The Advantages of Redis™ as an In-Memory Database for Low-Latency Applications

Overview

In-memory databases are able to store and react to changes in your data in order of magnitude faster than even databases that store data on solid state drives, allowing single thread benchmark speeds of over 140,000 storage operations in under 1 millisecond, and the same number of reads in under 0.1 milliseconds on consumer grade hardware. This raw speed advantage enables in-memory databases to address a different set of use cases like intelligent caching, accelerated user experiences, and adaptive traffic shaping. In-memory databases can persist data on disks by storing each operation in a log or by taking snapshots.

This white paper

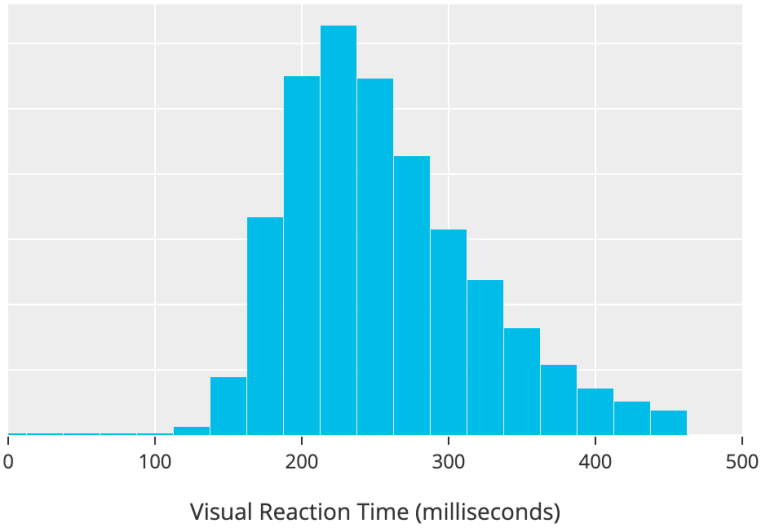
- analyzes the business value derived from low latency applications,
- examines some architectural patterns for in-memory databases, and finally
- looks at the leading in-memory database—Redis™, and offers our recommendations.

instaclustr
Now part of **Spot by NetApp**

Business Impact from Low-Latency Applications

In-memory databases can deliver considerable business value through the simple act of very quickly storing data that needs to be repeatedly referred to and thereby accelerating user interfaces (UI) and user experiences (UX). This section will provide a brief overview of the effect of latency on retail, technology, and other applications to demonstrate there is considerable value to be gained from accelerating your infrastructure. These potential gains go beyond in-memory databases and include scaling everything in your data layer appropriately to maintain application performance.

It takes a human being around a quarter of a second to press a button as soon as they can after a visual stimulus they've been waiting for occurs.[1]



Visual Reaction Time Data

If about 250 milliseconds is required for this kind of loop—of perception, decision, and action—then perception alone is occurring much sooner. You can also think of the most common frame rate of film, 24 individual pictures per second, which creates an effective illusion of motion by deploying a new picture around every 42 milliseconds.

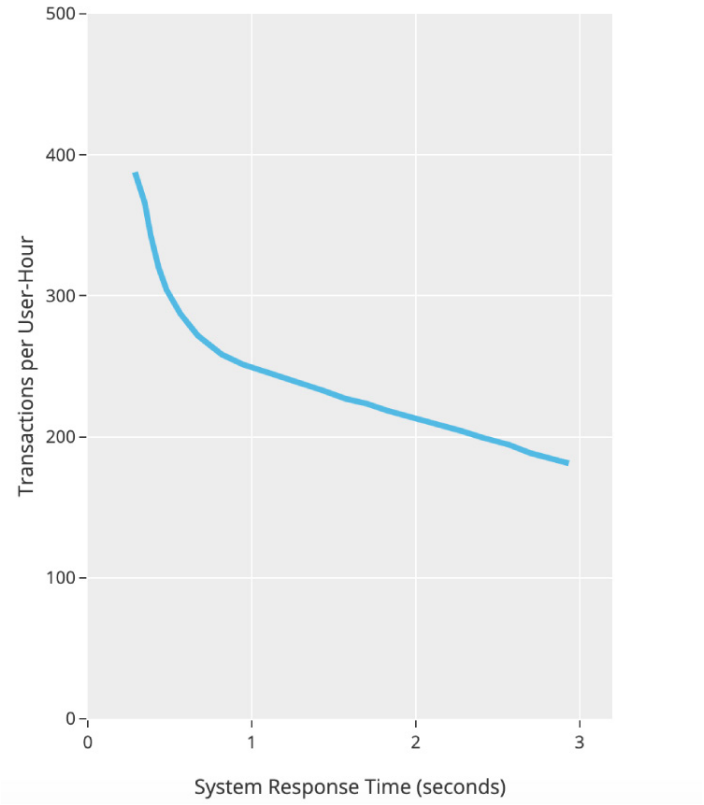
In 1968 Robert Miller at IBM suggested a series of task and context dependent response times for user interfaces which included 100 milliseconds for acknowledging an action while the user is looking at the screen and "no more than one second" before some readable text is displayed after a "next page" type action.[2]

