **Apple, Dropbox, Mixpanel, and Instacart**. Make your job search *O*(1), not *O*(*n*). Apply here.

- **Need excellent people? Advertise your job here!**

Cool Products And Services

- Discover the MongoDB data masking tool in **Studio 3T Enterprise**. Enable data compliance and bolster security with powerful field-level data obfuscation. Try for free!

- **Bridgecrew** is the **cloud security platform for developers**. By leveraging automation and delivering security-as-code, Bridgecrew empowers teams to find, fix, and prevent misconfigurations in deployed cloud resources and in infrastructure as code. Get started for free!

- **IP2Location** is **IP address geolocation service provider** since 2002. The geolocation database or API detects location, proxy and other >20 parameters. The technology has been cited in more than 700 research papers and trusted by many Fortune 500 companies. Try it today!

- **ipdata** is a **reliable IP Address Geolocation API** that allows you to lookup the approximate location of any IP Address, detect proxies and identify a company from an IP Address. Trusted by 10,000+ developers. Try it now!

- Developers care about shipping secure applications. Application security products and processes, however, have not kept up with advances in software development. There are a new breed of tools hitting the market that enable developers to take the lead on AppSec. Learn how engineering teams are using products like **StackHawk** and **Snyk** to add security bug testing to their CI pipelines.

- Learn the stuff they don't teach you in the AWS docs. Filter out the distracting hype, and focus on the parts of AWS that you'd be foolish not to use. Learn the Good Parts of AWS. Created by former senior-level AWS engineers of 15 years.

- Stateful JavaScript Apps. Effortlessly add state to your Javascript apps with **FaunaDB**. Generous free tier. Try now!

- Learn to balance architecture trade-offs and design scalable enterprise-level software. Check out **Educative.io**'s bestselling new 4-course learning track: Scalability and System Design for Developers. Join more than 300,000 other learners.

- Build, scale and personalize your news feeds and activity streams with getstream.io. Try the API now in this 5 minute interactive tutorial. **Stream** is free up to 3 million feed updates so it's easy to get started. Client libraries are available for Node, Ruby, Python, PHP, Go, Java and .NET. Stream is currently also hiring Devops and Python/Go developers in Amsterdam. More than 400 companies rely on Stream for their production feed infrastructure, this includes apps with 30 million users. With your help we'd like to ad a few zeros to that number. Check out the job opening on AngelList.

- **Advertise your product or service here!**

If you are interested in a sponsored post for an event, job, or product, please** contact **us for more information.

Make Your Job Search *o*(1) — Not *o*(*n*)

Triplebyte is unique because they're a team of engineers running their own centralized technical assessment. Companies like **Apple, Dropbox, Mixpanel, and Instacart** now let Triplebyte-recommended engineers skip their own screening steps.

We found that **High Scalability readers** are about 80% more likely to be in the top bracket of engineering skill.

Take Triplebyte's multiple-choice quiz (system design and coding questions) to see if they can help you scale your career faster.

If you are interested in a sponsored post for an event, job, or product, please contact us for more information.

Todd Hoff | Post a Comment | Permalink | Share Article Print Article Email Article

in sponsored post

Wednesday
Feb172021

Benchmark (YCSB) Numbers For Redis, MongoDB, Couchbase2, Yugabyte And BangDB

WEDNESDAY, FEBRUARY 17, 2021 AT 9:59AM

This is guest post by Sachin Sinha who is passionate about data, analytics and machine learning at scale. Author & founder of BangDB.

This article is to simply report the YCSB bench test results in detail for five NoSQL databases namely Redis, MongoDB, Couchbase, Yugabyte and BangDB and compare the result side by side. I have used latest versions for each NoSQL DB and have followed the recommendations to run all the databases in optimized conditions. I have also used the default six test scenarios as defined by the YCSB framework. I have restricted it to 10M records for each test. However, user can run the bench for as many numbers as they practically find suitable.

About YCSB

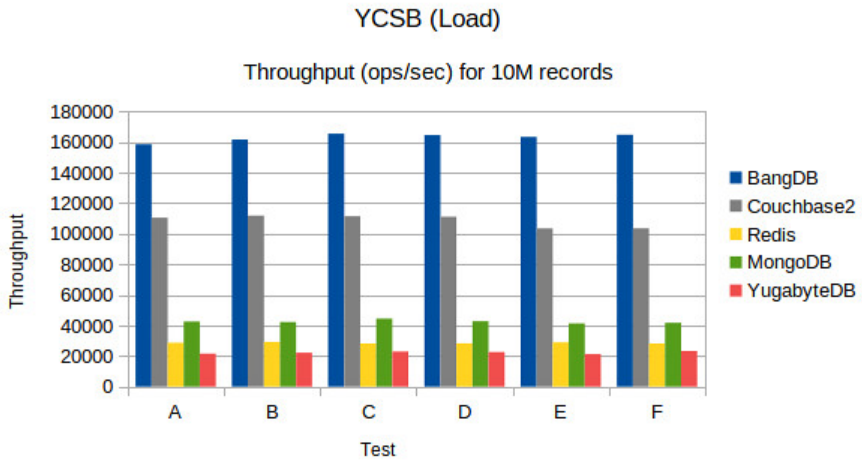
Following configurations were used for the evaluation purpose.

