Couchbase

WHITEPAPER

Why NoSQL Databases for Gaming?

Why successful projects rely on NoSQL database applications

Contents

PREFACE

CUSTOMER SUCCESS STORIES

Social gaming and online sports betting are competitive environments. Games must be able to handle large volumes of unpredictable traffic while simultaneously promising zero downtime. For these companies, user retention is no longer just desirable, it's critical. Gamers expect to play whenever they want for as long as they want, and have a low tolerance for delays or lag. At the heart of any successful game is a database that maintains 100% uptime, scales in real time to handle millions of users or more, and provides users with a responsive and personalized experience across all of their devices. This is why gaming and betting companies choose Couchbase as they move away from monolithic solutions to microservice-based architectures, focusing on building engaging, responsive, and scalable applications.

The most important aspect of any game is its player experience, and with rising expectations and the need for instant gratification, guaranteeing consistent, low-latency data access is critical. Gaming companies need a 360-degree view of their customers to provide them with real-time, personalized data for achievements, stats, and leaderboards. User profile data is stored to support multiple services and enable users to play games across different platforms and devices.

Player experience begins with availability. With users worldwide, a game not only has to be available 24x7, 365 days a year, but it also has to be available everywhere —with no single point of failure. **Nexon**, a global leader in virtual world games and massively multiplayer online role-playing games (MMORPG), uses Couchbase Capella™ Database-as-a-Service (DBaaS) for greater developer agility. Capella's high availability and distributed memory-first architecture deliver a consistent performance experience for players as game adoption grows.

Scalability is also a key consideration for gaming companies. In the present environment, social games have the potential to go viral at any time. Meeting this spike in demand can be overwhelming for relational databases. Couchbase supports these ever-increasing workloads and scales easily without disruption. **Jam City**, a leader in mobile entertainment, provides unique and deeply engaging games that appeal to broad global audiences. Its wildly popular puzzle game Cookie Jam won Facebook's Game of the Year after scaling to meet the demand of 35 million users globally in under 8 months. Jam City and Couchbase teamed up in preparation for the huge spike in social and mobile hits once Cookie Jam had begun gaining traction, successfully avoiding downtime with Couchbase performance at scale and flexibility in rebalancing and failover through cross data center replication (XDCR).

Betfair, one of the world's largest online sports betting providers, is another customer experiencing improved performance, flexibility, and scalability of its NoSQL deployments with Couchbase. Betfair uses numerous applications to serve over 4 million funded accounts across 140 countries. To eliminate unnecessary complexity, the company moved many of their applications from Oracle to Couchbase. The platform provided easy integration and the continuous delivery needed to process over 30,000 bets per minute.

WHAT IS NOSQL?

NoSQL is a modern database management system that stores information in JSON documents instead of the columns and rows used by relational databases. It can deliver data for applications in ways that make development easier and more robust.

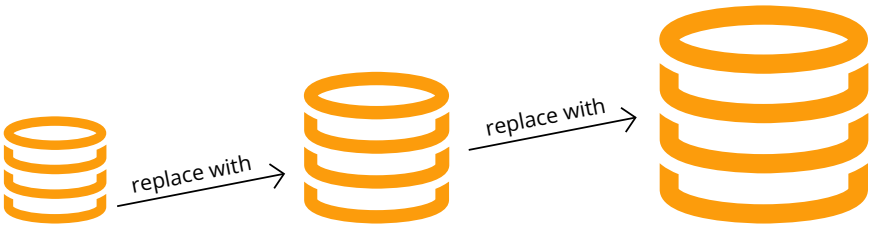
NoSQL databases are built from the ground up to be flexible, scalable, and capable of rapidly responding to the data management demands of modern businesses.

This paper introduces the modern challenges that NoSQL databases address and shows when to choose NoSQL over relational and why.

Customer experience drives enterprise to NoSQL solutions

Customer experience is the most important competitive differentiator and businesses are working to meet these new expectations with services that are on demand, real time, resilient, and responsive.

Bringing together all of these new systems requires flexibility, performance, and scalability and can consolidate multiple types of systems to reduce database sprawl.



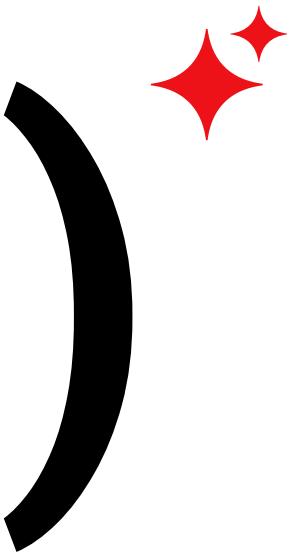
Relational vs. NoSQL: What's the difference?