---

[1] A scientifically controlled study, of 120 healthy young medical students recorded an average visual reaction time of 247ms. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4456887/ The Human Benchmark website with 81 million data points collected from the public, variously using computer mice and trackpads, has the average at 284ms and the median at 273ms. https://humanbenchmark.com/tests/reactiontime/statistics a sample of that data is plotted in the histogram show.

[2] https://dl.acm.org/doi/10.1145/1476589.1476628

Doherty and Thadani, also at IBM, showed large productivity gains from computer system latency reduction in 1982.[3]



Productivity and UX Response Time

In their study they also showed "average" users beating "expert" users, and likewise "novice" users beating "average" users, with only a 250-500 ms latency advantage. All this occurred in the first 1000 ms of latency identified by the curve above. Large productivity gains were also shown when latency was reduced for a variety of tasks.

In 1987 Rob Akscyn reported a similar real world usability threshold of 250 milliseconds for the hypertext systems he had been building.[4] In 1993 Nielsen formulated three orders of magnitude (0.1s, 1s, and 10s) as a rough guide to UIs being perceived as instantaneous, within someone's flow of thought, or distracting.[5]

This research shows the kinds of speeds at which human beings are able to perceive information visually. Empirical evidence also shows people are much more productive using computer systems that react closer to their perceptual speed. Below we look at the copious evidence that latency has a large impact across a number of important business variables and key performance indicators for consumer facing applications.

[3] https://jlelliotton.blogspot.com/p/the-economic-value-of-rapid-response.html
[4] https://www.eastgate.com/HypertextNow/archives/Akscyn.html
[5] https://www.nngroup.com/articles/response-times-3-important-limits/

Retail and Latency

Online shopping examples are linked very closely to revenue and serve as a great benchmark of the effects of speed on human decision processes. The variables affected are revenue, conversions, and bounce rates (people clicking the back button/abandoning the site).

Amazon found that increasing website latency by **+100 milliseconds** led to a **1 percent loss of revenue**.[6]

Walmart found that changing page load times by **-1000 milliseconds** led to **2 percent gain in conversions**, with a similar effect on revenue as Amazon.[7]

Etsy found in an experiment that adding weight to their mobile site of **+160 kilobytes** led to a **12% increase in the bounce rate** of people clicking away from their site.[8]

Staples changed their median page load time by **-1000 milliseconds** and specifically addressed the experience of the 2% of users suffering the largest delays and saw a **10% increase to conversions**.[9]

Akamai ran an analysis over billions of user sessions, on multiple retail sites, and found the inflection point for decreased conversions and an increased bounce rate was around **2000 milliseconds**.[10]

Technology Products and Latency

Beyond retail, large technology corporations have looked at the engagement and revenue effects of latency. Decreased use and traffic, people swapping to other services, conversions, and revenue all seem to be affected.