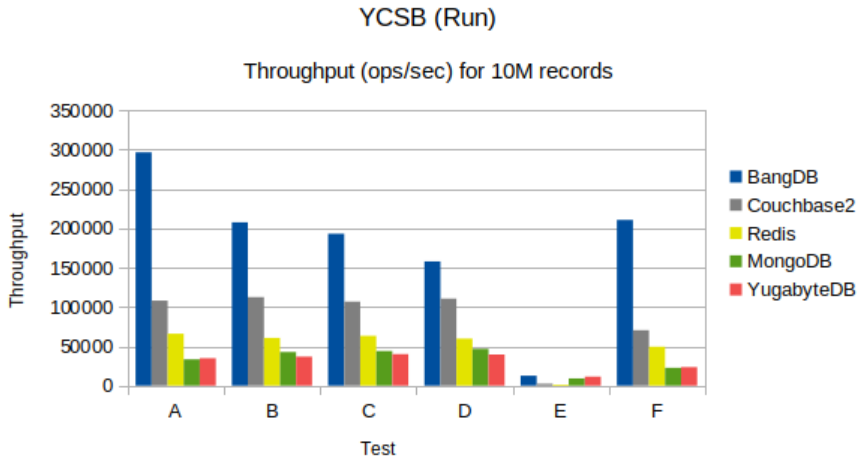
- Redis Server: 5.07, x86/64
- MongoDB server: 4.4.2, x86_64
- YugabyteDB:2.5.0, x86_64
- Couchbase2: 7.0 Beta, x86_64
- BangDB server: 2.0.0, x86_64
- Number of records: 10M
- RAM: 32GB, Cores: 16
- YCSB workloads: see github.com/brianfrankcooper/YCSB/wiki/Core-Workloads

Each of these workload test runs in two steps, 1. Load and 2. Run. Load stage is to load the data and then run stage we run the test. I have run each test with clean database to reflect the numbers in fair manner

Summary

To summarize the test, here is the high-level report of the tests, load and run both.

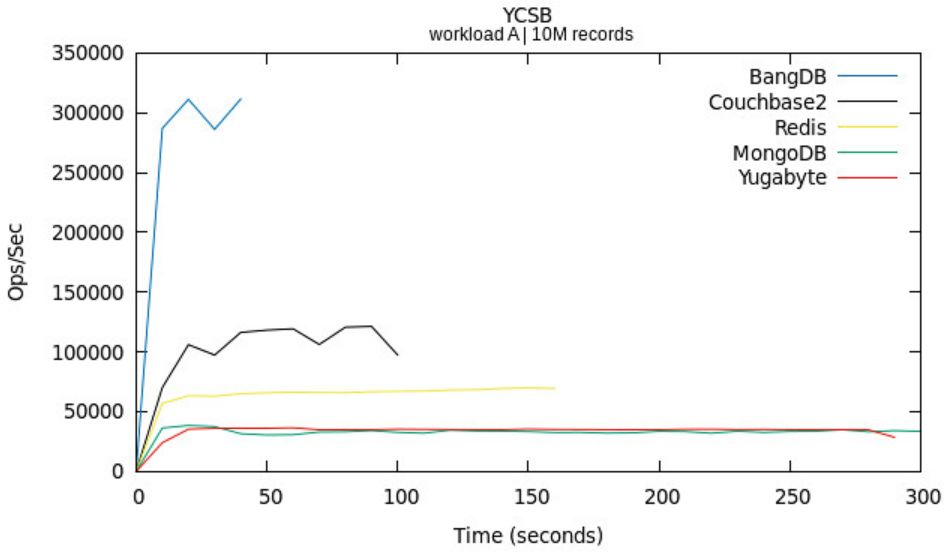


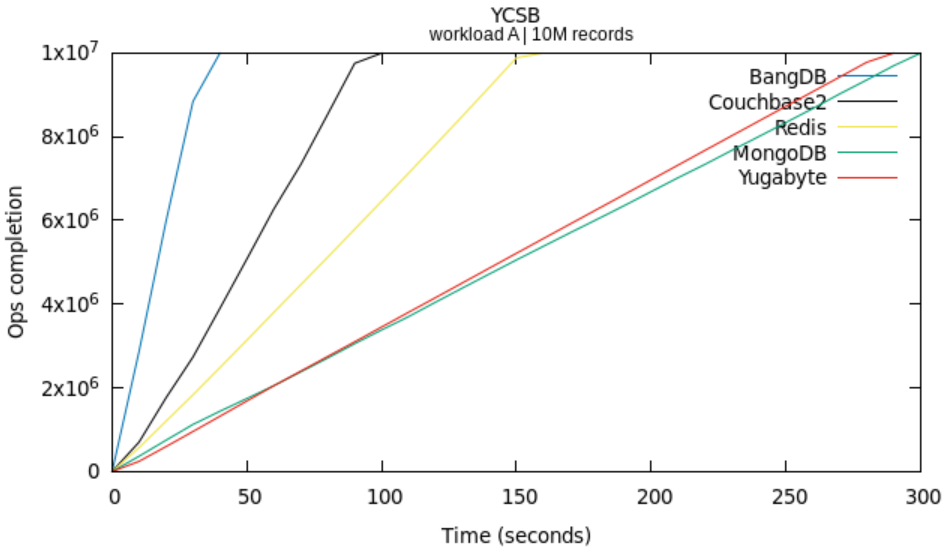


Load is consistent for all dbs for all tests as expected as this phase is to load the data. Run phase is where each db is tested for different test conditions.

Workload A: Update Heavy Workload

This workload has a mix of 50/50 reads and writes. An application example is a session store recording recent actions.





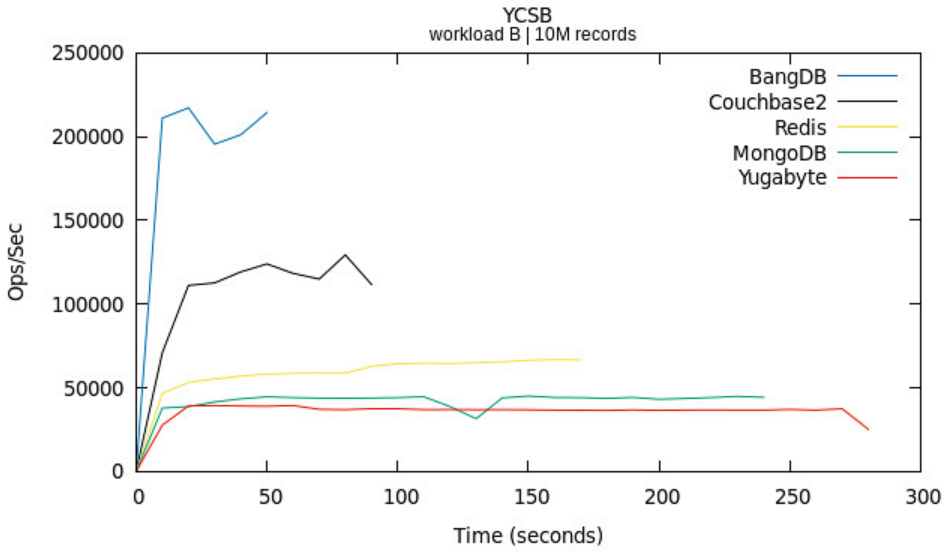
The first graph shows the ops/sec (throughput) for the 10M records. However the second chart shows how quickly the test was completed by DBs.