Relational databases were born in the era of mainframes and back-office business applications—long before the internet, the cloud, big data, mobile, etc. In fact, the first commercial implementation was released by Oracle in 1979. These databases were engineered to run on a single server—the bigger, the better. The only way to increase the capacity of these databases was to upgrade the server—processors, memory, and storage—to scale up as Moore's Law allowed.

NoSQL databases emerged as a result of the exponential growth of the internet and the rise of web applications. Google Bigtable research was released in 2006, as well as an Amazon Dynamo research paper in 2007. Efficient distributed key-value (KV) engines were essential to this evolutionary step and have propelled the technology much further.

New databases were engineered to meet the next generation of enterprise requirements, which companies like Couchbase have taken even further to meet needs going into the future—the need to **develop with agility** and to **operate at any scale**.

Agility means providing flexible schemas, APIs, robust SQL-based querying, text search, analytics, and more. Scalability allows data to grow without sacrificing performance and stability.



Supporting SQL and NoSQL developers

Traditional relational systems manage tabular data and return it as rows and columns. NoSQL databases can do this without forcing application developers into using a static schema that has to be reworked every time there is a change. Instead, NoSQL databases give developers the flexibility they need to help them excel at their work.

NoSQL systems hold hierarchical JSON data but return it to the application as full or partial JSON data structures, full-text search matches, SQL query results, key-based values, or even big data analytics systems.

This convergence of the best of relational and the best of modern NoSQL simplifies the information architecture of enterprises and helps developers deliver applications more efficiently with familiar concepts and tooling, without needing to learn a dozen different platforms.

Scaling beyond SQL databases

To operate at scale, NoSQL systems approach cluster-based computing with efficient, automatic cluster management to keep data synchronized and flowing at high speeds.

For example, teams are now expected to build enterprise data management infrastructure that includes the following characteristics:

- Support large numbers of concurrent users (tens of thousands, perhaps millions)
- Deliver highly responsive experiences to a globally distributed base of users
- Be always available—no downtime
- Handle semi- and unstructured data
- Rapidly adapt to changing requirements with frequent updates and new features

SQL-based relational databases are unable to meet these requirements efficiently and cost-effectively.

Consider just a few examples of Global 2000 enterprises that are deploying NoSQL for mission-critical applications that have been featured in recent news reports:

- **Tesco**, Europe's No. 1 retailer deploys NoSQL for e-commerce, product catalog, and other applications
- **Ryanair**, the world's busiest airline uses NoSQL to power its mobile app serving over 3 million users
- **Marriott** hosts 30 million documents, accessed at 4,000 transactions per second
- **GE** deploys NoSQL for its Predix platform to help manage the industrial internet
- **Sky / Peacock** deploys NoSQL to offer a seamless viewing experience during peak watching times

TODAY'S TRENDS—TOMORROW'S CHALLENGES

Today's customer experience goals depend on tightly aligned technical integration more than ever before, but it must be able to handle trends going forward or risk becoming outdated.

Some of the high-level technical goals include:

- Consolidated platforms that work together efficiently
- Simplified system architectures that are easy to manage
- Effective data "plumbing" for real-time web applications and low latency
- Act as a service layer that pushes data as close to the customer as possible

Five advantages of NoSQL databases

Ambitious customers with big ideas for how to use data have unleashed a new set of technology requirements for CIOs and technical leaders.

Here are five trends that play into the advantages of NoSQL databases for addressing the challenges of building software.

Trends	Requirements
Customer shift continues online	• Scaling to support thousands or even millions of users • Meeting UX requirements with consistently high performance • Maintaining availability 24 hours a day, 7 days a week
The internet is connecting everything	• Supporting many different applications with different data structures • Ensuring software is “always on” with no excuse for downtime • Supporting continuous streams of data from the real-time web
Big data is getting bigger	• Storing customer-generated semi-structured and unstructured data • Storing different types of data from different sources in the same infrastructure or even the same cluster • Storing data generated by thousands or millions of customers and IoT devices
Applications are moving to the cloud	• Scaling on demand to support more customers, and store more data • Operating fully managed applications on a global scale to support customers worldwide • Minimizing infrastructure and operating costs, achieving a faster time to market
The world has gone mobile	• Creating “offline-first” apps – network connection not required • Synchronizing mobile/edge data with remote databases in the cloud • Supporting multiple mobile platforms with a single backend