Google found that **adding 500 milliseconds** to search page loads **decreased searches by 25%**.[11]

Yahoo! found that adding **400 milliseconds of delay** to their pages **dropped traffic by 9%**.[12]

---

[6] http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html and slides by the same author https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxnbGluZGVufGd4OjVmZDlzMWMzMGI4MDc3OWM
[7] https://www.slideshare.net/devonauerswald/walmart-pagespeedslide
[8] http://radar.oreilly.com/2014/01/web-performance-is-user-experience.html
[9] https://www.slideshare.net/cliffcrocker/velocity-ny-how-to-measure-revenue-in-milliseconds
[10] https://blogs.akamai.com/2017/04/new-findings-the-state-of-online-retail-performance-spring-2017.html
[11] http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html
[12] https://www.slideshare.net/stoyan/yslow-20-presentation

Bing found that **adding 2000 milliseconds** of delay to search caused a **decrease in searches of 1.8%** and a **decrease in revenue of 4.3%**.[13]

For software **Mozilla**, the makers of the open source Firefox browser, found that **removing 2200 milliseconds** of delay to a page for downloading their application **increased conversions by 15.4%**.[14]

Information Consumption and Latency

Beyond search or application downloads, engagement on sites that offer text or images for users to enjoy has also been demonstrated to be highly sensitive to latency.

An experiment at the **Financial Times** found that **adding 1000 milliseconds of delay** led to a **4.9% decrease** in pageviews. Similarly adding a **3000 millisecond delay** induced a **7.2% reduction** in pageviews.[15]

Instagram made a number of optimizations to the weight of their pages, which **reduced wait times by 30-70 milliseconds** which led to a **0.7% increase in impressions**.[16]

An experiment conducted with the users of **The Telegraph** showed **4000 milliseconds of delay caused an 11% reduction in pageviews**. Doubling that to **8,000 milliseconds of delay** caused a **17.5% drop in pageviews**.[17]

GQ magazine conducted an overhaul of their slow site and **removing 5000 milliseconds** of wait time led to **83% more traffic**, **32% more time on the site, and a 108% increase in ad interactions**.[18]

Gaming and Latency

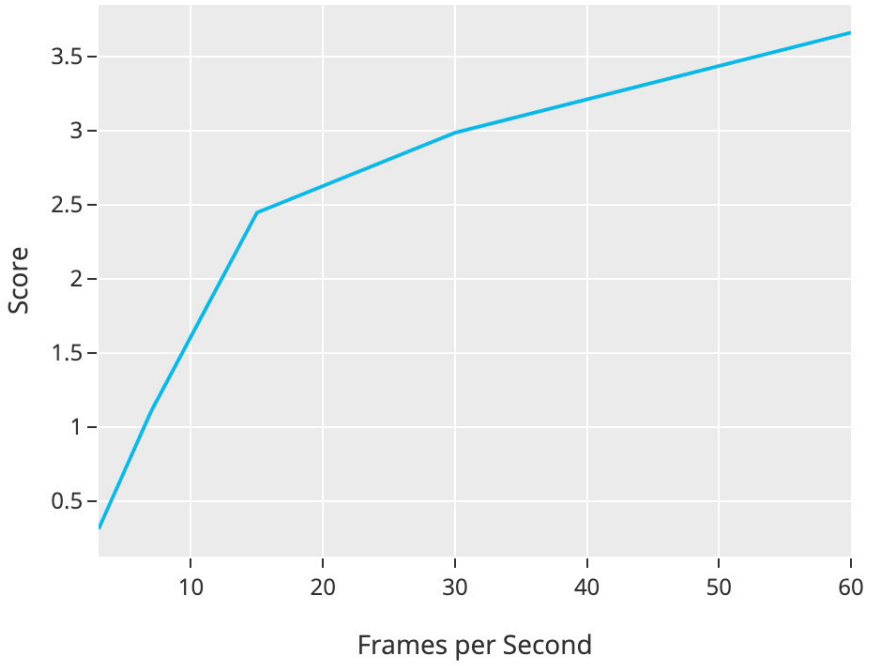
If shopping, searching, and reading are sensitive to latency it's not surprising that gaming is as well. Survey and experimental work on gaming shows that there is a very strong belief among people that latency affects gaming, and that **after 100ms of latency both real-world scores and reported enjoyment decline starkly**.[19]

In a study published in 2007 a controlled experiment with a first person shooter game was