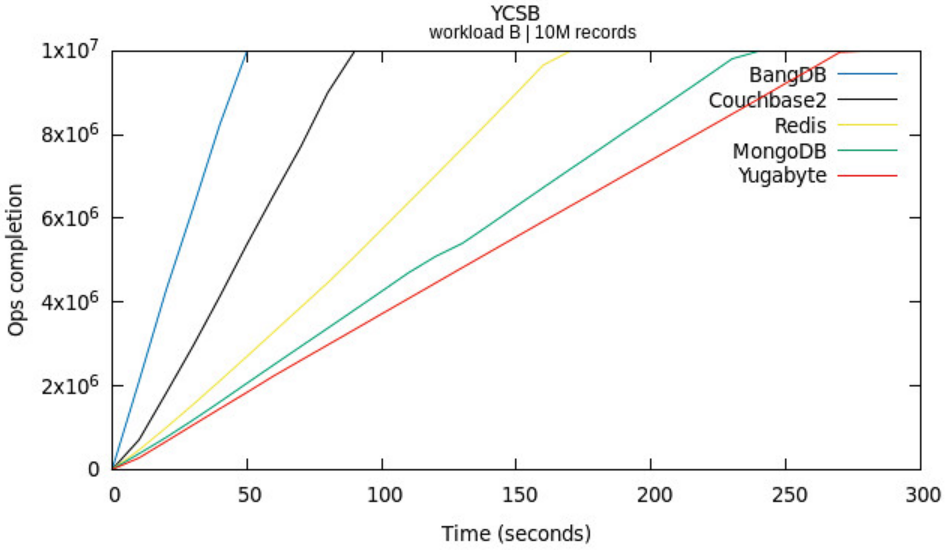
DB	Update Latency [us]		Read Latency [us]	
	Avg	99th perc	Avg	99th perc
MongoDB	53	352	11851	110719
Redis	3315	6591	3319	6587
Couchbase 2	2285	5107	2253	4799
Yugabyte	12071	335615	2214	6407
BangDB	769	3099	693	3053

We note that for MongoDB update latency is really very low (low is better) compared to other dbs, however the read latency is on the higher side.

Workload B: Read Mostly Workload

This workload has a 95/5 reads/write mix. Application example: photo tagging; add a tag is an update, but most operations are to read tags.



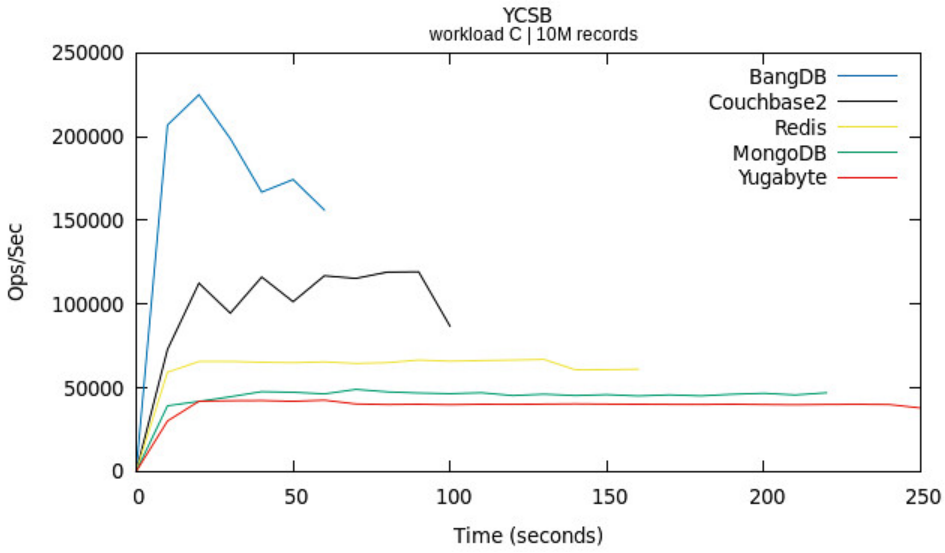


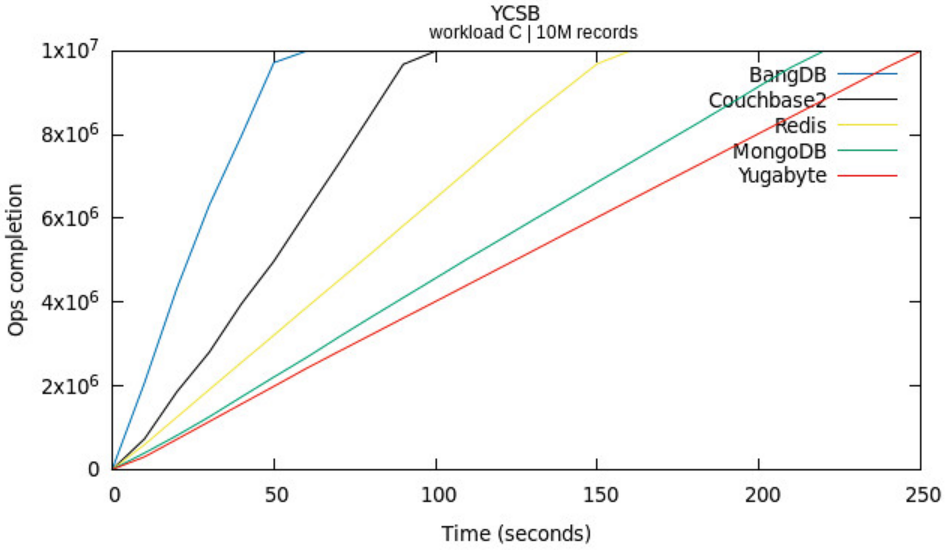
The latency table shows that 99th percentile latency for Yugabyte is quite high compared to others (lower is better)

