Couchbase

Why NoSQL?

The Shift to the Digital Economy is driving NoSQL

The business world is undergoing massive change as industry after industry shifts to the Digital Economy. It's an economy powered by the Internet and other 21st century technologies – the cloud, mobile, social media, and big data. At the heart of every Digital Economy business are its web, mobile, and Internet of Things (IoT) applications: They're the primary way companies interact with customers today, and how companies run more and more of their business. The experiences that companies deliver via those apps largely determine how satisfied — and how loyal — customers will be.

How are these applications different from legacy enterprise applications like ERP, HR and financial accounting? Today's web, mobile, and IoT applications share one or more (if not all) of the following characteristics. They need to:

- 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

Building and running these web, mobile, and IoT applications has created a new set of technology requirements. The new enterprise technology architecture needs to be far more agile than ever before, and requires an approach to real time data management that can accommodate unprecedented levels of scale, speed, and data variability. Relational databases are unable to meet these new requirements, and enterprises are therefore turning to NoSQL database technology.

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

- **Tesco**, Europe's #1 retailer, deploys NoSQL for ecommerce, product catalog, and other applications
- **Marriott** is deploying NoSQL for its reservation system that books $38 billion annually
- **Gannett**, the #1 U.S. newspaper publisher, uses NoSQL for its proprietary content management system, Presto
- **GE** is deploying NoSQL for its Predix platform to help manage the Industrial Internet

NoSQL was pioneered a decade ago by leading Internet companies – including Google, Amazon, Facebook, and LinkedIn – to overcome limitations of relational databases like Oracle, and MySQL for modern web applications. Once these early pioneers proved the advantages and efficacy of NoSQL, adoption by enterprises started to unfold in three overlapping phases:

- **Grassroots Experimentation:** In phase I (which started around 2010), enterprise developers began experimenting with NoSQL on side projects and non-mission-critical applications. Their key requirement was flexibility to support agile development of proofs of concept and small applications.

- **Mission Critical Deployments:** In phase II (which began around 2013), enterprises started to adopt NoSQL for mission critical applications. In this phase, the key requirements are performance, scalability, and availability to develop and / or migrate targeted services.

- **Broad Replatforming:** In phase III (which is just starting in late 2015), both developers and enterprises require a general purpose database for broad enterprise adoption to re-platform all mission-critical applications and services for the Digital Economy. In this phase, NoSQL database requirements include flexibility, performance, scalability, and availability as well as a comprehensive query language and powerful indexing.

Five Trends Create New Technical Challenges that NoSQL Addresses

At a more granular level, enterprises are adopting NoSQL in order to address a new set of technical challenges and requirements that are the result of five major trends:

1. More customers are going online

More and more customers are doing more and more online, whether at home, at work, or on the go. They're paying bills, making reservations, shopping for groceries – the list goes on and on – and they're doing it online, not at the local branch, travel agency, or grocery store.

Technical challenges from the shift to online include:

- Scaling to support thousands if not millions of users
- Meeting user experience requirements with consistent high performance
- Maintaining availability 24 hours a day, 7 days a week

2. The Internet is connecting everything

It's not just people going online and getting connected. Today, more and more "things" are connected to the Internet – the "Internet of Things" – and they're both consuming and producing data. They're consumer things – appliances, watches, automobiles, and more. They're industrial things – airplane engines, wind turbines, oil rigs and pipelines, and more. They're big. They're small. They're local. They're remote.

Technical challenges from the Internet of Things include:

- Supporting many different (and often new) things, with different data structures
- Supporting hardware and software updates that change the structure of data
- Supporting continuous streams of real-time data

3. Big Data is getting bigger

Today, enterprises are leveraging big data to improve the efficiency and effectiveness of operations, applications, services, and more as a result of customers doing more and more online and machines generating more and more data.

Technical challenges from Big Data include:

- Storing semi-structured and unstructured data generated by customers
- Storing different types of data, potentially from different sources, together
- Storing data generated by thousands, if not millions, of customers and things

4. Applications are moving to the cloud

The ability to scale on demand and to operate at scale requires elastic, if not global, infrastructure, as enterprises shift to distributed software and commodity hardware for cost, flexibility, and performance advantages. This has led to adoption of cloud infrastructure, whether public, private, hybrid, virtualized, containerized, or bare-metal.