The above requirements are extensive and challenge even the best systems to do even more with less. Today’s extreme requirements can be loosely grouped into two categories that impact two different levels of end users:

- Providing agile platforms for application developers to excel
- Supporting scalable system architectures that outperform others

DEVELOP WITH AGILITY

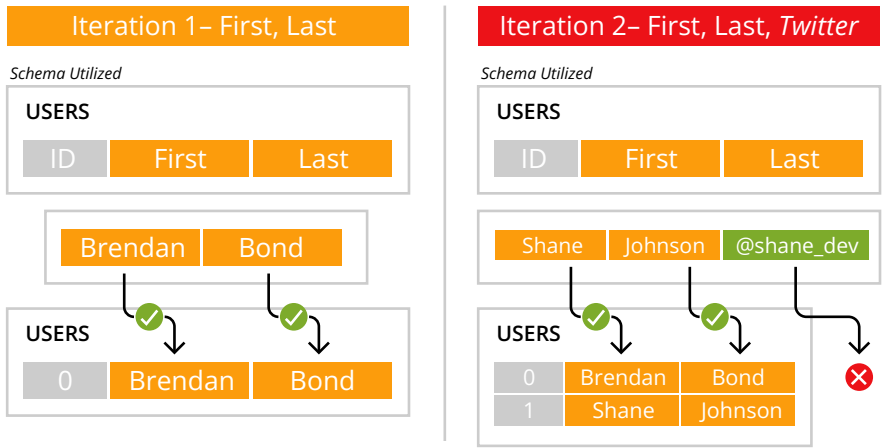
To remain competitive, enterprises must innovate – and now they have to do it faster than ever before. Developers are under extraordinary pressure. Speed is critical, but so is agility, since these applications evolve far more rapidly than legacy applications. Relational databases have a restricted, flat data structure and don't respond well to frequent changes in the data model. This often impedes the needs of modern, agile projects, applications, and business requirements.

Adaptable NoSQL schema requirements

A core principle of agile development is responding to change. When the requirements change, the data model also changes. This is a problem for relational databases because the data model can be cumbersome to change. In order to change the data model, developers have to modify the schema, or in some organizations, formally request a "schema change" from the database administrators. This slows down or stops development, not only because it is a manual, time-consuming process, but because it also impacts other applications and services

FIGURE 1

RDBMS – An explicit schema prevents the addition of new attributes on demand

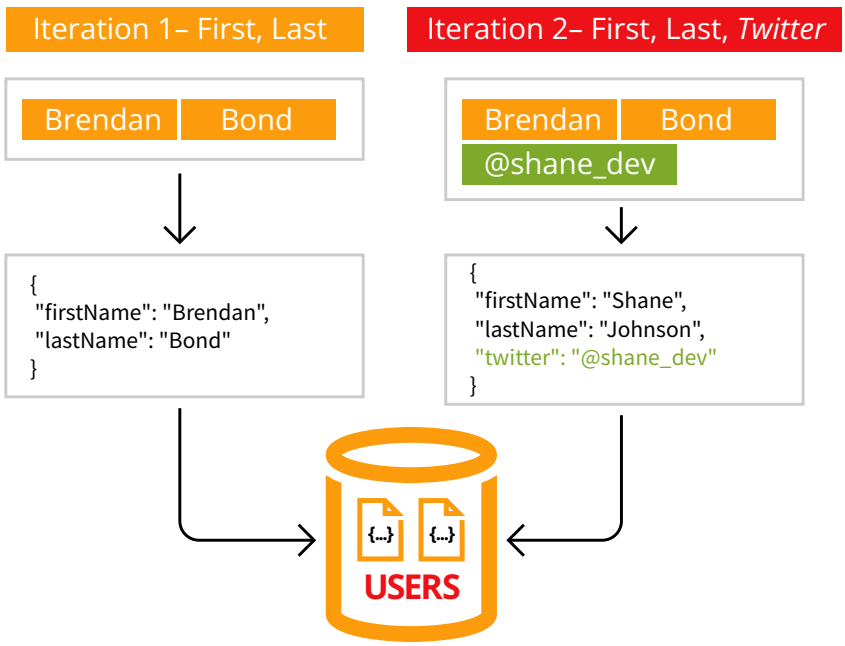


Flexibility for faster development

By comparison, a NoSQL document database fully supports agile development, because it is schema-less and does not statically define how the data must be modeled. Instead, it defers to the applications and services, and thus to the developers as to how data should be modeled. With NoSQL, the data model is defined by the application model. Applications and services model data as objects.