DB	Update Latency [us]		Read Latency [us]	
	Avg	99th perc	Avg	99th perc
MongoDB	68	623	5409	8607
Redis	3610	7179	3613	7163
Couchbase 2	2200	4915	2150	4587
Yugabyte	6875	21327	6825	23471
BangDB	1260	4151	1217	4143

Workload C: Read Only

This workload is 100% read. Application example: user profile cache, where profiles are constructed elsewhere (e.g., Hadoop).



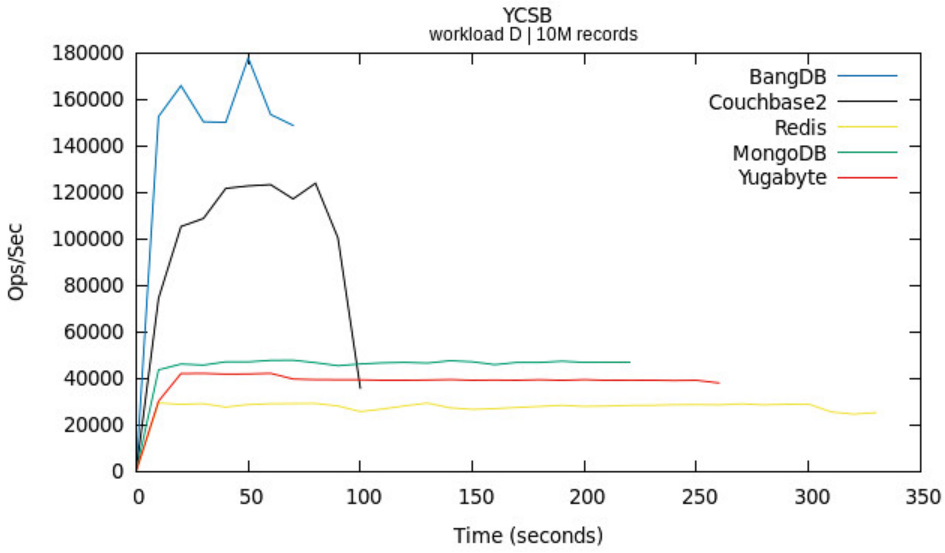


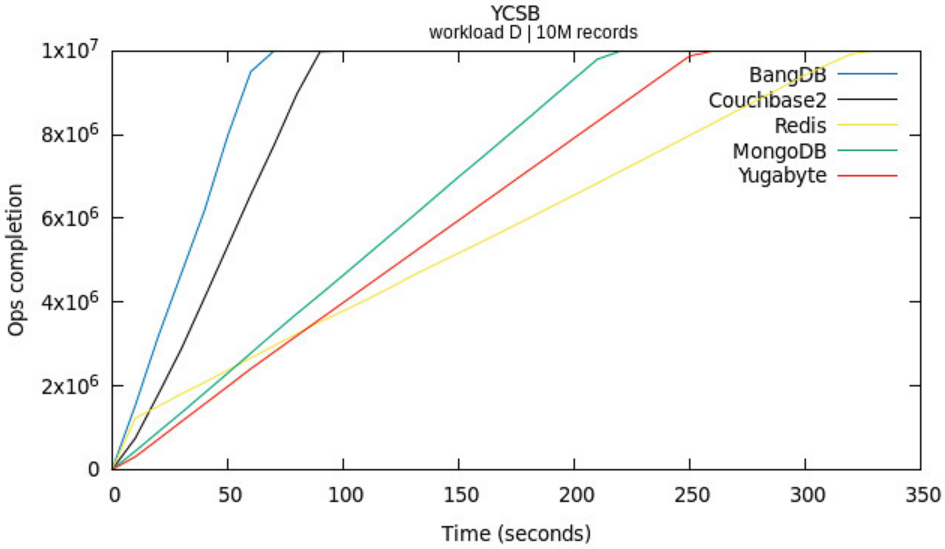
The latency table is following for test C and since it was read only test hence there is no update latency figure here. Again Yugabyte latency is quite high

DB	Update Latency [us]		Read Latency [us]	
	Avg	99th perc	Avg	99th perc
MongoDB			2787	5171
Redis			1978	4001
Couchbase 2			2276	4855
Yugabyte			6268	22543
BangDB			1308	4231

Workload D: Read Latest Workload

In this workload, new records are inserted, and the most recently inserted records are the most popular.
Application example: user status updates; people want to read the latest.