[13] http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html
[14] https://blog.mozilla.org/metrics/2010/04/05/firefox-page-load-speed-part-ii/
[15] https://medium.com/ft-product-technology/a-faster-ft-com-10e7c077dc1c
[16] https://instagram-engineering.com/performance-usage-at-instagram-d2ba0347e442#.5snxp5z1t
[17] https://blog.optimizely.com/2016/07/13/how-does-page-load-time-impact-engagement/
[18] https://digiday.com/media/gq-com-cut-page-load-time-80-percent/
[19] https://web.cs.wpi.edu/~claypool/mqp/latency-comp/imcneil-mbeyler-mqpreport.pdf

conducted with variable frame rates. The different frame rates were:

- 3, approx. one frame per 333ms
- 7, approx. one frame per 142ms
- 15, approx. one frame per 66ms
- 30, approx. one frame per 33ms
- 60, approx. one frame per 17ms



Effect of Frames Per Second on Score[20]

Performance increased in large steps between 3, 7, and 15 frames per second. The gaps between frames at 15 frames per second finally goes below the 100 ms threshold identified in other research as being important for instantaneous feeling experiences. Performance continues to improve with more frames beyond this point, but at a slower pace.

Take-Aways and the Time Budget

Linear extrapolations cannot be made from any individual case but together these cases provide a formidable body of evidence showing that users are indeed highly sensitive to delays, at quite granular levels, across a number of domains.

[20] Claypool & Claypool, 2007 https://web.cs.wpi.edu/~claypool/papers/fr/fulltext.pdf

Speed has an impact whether users are looking to get work done, purchase a product, search, download an app, read an article, scroll through a social feed, or play a game. Large double digital percentage impacts on key business performance measures can be seen and even incremental improvements of 100 milliseconds can have visible effects on engagement, conversions, and revenue.

It's possible to conclude from these cases that many large organizations could gain significant revenue and small organizations have the potential to accelerate their user growth and market share by paying attention to any user interfaces, and likely whole user experiences, that are in any way slow. To a degree this applies across all applications and data layers beneath them, but it is particularly relevant to in-memory databases situated directly behind applications and user experiences.

We argue it's vital to measure and establish a UX time budget for any application sensitive to these elements. Such a budget ensures that changes, like new features, are evaluated not just with their potential upside but also with regard to their impact on your time budget.

Measurement can be done simply from literal stopwatches to adding programmatic timers to various functions and processes. Products like Elasticsearch™, with Kibana or Grafana, are popular for ingesting, exploring, and visualizing large amounts of application performance monitoring (APM) data or user logs. With this data in hand it is important to remember to not only examine the averages but also to profile the slowest groups of actions and user experiences. In the next section we consider the most common kinds of architectural patterns we see in the in-memory database space.

In-Memory Architecture Patterns

In this section we will look at different architectural patterns for the deployment of an in-memory database. The unique speed of in-memory storage and structures allows not only for more responsive applications, but for more value to be added without depleting your limited time budget.

Intelligent Caching

This pattern addresses the problems identified above head on. It is important to make a distinction between the caching of substantial static resources like images or video files and the caching of other important objects. Caching an image file might be better achieved through a web server optimization or even a content delivery network partner. But caching user sessions, a shopping cart, a list of who's online, a recommendation, or a user search can be extremely useful and accelerated with an in-memory database.

In these cases, there are always rules about how long something might be cached for, and rules around what to do when a cache is full. In this sense there is no "dumb caching" opposed to intelligent caching, only different rules and assumptions about what to do.

As a cache fills up which elements are going to be discarded? Powerful in-memory databases enable you to have different policies for different aspects of your data. For example, someone's session data or shopping cart contents might become a candidate for eviction in a brimming in-memory database under a naive "Least Frequently Used" eviction policy; however, from a business perspective it would be far better to consider dropping some of the least used cached search terms on the product search rather than making a user log-in again or lose their shopping cart.

Having the capacity to explicitly expire data at varying time intervals, make a distinction between "volatile" keys that are candidates for eviction and more permanent keys, and configurable expiration policies provide the basis for intelligent caching systems that automatically adapt to usage patterns and spikes in real-time.

Adaptive Traffic Shaping

Monitoring real-time traffic and adapting to it can be extremely difficult if the traffic is of high volume. In-memory databases have the speed to allow real-time responsiveness like enforcing resource limits, e.g. 10 requests per minute per API key on the free tier. This can be used to permit "burstability" like an "an average of 20 requests per minute with bursts to 40 requests permitted on the $10 tier". The inverse of this pattern is that traffic can be shaped according to resources rather than the users to ensure fixed resources slow down equally for all users rather than erroring out.