FIGURE 2

JSON – The data model evolves as new attributes are added on demand



Simplicity for easier development

Applications and services model data as objects (e.g., employee profile), multi-valued data as arrays (e.g., roles), and related data as nested objects or arrays (e.g., manager relation). However, relational databases model data as tables of rows and columns—related data as rows within different tables, multi-valued data as rows within the same table. One problem with relational data modeling is that data is read and written by disassembling, or "shredding," (like a paper shredder turns a single document into multiple smaller strips) and reassembling objects. This is the object-relational "impedance mismatch."

The workaround is object-relational mapping (ORM) frameworks, which can be effective in simple scenarios, but can become problematic in more complex situations.

Consider an application for managing resumes. It interacts with resumes as an object of user objects. It contains an array for skills and a collection for positions. However, writing a resume to a relational database requires the application to "shred" the user object.

Storing this resume would require the application to insert six rows into three tables, as illustrated in Figure 3.

However, reading this profile would require the application to read six rows from three tables, as illustrated in Figure 4.

FIGURE 3

RDBMS – Applications "shred" objects into rows of data stored in multiple tables

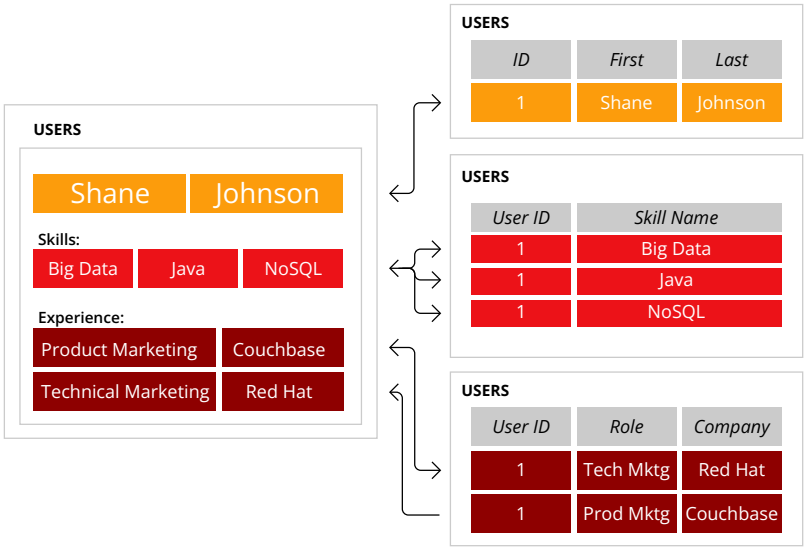


FIGURE 4

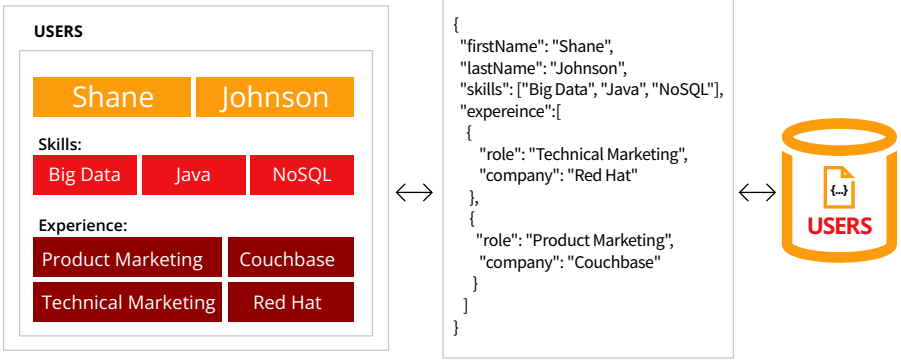
RDBMS – An explicit schema prevents the addition of new attribues on demand

Shane	Johnson	Big Data	Product Mktg	Couchbase
Shane	Johnson	Big Data	Technical Mktg	Red Hat
Shane	Johnson	Java	Product Mktg	Couchbase
Shane	Johnson	Java	Technical Mktg	Red Hat
Shane	Johnson	NoSQL	Product Mktg	Couchbase
Shane	Johnson	NoSQL	Technical Mktg	Red Hat

In contrast, a document-oriented NoSQL database reads and writes data formatted in JSON—which is the de facto standard for consuming and producing data for web, mobile, and IoT applications. It not only can eliminate the object-relational impedance mismatch, it can eliminate the overhead of ORM frameworks and simplifies application development because objects are read and written without "shredding" them—i.e., a single object can be read or written as a single document, as illustrated in Figure 5.