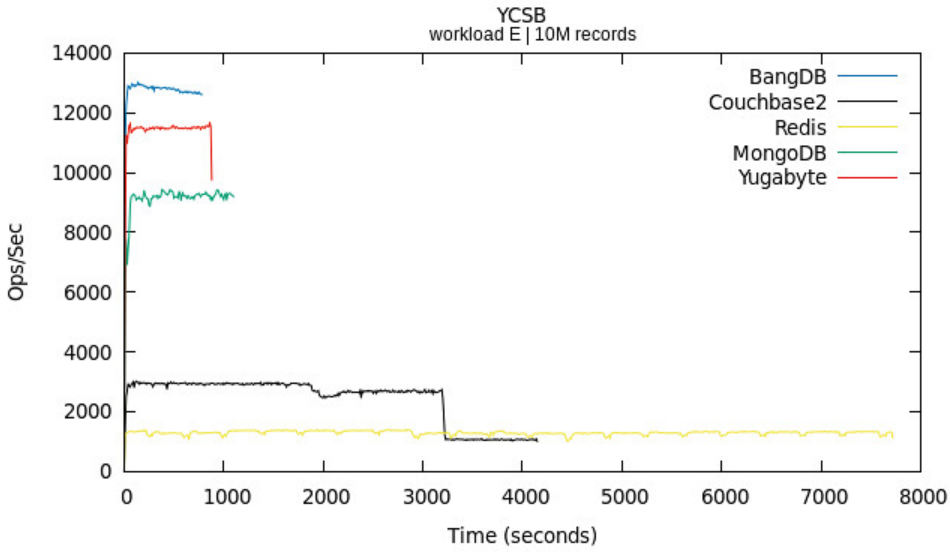


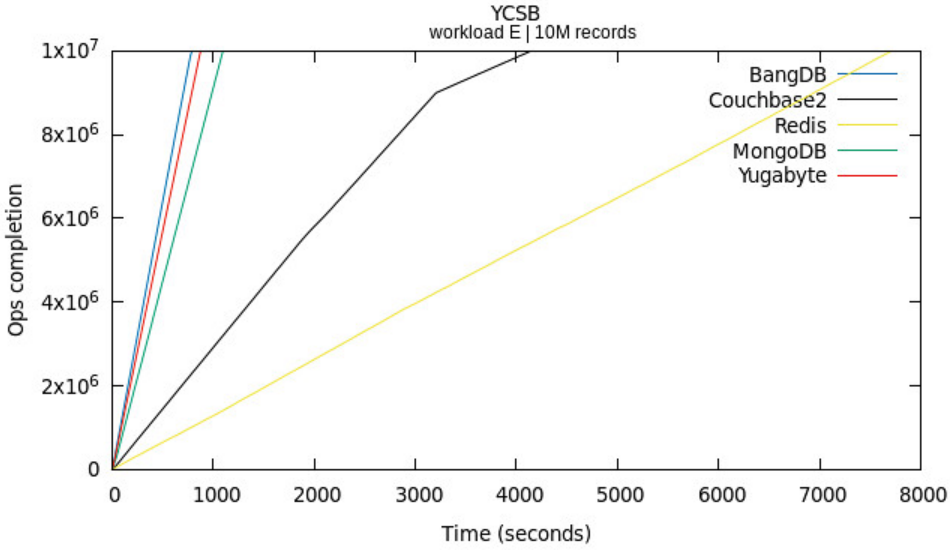
The latency table for test D is as below. Here Redis and Yugabyte have higher latencies, Yugabyte performs bad for both Insert and Read for the test

DB	Insert Latency [us]		Read Latency [us]	
	Avg	99th perc	Avg	99th perc
MongoDB	87	1147	4951	9495
Redis	6387	11807	3178	6499
Couchbase 2	2265	4999	2167	4487
Yugabyte	6874	21055	6365	21551
BangDB	1696	4539	1598	4423

Workload E: Short Ranges

In this workload, short ranges of records are queried, instead of individual records. Application example: threaded conversations, where each scan is for the posts in a given thread (assumed to be clustered by thread id).



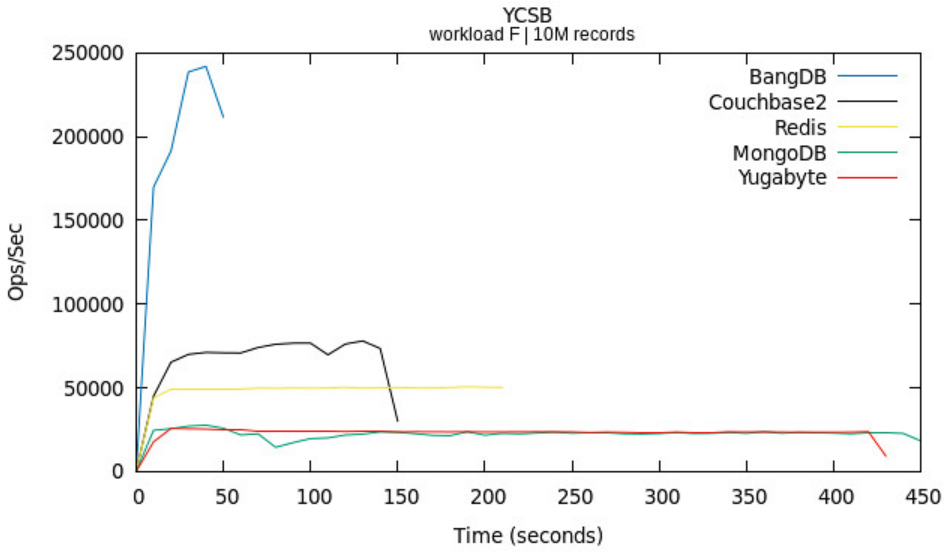


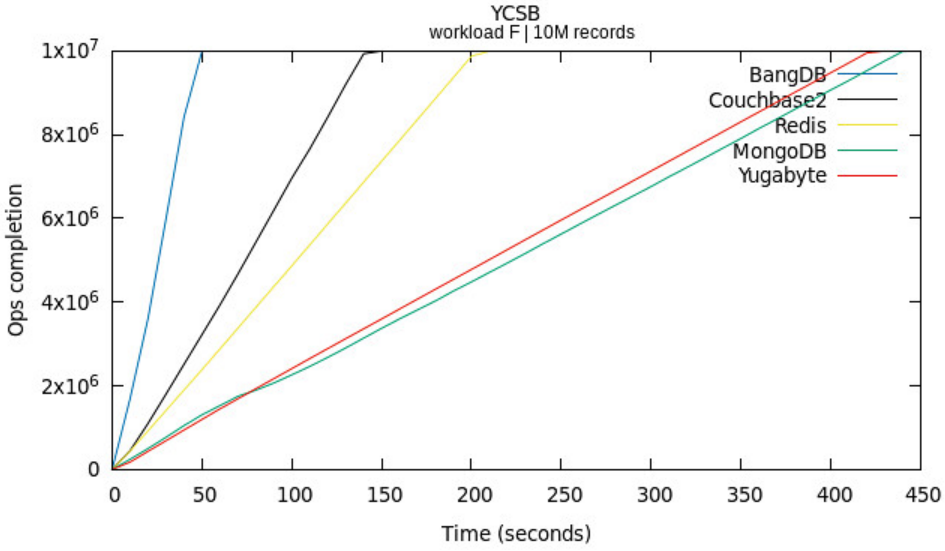
This requires a lots of scans, hence the throughput decreases for all dbs, however Redis is slowest, understandably so as it is also reflected in the latency table below. Yugabyte performs really good in this test