The traffic shaping pattern can be used to make your architecture aware of system load and sacrifice non-core features adaptively to protect the core user features. This might involve a miniscule section of your in-memory database holding logs and metrics of the last 5 seconds, or 5 minutes, that produces a load score. Other pieces of architecture can then be easily made aware of what they should or shouldn't be doing when load is high. The BBC uses this pattern to dynamically turn off non-core parts of their website experience during large live events.

Certain data structures like approximate counting methods allow you to keep an (approximate) count of extremely large sets of unique objects in very small spaces in ways that a simple count (not unique) or list of the items (too large) would be unable to do. It is important to make sure any database you are considering has the tools and data structures to address the use cases you might need it for.

Publish/Subscribe

This pattern allows for ephemeral messaging to be pushed out in a highly efficient way. Publishers push messages to channels, which subscribers listen to. The key element that makes it ephemeral is that a subscriber is not guaranteed to be listening when a message is sent.

This pattern is common on gaming platforms for scores and is used where message efficiency is paramount such as informing friends of users on a gamified learning platform of a user's achievement, inviting them to like and encourage. In this case it doesn't matter if a percentage of the users are offline, or engaged in a lesson themselves, just that it reaches some users and they have the opportunity to congratulate their friend and create a compelling social experience.

Streams and Stream Processing

Explicit stream processing data structures and tooling are relatively new to in-memory databases. However, it was not an uncommon use case in the previous generations of in-memory databases. The last generation simply used a list structure to implement queue (first in, first out) or stack (last in, first out) processing. The problem with queue or stack processing is that when processing events failed this was not tracked.

Inspired by the Apache Kafka® project the in-memory database Redis™ has added a far more robust stream processing architecture which uses "consumer groups", who subscribe to streams of data and explicitly acknowledge when they "check out" work and when their work is complete. This addresses the failure case in list based approaches, because if a consumer crashes it will not mark a parcel of work that it has checked out as complete, and eventually another consumer can take up the task and finish it.

We still recommend the robust power and durability of Apache Kafka for critical data and for enterprise grade integrations across a data layer (such as provided by Kafka Connect), but in-memory stream processing can be used close to the application or the user interface level to provide delightful 'non-blocking' experiences.

An example of this pattern would be the preview of an article you see when you are about to share an article on social media. Once the URL is detected the platform shows you a loading bar to indicate processing and behind the scenes the task of getting an article summary is put on a queue and eventually returns this to the browser. There is no need to burden a disk based database with this kind of message or to leave the user wondering if the system has stalled by doing it slowly or via a new page load. The operation is complex enough, pulling unknown data from the web, that it could crash the process, stream processing at high speed allows for this to be handled robustly.

Data Persistence

The downside of an in-memory database is linked to the basic properties of Random Access Memory (RAM) in that it is volatile. Some in-memory databases like Redis mitigate these downsides through configurable disk persistence, both through binary copies of what's in memory and append only logs of commands as they appear "on the wire".

If you are considering adopting an in-memory database it's important to understand its disk persistence options. If a large cache were to be empty after a restart then initially 100% traffic could pass through to the systems behind it and crash them as well. Disk persistence allows recovery from a crash with an already warm cache which could handle most of the load protecting your infrastructure. Additionally, the valuable user sessions and shopping carts mentioned above would persist.

Clustering

A cluster topology can increase the horizontal scalability of an in-memory database beyond the RAM size of a single node and enables you to store more. The only draw-back to cluster topology is the architecture, management, and support elements of this. In this sense any in-memory database that you insert into critical paths in your data layer requires engineering thought, and a support story.

Redis: A Fast and Industry Leading In-Memory Database

First released in 2009, Redis is the leading open source in-memory cache and database. The name Redis is derived from "Remote Dictionary Server". It is often called a data structure server because its core data types are similar to those found in programming languages like strings, lists, dictionaries (or hashes), sets, and sorted sets.

Latency is always a consideration and for speed nothing beats an in-memory database. Redis brings this speed and a simple interface to a variety of applications like storage, caching, messaging, and stream processing.