FIGURE 5

RDBMS – Applications can store objects with nested data as single documents



Grouping documents to ease access

Unlike using predefined sets of schemas to differentiate tables from one another, NoSQL databases have a concept, such as buckets, that serve as a general holding area for all documents. A database can have many logical, named, buckets for various purposes. The name is provided while connecting or requesting data and allows applications to have their own area in the system.

Within those buckets are additional hierarchical logical groupings that can be restricted to particular users or roles. These are called collections and/or scopes, allowing subsets of documents in a bucket to be named. Because this flexibility helps segregate data of one user or application from another, the developer does not have to build their own security and reliability code, but can instead let the underlying database do it.

Querying using SQL

Application developers that are used to querying with SQL can continue to use the same language in NoSQL platforms but operate against the JSON data that is stored. For example, Couchbase provides a SQL-based query standard known as SQL++ (sometimes known as N1QL) that returns results in JSON with sets of rows and subdocument components where appropriate. This is in contrast to the vast majority of other NoSQL databases (like MongoDB™) that don't use SQL and require developers to climb a new language learning curve.

Standard statements are supported including SELECT … FROM … WHERE syntax. SQL++ also supports aggregation, sorting, and joins (GROUP BY … SORT BY … LEFT OUTER/INNER JOIN). Querying collections, scopes, and even nested arrays is supported. Query performance can be improved with composite, partial, covering indexes, and more.

There are minimal changes for SQL experts to move to SQL++ where desired, with many basic queries working out of the box.

SQL	SQL++
SELECT p.FirstName + ' ' + p.LastName AS Name, d.City FROM AdventureWorks2016.Person.Person AS p INNER JOIN AdventureWorks2016.HumanResources.Employee e ON p.BusinessEntityID = e.BusinessEntityID INNER JOIN (SELECT bea.BusinessEntityID, a.City FROM AdventureWorks2016.Person.Address AS a INNER JOIN AdventureWorks2016.Person.BusinessEntityAddress AS bea ON a.AddressID = bea.AddressID) AS d ON p.BusinessEntityID = d.BusinessEntityID ORDER BY p.LastName, p.FirstName;	SELECT p.FirstName || ' ' || p.LastName AS Name, d.City FROM AdventureWorks2016.Person.Person AS p INNER JOIN AdventureWorks2016.HumanResources.Employee e ON p.BusinessEntityID = e.BusinessEntityID INNER JOIN (SELECT bea.BusinessEntityID, a.City FROM AdventureWorks2016.Person.Address AS a INNER JOIN AdventureWorks2016.Person.BusinessEntityAddress AS bea ON a.AddressID = bea.AddressID) AS d ON p.BusinessEntityID = d.BusinessEntityID ORDER BY p.LastName, p.FirstName;

What about ACID transactions in NoSQL?

NoSQL databases also operate as operational systems with large numbers of transactions. When you flatten out a business entity into multiple separate tables, you require a transaction for almost every update. With NoSQL databases, you don't need to flatten out the entity, but can usually contain it in a single document. Updates to a single document are atomic and don't require a transaction.

However, there may be updates that span multiple documents and require a check to ensure "all or nothing" of the transaction occurs. For instance, a transfer of credits from one user's account to another.

This is why NoSQL databases like Couchbase support transactions.

Java example:

```
Transactions transactions = Transactions.create(cluster,
      TransactionConfigBuilder.create()
.durabilityLevel(TransactionDurabilityLevel.PERSIST_TO_MAJORITY)
                  .logOnFailure(true, Event.Severity.WARN)
            .build());

TransactionResult result = transactions.run((ctx) -> {
      // Inserting a doc:
      ctx.insert(collection, "doc-a", JsonObject.create());

// Getting documents:
// Use ctx.getOptional if the document may or may not exist
Optional<TransactionGetResult> docOpt =
                  ctx.getOptional(collection, "doc-a");

      // Use ctx.get if the document should exist, and the transaction
      // will fail if it does not
      TransactionGetResult docA = ctx.get(collection, "doc-a");

      // Replacing a doc:
      TransactionGetResult docB = ctx.get(collection, "doc-b");
      JsonObject content = docB.contentAs(JsonObject.class);
      content.put("transactions", "are awesome");
      ctx.replace(docB, content);
      // Removing a doc:

      TransactionGetResult docC = ctx.get(collection, "doc-c");
      ctx.remove(docC);

      ctx.commit();
});
```

The combination of transactions and SQL greatly expand the number of use cases where a NoSQL database can be considered. In the past, the inability to join or handle transactional operations meant that NoSQL databases were only chosen for the highest volume and scale use cases. But the option of using SQL and transactions means that NoSQL databases can also be chosen for traditional RDBMS cases that need more flexibility and power.