DB	Insert Latency [us]		Scan Latency [us]	
	Avg	99th perc	Avg	99th perc
MongoDB	58	107	23043	33599
Redis	6353	12463	161738	345599
Couchbase 2	19275	56063	87289	159743
Yugabyte	14705	61663	23156	85951
BangDB	17010	19791	19530	24287

Workload F: Read-Modify-Write

In this workload, the client will read a record, modify it, and write back the changes. Application example: user database, where user records are read and modified by the user or to record user activity.





Here Yugabyte has high latency esp for 99th percentile for Update and Read-modify-write. However, Very high Read latency for MonoDB makes it the last db to finish the task

DB	Update Latency [us]		Read-Modify-write Latency [us]		Read Latency [us]	
	Avg	99th perc	Avg	99th perc	Avg	99th perc
MongoDB	72	619	8825	69759	8743	68927
Redis	2682	5275	5376	10447	2690	5275
Couchbase 2	2290	5395	4549	10159	2255	5191
Yugabyte	15166	417279	18357	425215	3184	10559
BangDB	832	3499	1617	6707	779	3485

Conclusion

While each database has been designed for different goals and use cases, YCSB test provides somewhat a common ground for the benchmark, therefore the numbers shown in this document can be used by developers or users to help select the db suitable for their requirement. All of these dbs are available free of cost for download / install and it will be fairly straightforward to run these tests in your environment for further analysis. The tests can be modified or added in order to cover a set of specific scenarios as needed.

Todd Hoff | 6 Comments | Permalink | Share Article Print Article Email Article

Tuesday

Feb022021

Sponsored Post: 3T, Bridgecrew, Toptal, IP2Location, Ipdata, StackHawk, InterviewCamp.Io, Educative, Triplebyte, Stream, Fauna

TUESDAY, FEBRUARY 2, 2021 AT 10:17AM

Who's Hiring?

- Looking to rapidly **hire Top Software Developers**? Get Started with Toptal. **Toptal** will match you with top-quality, pre-screened freelance software developers that meet your project requirements. All in under 48 hours. Get started right away with a no risk trial.

- InterviewCamp.io has hours of system design content. They also do live system design discussions every week. They break down interview prep into fundamental building blocks. Try out their platform.

- **Triplebyte** lets exceptional software engineers skip screening steps at hundreds of top tech companies like **Apple, Dropbox, Mixpanel, and Instacart**. Make your job search *O*(1), not *O*(*n*). Apply here.

- **Need excellent people? Advertise your job here!**

Cool Products And Services

- Discover the MongoDB data masking tool in **Studio 3T Enterprise**. Enable data compliance and bolster security with powerful field-level data obfuscation. Try for free!

- **Bridgecrew** is the **cloud security platform for developers**. By leveraging automation and delivering security-as-code, Bridgecrew empowers teams to find, fix, and prevent misconfigurations in deployed cloud resources and in infrastructure as code. Get started for free!

If you are interested in a sponsored post for an event, job, or product, please** contact **us for more information.

Make Your Job Search *o*(1) — Not *o*(*n*)

Triplebyte is unique because they're a team of engineers running their own centralized technical assessment. Companies like **Apple, Dropbox, Mixpanel, and Instacart** now let Triplebyte-recommended engineers skip their own screening steps.

We found that **High Scalability readers** are about 80% more likely to be in the top bracket of engineering skill.

Take Triplebyte's multiple-choice quiz (system design and coding questions) to see if they can help you scale your career faster.

If you are interested in a sponsored post for an event, job, or product, please contact us for more information.

Monday
Feb012021

Stuff The Internet Says On Scalability For February 1st, 2021

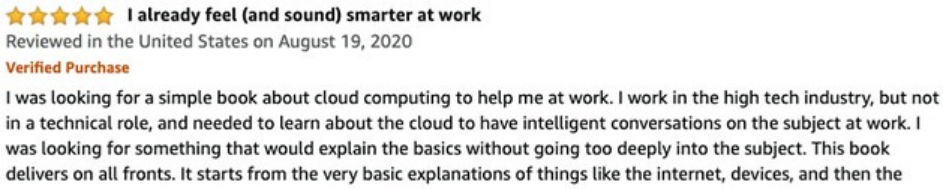
MONDAY, FEBRUARY 1, 2021 AT 9:15AM

Hey, it's HighScalability time once again!

Amazon converts expenses into revenue by transforming needs into products. Take a look at a fulfillment center and you can see the need for Outpost, machine learning, IoT, etc, all dogfooded. Willy Wonka would be proud.

Do you like this sort of Stuff? Without your support on Patreon this Stuff won't happen.

Know someone who could benefit from becoming one with the cloud? I wrote Explain the Cloud Like I'm 10 just for them. On Amazon it has 238 mostly 5 star reviews. Here's a review that has not been shorted by a hedge fund:



★★★★★ **I already feel (and sound) smarter at work**
Reviewed in the United States on August 19, 2020
Verified Purchase
I was looking for a simple book about cloud computing to help me at work. I work in the high tech industry, but not in a technical role, and needed to learn about the cloud to have intelligent conversations on the subject at work. I was looking for something that would explain the basics without going too deeply into the subject. This book delivers on all fronts. It starts from the very basic explanations of things like the internet, devices, and then the

Number Stuff:

Don't miss all that the Internet has to say on Scalability, click below and become eventually consistent with all scalability knowledge (which means this post has many more items to read so please keep on reading)...