Maturity and Popularity

Over the last decade Redis has matured and its properties are well known. In StackOverflow's annual survey of developers it is the database technology voted "most loved" for the last four years in a row. Importantly this is not just a "fun" metric but the

phrasing is calculated from the percentage of users who are currently using the technology and also plan to use it again.[21]

On GitHub Redis is four times more followed and "forked" than the nearest competitor, which is Memcached. On stackshare.io Redis is used five times more frequently in technology stacks than Memcached. Indeed in StackOverflow's 2020 survey of over 40,000 professional developers 1 in 5 reported they were using Redis currently with no other in-memory database even clearing the 1 in 50 threshold to be reported.[22]

Usability and Featureset

Developers really do love Redis for its elegant design and features.

> *It's not simply easy to use; it's a joy. If an API is UX for programmers, then Redis should be in the Museum of Modern Art alongside the Mac Cube.*[23]

More than this the versatility of the feature set of Redis enables some of the architecture patterns above which are simply not possible on other reasonable, but simpler, platforms like Memcached. There is nothing wrong with these platforms but Redis with a superset of their features is able to do what they do as well as much more. This versatility is why you will see Redis not just referred to as a key value or NoSQL server but as a data structure server.

In addition to these structures Redis has also added pub/sub messaging and stream processing, providing a wealth of options that other in-memory databases lack.

Open Source

Redis, as with all other Instaclustr offerings and recommendations, is open source. Some providers offer proprietary extensions but our recommendation is always the same on this point. The data layer is a business input to your application and your value proposition. You should want as many of your business inputs to be completely commoditized in a market with price competition rather than locked to a single vendor where the roadmap, particularly price, is decided by a single entity. Providers of pure open source don't have the capacity to lock in their customers so they have to focus on the only levers available to them: price, SLAs, and customer happiness.

There are now leading open source technologies for all aspects of the data layer but Redis is unique in that it has an astonishing market lead with no real proprietary challengers in the in-memory space. The only proprietary software challenger in the in-memory space

[21] https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-databases-loved4
[22] https://insights.stackoverflow.com/survey/2020#technology-databases-professional-developers4
[23] Eric Redmond and Jim Wilson, Chapter 8, Seven Databases in Seven Weeks

is Redis itself with proprietary extensions, readily replaced with open source software, that seek to lure customers into a proprietary style single-vendor relationship.

Summary

Miller in 1968 and Nielsen in 1993 agree that human perceptual speed has a threshold where under 100ms is largely seen as "instantaneous" and that faster is universally better. Research for the last 50 years and a large number of cross-domain case studies over the last 20 years, involving billions of real world data points, show that latency cannot be ignored and excess latency will negatively impact revenue, engagement, and productivity.

We contend that it is vital to measure the speed of user experiences and to have a data layer that scales to demand without slowing these down. Additionally, there is always a place in a company's data layer for a highly adaptable in-memory system with versatile and easy to use data structures so that you can easily deploy common in-memory architecture patterns, like intelligent caching and adaptive traffic shaping, on your data flows.

There are many systems which can analyze your logs after the fact, but only a system that's operating at much higher speed than the rest of your infrastructure can allow real-time applications, adaptations and experiences.

In-memory databases can even handle some stream processing although we argue that this should not be considered a replacement for enterprise grade stream processing, where other products like Apache Kafka are more robust.

Finally, we believe Redis's high popularity with developers in use, and its place as the most popular database people are considering to use again, is a warranted position. Redis is open source, robust, has an elegant interface and provides a superset of the features provided by competitors who offer at best similar performance in a far narrower set of use cases. Finally, Redis has great disk persistence and clustering capabilities which make it able to be scaled and deployed in high availability scenarios where warm restarts are required to protect core infrastructure.

About Instaclustr

Instaclustr helps organizations deliver applications at scale through its managed platform for open source technologies such as **Apache Cassandra®**, **Apache Kafka®**, **Apache Spark™**, **Redis™**, **OpenSearch®**, **PostgreSQL®**, and **Cadence®**.

Instaclustr combines a complete data infrastructure environment with hands-on technology expertise to ensure ongoing performance and optimization. By removing the infrastructure complexity, we enable companies to focus internal development and operational resources on building cutting edge customer-facing applications at lower cost. Instaclustr customers include some of the largest and most innovative Fortune 500 companies.