Additionally, by selecting a transactional NoSQL database, many traditionally complex applications can be simplified because there is no need for an ORM tool.

One data source—multiple access methods

NoSQL databases operate as a primary content store, meaning you enter the data in one application but can access it multiple ways depending on the use case. For example, developers can use direct API calls to access a specific document using a key or through a SQL query that returns multiple rows of data in a JSON response. This is known as "multi-model."

Other access methods are available depending on the database, including full-text search systems that allow natural language search requests. Requests can be made for full or partial "fuzzy" matches, geographic ranges, or wildcard searches. The response includes a JSON document with lists of matching document IDs, contextual information, and a relevancy score.

Full-text search systems are often separate from a database but NoSQL databases like Couchbase include them as part of the underlying system, allowing managers to simplify the overall architecture.

Big data analytics are possible as well, using complimentary subsystems that process larger volumes of historical data. Using advanced indexing and query capabilities, also based on SQL++, more advanced analytics can be done in the same database without needing a separate, external OLAP system.

Because the data is stored and indexed all within the one database product, it allows developers to connect to one system and pass through the relevant requests.

OPERATE AT ANY SCALE

Databases that support web, mobile, and IoT applications must be able to operate at any scale. While it is possible to scale a relational database like Oracle (using, for example, Oracle RAC), doing so is typically complex, expensive, and not fully reliable. With Oracle, for example, scaling out using RAC technology requires numerous components and creates a single point of failure that jeopardizes availability.

By comparison, a NoSQL distributed database—designed with a scale-out architecture and no single point of failure—provides compelling operational advantages.

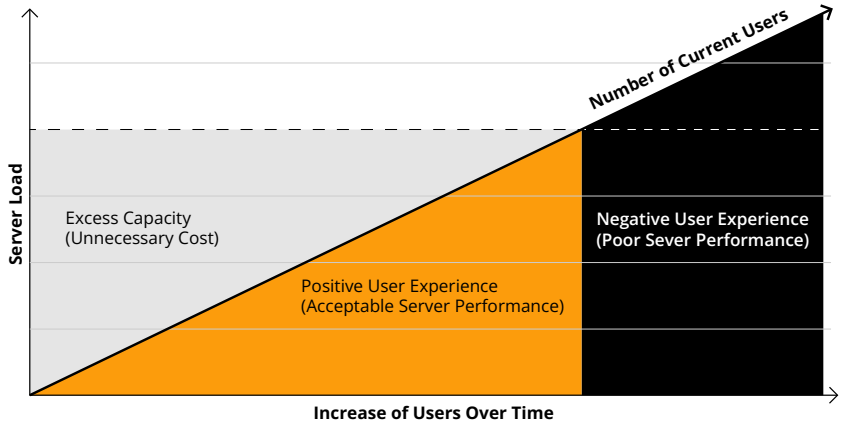
Elasticity for performance at scale

Applications and services have to support an ever-increasing amount of users and data—hundreds to thousands to millions of users, and gigabytes to terabytes of operational data. At the same time, they have to efficiently scale to maintain performance.

The database has to be able to scale reads, writes, and storage. This is a problem for relational databases that are limited to scaling up—i.e., only being able to scale by adding more processors, memory, and storage to a single physical server (sometimes known as "vertical scaling"). As a result, the ability to scale efficiently, and on demand, is a challenge. It becomes increasingly expensive, because enterprises have to purchase bigger and bigger servers to accommodate more users and more data. In addition, it can result in downtime if the database has to be taken offline to perform hardware upgrades.