Click to read more ...

Todd Hoff | 2 Comments | Permalink | Share Article Print Article Email Article

in hot links

Monday
Jan252021

Sponsored Post: 3T, Bridgecrew, Toptal, IP2Location, Ipdata, StackHawk, InterviewCamp.Io, Educative, Triplebyte, Stream, Fauna

MONDAY, JANUARY 25, 2021 AT 9:43AM

Who's Hiring?

- Looking to rapidly **hire Top Software Developers**? Get Started with Toptal. **Toptal** will match you with top-quality, pre-screened freelance software developers that meet your project requirements. All

in under 48 hours. Get started right away with a no risk trial.

- InterviewCamp.io has hours of system design content. They also do live system design discussions every week. They break down interview prep into fundamental building blocks. Try out their platform.

- **Triplebyte** lets exceptional software engineers skip screening steps at hundreds of top tech companies like **Apple, Dropbox, Mixpanel, and Instacart**. Make your job search *O*(1), not *O*(*n*). Apply here.

- **Need excellent people? Advertise your job here!**

Cool Products And Services

- Discover the MongoDB data masking tool in **Studio 3T Enterprise**. Enable data compliance and bolster security with powerful field-level data obfuscation. Try for free!

- **Bridgecrew** is the **cloud security platform for developers**. By leveraging automation and delivering security-as-code, Bridgecrew empowers teams to find, fix, and prevent misconfigurations in deployed cloud resources and in infrastructure as code. Get started for free!

- **IP2Location** is **IP address geolocation service provider** since 2002. The geolocation database or API detects location, proxy and other >20 parameters. The technology has been cited in more than 700 research papers and trusted by many Fortune 500 companies. Try it today!

- **ipdata** is a **reliable IP Address Geolocation API** that allows you to lookup the approximate location of any IP Address, detect proxies and identify a company from an IP Address. Trusted by 10,000+ developers. Try it now!

- Developers care about shipping secure applications. Application security products and processes, however, have not kept up with advances in software development. There are a new breed of tools hitting the market that enable developers to take the lead on AppSec. Learn how engineering teams are using products like **StackHawk** and **Snyk** to add security bug testing to their CI pipelines.

- Learn the stuff they don't teach you in the AWS docs. Filter out the distracting hype, and focus on

the parts of AWS that you'd be foolish not to use. Learn the Good Parts of AWS. Created by former senior-level AWS engineers of 15 years.

- Stateful JavaScript Apps. Effortlessly add state to your Javascript apps with **FaunaDB**. Generous free tier. Try now!

- Learn to balance architecture trade-offs and design scalable enterprise-level software. Check out **Educative.io**'s bestselling new 4-course learning track: Scalability and System Design for Developers. Join more than 300,000 other learners.

- Build, scale and personalize your news feeds and activity streams with getstream.io. Try the API now in this 5 minute interactive tutorial. **Stream** is free up to 3 million feed updates so it's easy to get started. Client libraries are available for Node, Ruby, Python, PHP, Go, Java and .NET. Stream is currently also hiring Devops and Python/Go developers in Amsterdam. More than 400 companies rely on Stream for their production feed infrastructure, this includes apps with 30 million users. With your help we'd like to ad a few zeros to that number. Check out the job opening on AngelList.

- **Advertise your product or service here!**

If you are interested in a sponsored post for an event, job, or product, please** contact **us for more information.

Make Your Job Search *O*(1) — Not *O*(*N*)

Triplebyte is unique because they're a team of engineers running their own centralized technical assessment. Companies like **Apple, Dropbox, Mixpanel, and Instacart** now let Triplebyte-recommended engineers skip their own screening steps.

We found that **High Scalability readers** are about 80% more likely to be in the top bracket of engineering skill.

Take Triplebyte's multiple-choice quiz (system design and coding questions) to see if they can help you scale your career faster.

If you are interested in a sponsored post for an event, job, or product, please contact us for more information.

Todd Hoff | Post a Comment | Permalink | Share Article Print Article Email Article

Tuesday
Jan052021

Sponsored Post: Bridgecrew, Toptal, IP2Location, Ipdata, StackHawk, InterviewCamp.Io, Educative, Triplebyte, Stream, Fauna

TUESDAY, JANUARY 5, 2021 AT 8:37AM

Who's Hiring?

- Looking to rapidly **hire Top Software Developers**? Get Started with Toptal. **Toptal** will match you with top-quality, pre-screened freelance software developers that meet your project requirements. All in under 48 hours. Get started right away with a no risk trial.

- InterviewCamp.io has hours of system design content. They also do live system design discussions every week. They break down interview prep into fundamental building blocks. Try out their platform.

- **Triplebyte** lets exceptional software engineers skip screening steps at hundreds of top tech companies like **Apple, Dropbox, Mixpanel, and Instacart**. Make your job search *O*(1), not *O*(*n*). Apply here.

- **Need excellent people? Advertise your job here!**

Cool Products And Services

- **Bridgecrew** is the **cloud security platform for developers**. By leveraging automation and delivering security-as-code, Bridgecrew empowers teams to find, fix, and prevent misconfigurations in deployed cloud resources and in infrastructure as code. Get started for free!

- **IP2Location** is **IP address geolocation service provider** since 2002. The geolocation database or API detects location, proxy and other >20 parameters. The technology has been cited in more

Make Your Job Search *o*(1) — Not *o*(*n*)

Triplebyte is unique because they're a team of engineers running their own centralized technical assessment. Companies like **Apple, Dropbox, Mixpanel, and Instacart** now let Triplebyte-recommended engineers skip their own screening steps.

We found that **High Scalability readers** are about 80% more likely to be in the top bracket of engineering skill.

Take Triplebyte's multiple-choice quiz (system design and coding questions) to see if they can help you scale your career faster.

If you are interested in a sponsored post for an event, job, or product, please contact us for more information.