FIGURE 6

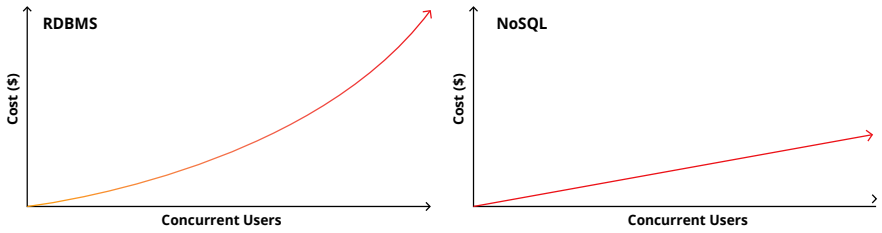
RDBMS – The server is too big or too small, leading to unnecessary costs or poor performance



A distributed NoSQL database runs on commodity hardware to scale out—i.e., add more resources simply by adding more servers to a cluster (sometimes known as "horizontal scaling"). The ability to scale out enables enterprises to scale more efficiently by (a) deploying no more hardware than is required to meet the current load; (b) applying less expensive hardware and/or cloud infrastructure; and (c) scaling on demand and without downtime.

FIGURE 7

RDBMS – Add commodity servers on demand so the hardware resources match the application load



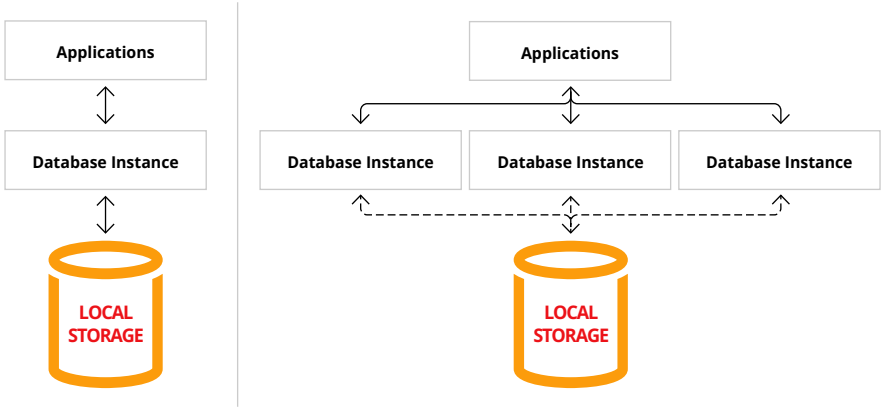
By distributing reads, writes, and storage across a cluster of nodes, NoSQL databases are able to operate at any scale. Additionally, they are designed to be easy to configure, install, and manage both small and large clusters.

Availability for always-on, global deployment

As more and more customer engagements take place online via web and mobile apps, availability becomes a major concern. These mission-critical applications have to be available 24 hours a day, 7 days a week—no exceptions. Delivering 24x7 availability is a challenge for relational databases that are deployed to a single physical server or that rely on clustering with shared storage. If the single server or shared storage fails, the database becomes unavailable, applications stop, and customers disengage.

FIGURE 8

RDBMS – The failure of a server or storage device brings down the entire database



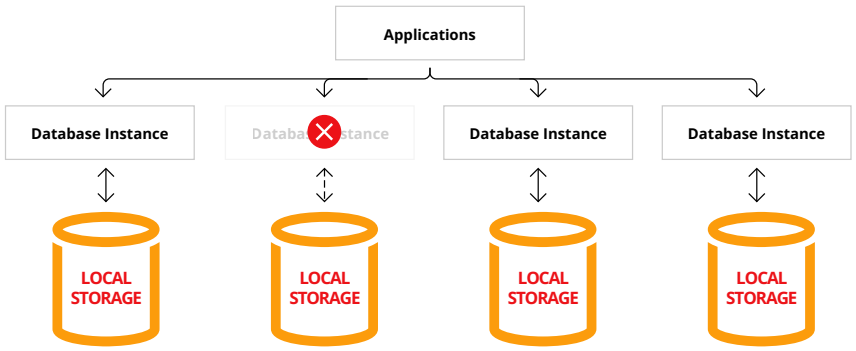
In contrast to relational technology, a distributed, NoSQL database partitions and distributes data to multiple database instances with no shared resources. Couchbase goes a step further and does this automatically.

In addition, the data can be replicated to one or more instances for high availability (intercluster replication) and in different geographic locations. While relational databases like Oracle require separate software for replication, for example, Oracle Active Data Guard, NoSQL databases do not—it's built in and it's automatic. Couchbase's cross data center replication (XDCR) feature also provides this automatically.

In addition, automatic failover ensures that if a node fails, the database can continue to perform reads and writes by sending the requests to a different node. Again, Couchbase will perform this failover recovery and rebalancing automatically.

FIGURE 9

NoSQL – If an instance fails, the application can send requests to a different one

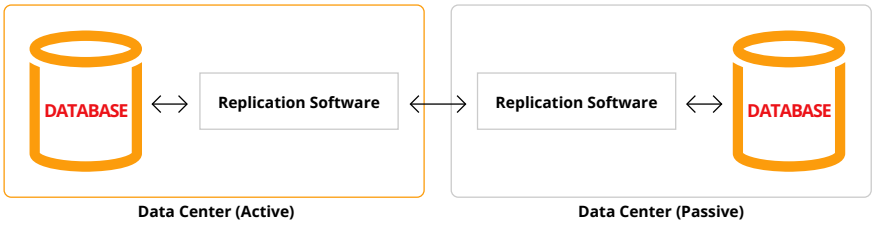


Customer behavior often requires organizations to support multiple physical, online, and mobile channels in multiple regions and often multiple countries. While deploying a database to multiple data centers increases availability and helps with disaster recovery, it also has the benefit of increasing performance too. All reads and writes can be executed on the nearest data center, thereby reducing latency.

Ensuring global availability is difficult for relational databases where separate add-ons are required—which increase complexity—or where replication between multiple data centers can only be used for failover, because only one data center is active at a time. Oracle, for example, requires Oracle GoldenGate. When replicating between data centers, applications built on relational databases can experience performance degradation or find that the data centers are severely out of sync.

FIGURE 10

RDBMS – Requires separate software to replicate data to other data centers

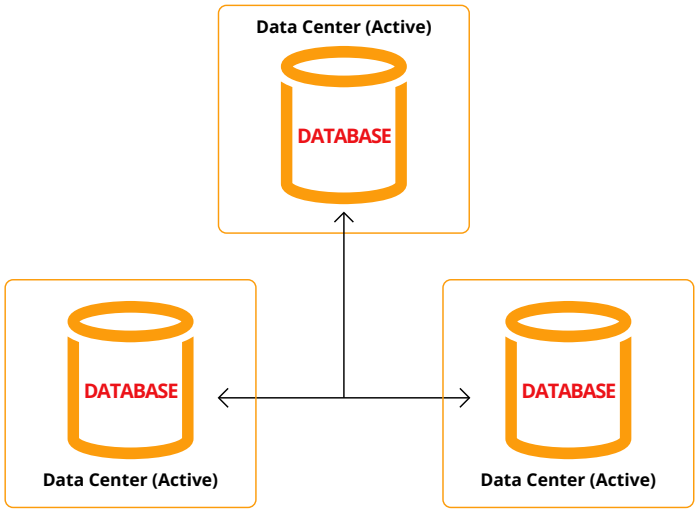


A distributed, NoSQL database includes built-in replication between data centers—no separate software is required. In addition, some include bidirectional replication enabling full active-active deployments to multiple data centers. This enables the database to be deployed in multiple countries or regions while providing local data access to local applications and their users.

Deploying to multiple data centers not only improves performance, but enables immediate failover via hardware routers. Applications don't have to wait for the database to discover the failure and perform its own failover.

FIGURE 11

RDBMS – Replication between data centers is fully built-in and can be bidirectional



NOSQL IS A BETTER FIT FOR LARGE-SCALE REQUIREMENTS

As enterprises shift to cloud, mobile, social media, and big data technologies, developers, architects, and operations teams have to build and maintain web, mobile, and IoT applications faster, and at a greater scale. NoSQL is increasingly the preferred database technology to power today's web, mobile, and IoT applications.

Hundreds of Global 2000 enterprises, along with tens of thousands of smaller businesses and startups, have adopted NoSQL. For many, the use of NoSQL started with caching, proof of concept, or a small application, then expanded to targeted mission-critical applications, and can become the foundation for all application development. Today, the Couchbase NoSQL database serves thousands of these types of customers.

With NoSQL, enterprises are better able to both develop with agility and operate at any scale—and to deliver the performance and availability required to meet the demands of businesses.

Modern customer experiences need a flexible database platform that can power applications spanning from cloud to edge and everything in between. Couchbase's mission is to simplify how developers and architects develop, deploy and consume modern applications wherever they are. We have reimagined the database with our fast, flexible and affordable cloud database platform Capella, allowing organizations to quickly build applications that deliver premium experiences to their customers—all with best-in-class price performance. More than 30% of the Fortune 100 trust Couchbase to power their modern applications.

For more information, visit www.couchbase.com and follow us on Twitter.