Todd Hoff | Post a Comment | Permalink | Share Article Print Article Email Article

in sponsored post

Wednesday
Dec232020

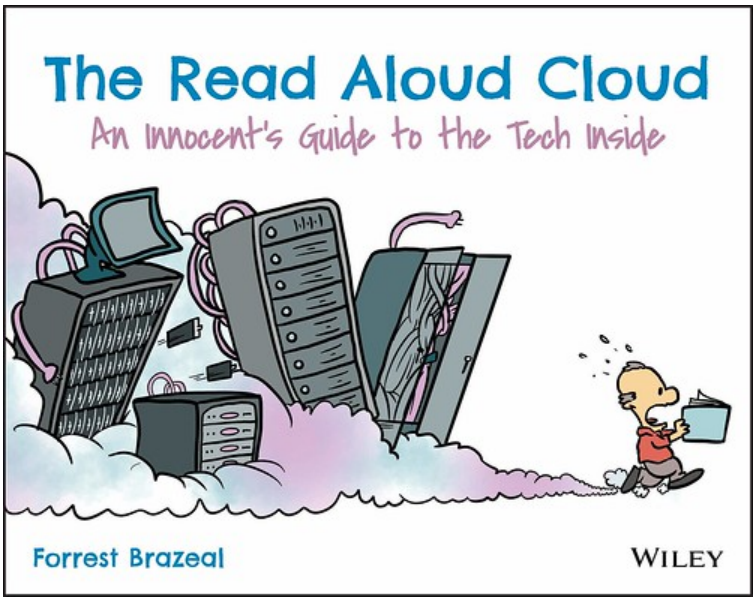
The Read Aloud Cloud: An Interview With Forrest Brazeal On His New Book

WEDNESDAY, DECEMBER 23, 2020 AT 9:16AM

It's Christmas time and you've been wracking your brain trying to find the perfect gift that will teach your loved ones about the cloud in a simple and entertaining way. What to do?

Fortunately for you, Santa has a new elf this year—Forrest Brazeal—who is part AWS Serverless Hero, part skilled cartoonist, and part cloud guru.

Yes, it's a cartoon book about the cloud!



The Read Aloud Cloud

No, I didn't think it could be done either, but Forrest pulled it off with a twinkle in his eyes and little round belly that shook when he laughed, like a bowl full of jelly.

One of the many five-star reviews:

> *The Read Aloud Cloud is a delightful book. It's 165 pages of hand-drawn cartoons, entertaining verse, and hard-won wisdom.*
>
> *It truly is a load of fun to flip through. I read it to my kids (8 & 6), and they love it. I'll share it with my parents so they can finally understand what I do. I learned a ton about the history of computing and of course all the ways that we as humans stumble through making our computers do what we want.*

Here's my email interview with Forrest Brazeal on The Read Aloud Cloud: An Innocent's Guide to the Tech Inside. Enjoy. And happy Christmas to all, and to all a good night!

Please Tell Us Who You Are And What You've Brought To Show

And Tell Today?

Click to read more ...

Todd Hoff | Post a Comment | Permalink | Share Article Print Article Email Article

in Book

Tuesday
Dec222020

Sponsored Post: Toptal, IP2Location, Ipdata, StackHawk, InterviewCamp.Io, Educative, Triplebyte, Stream, Fauna

TUESDAY, DECEMBER 22, 2020 AT 12:39PM

Who's Hiring?

- Looking to rapidly **hire Top Software Developers**? Get Started with Toptal. **Toptal** will match you with top-quality, pre-screened freelance software developers that meet your project requirements. All in under 48 hours. Get started right away with a no risk trial.

- InterviewCamp.io has hours of system design content. They also do live system design discussions every week. They break down interview prep into fundamental building blocks. Try out their platform.

- **Triplebyte** lets exceptional software engineers skip screening steps at hundreds of top tech companies like **Apple, Dropbox, Mixpanel, and Instacart**. Make your job search *O*(1), not *O*(*n*). Apply here.

- **Need excellent people? Advertise your job here!**

Cool Products And Services

- **IP2Location** is **IP address geolocation service provider** since 2002. The geolocation database or API detects location, proxy and other >20 parameters. The technology has been cited in more than 700 research papers and trusted by many Fortune 500 companies. Try it today!

Make Your Job Search *o*(1) — Not

Saturday
Dec192020

Stuff The Internet Says On Scalability For December 19th, 2020

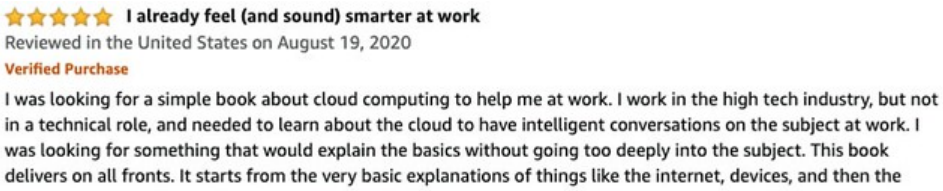
SATURDAY, DECEMBER 19, 2020 AT 10:03AM

Hey, it's HighScalability time once again!

NASA

Do you like this sort of Stuff? Without your support on Patreon this Stuff won't happen.

Know someone who could benefit from becoming one with the cloud? I wrote Explain the Cloud Like I'm 10 just for them. On Amazon it has 212 mostly 5 star reviews. Here's a load-balanced and fault-tolerant review:



★★★★★ **I already feel (and sound) smarter at work**

Reviewed in the United States on August 19, 2020

Verified Purchase

I was looking for a simple book about cloud computing to help me at work. I work in the high tech industry, but not in a technical role, and needed to learn about the cloud to have intelligent conversations on the subject at work. I was looking for something that would explain the basics without going too deeply into the subject. This book delivers on all fronts. It starts from the very basic explanations of things like the internet, devices, and then the

Number Stuff:

Don't miss all that the Internet has to say on Scalability, click below and become eventually consistent with all scalability knowledge (which means this post has many more items to read so please keep on reading)...

Click to read more ...

Todd Hoff | 1 Comment | Permalink | Share Article Print Article Email Article

in hot links