Technical challenges from moving to the cloud include:

- Scaling on demand to support more customers and/or store more data
- Operating applications on a global scale to serve customers worldwide
- Minimizing infrastructure costs and achieving a faster time to market

5. The world has gone mobile

More and more customers are interacting via mobile platforms – whether it's a smartphone, a tablet, a watch, or something else. And enterprises are not only extending applications and services to mobile platforms, they're evaluating "mobile first" and now, "offline first" solutions.

Technical challenges from the move to mobile include:

- Creating "offline first" apps that don't require a network connection
- Synchronizing mobile data with remote databases in the cloud
- Supporting multiple mobile platforms with a single backend

Why relational databases fall short

Enterprises have relied on relational databases like Oracle, SQL Server, DB2, MySQL and others for decades. So why does relational technology fail to meet the requirements of today's web, mobile, and IoT applications?

Relational databases were born in the era of mainframes and business applications – long before the Internet, the cloud, big data, mobile and now, the Digital Economy. 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 servers – processors, memory, and storage – to scale up.

NoSQL databases emerged as a result of the exponential growth of the Internet and the rise of web applications. Google released the BigTable research in 2006, and Amazon released the Dynamo research paper in 2007. These databases were engineered to meet a new generation of enterprise requirements: The need to **develop with agility** and to **operate at any scale**.

Develop with Agility

To remain competitive in the Digital Economy, enterprises must innovate – and now they have to do it faster than ever before. As this innovation centers on the development of modern web, mobile, and IoT applications, developers have to deliver applications and services faster than ever before. Speed is critical, but so is agility, since these applications evolve far more rapidly than legacy applications like ERP. Relational databases are a major roadblock to agility, since they do not support agile development very well due to their fixed data model.

Flexibility for Faster Development

A core principle of agile development is adapting to evolving application requirements: when the requirements change, the data model also changes. This is a problem for relational databases because the data model is fixed and defined by a static schema. So in order to change the data model, developers have to modify the schema, or worse, 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 it also impacts other applications and services.

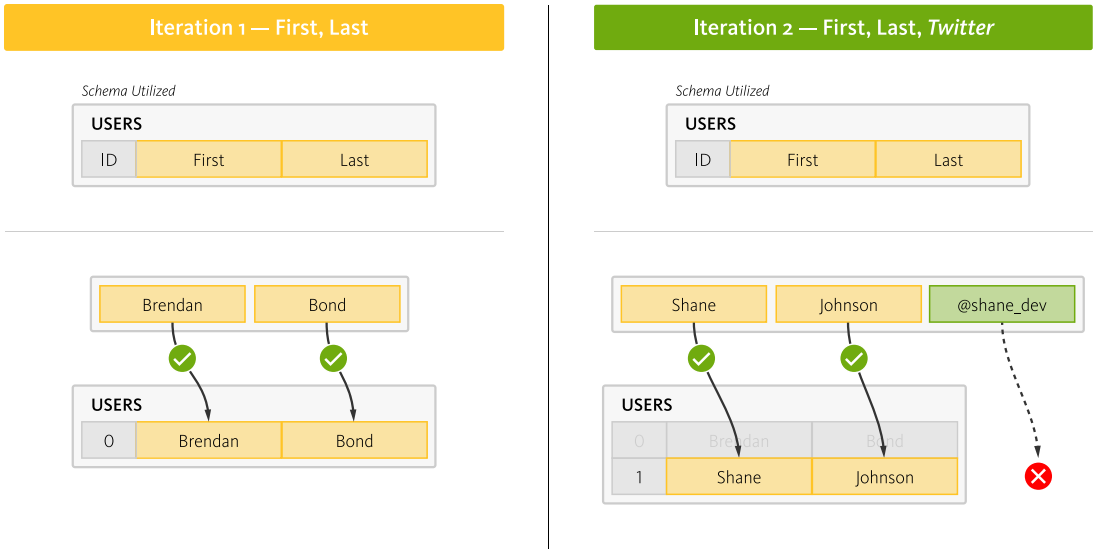

Figure 1: *RDBMS – An explicit schema prevents the addition of new attributes on demand*

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.

With a document database, applications and services simply read and write documents. They write data by serializing an object to a JSON document, and read data by deserializing a JSON document to an object. As a JSON document is self-describing, an external schema is unnecessary. So by modifying an object to add or remove attributes, a developer can change the data model without having to change the database. This translates into a huge boost in developer productivity and agility.

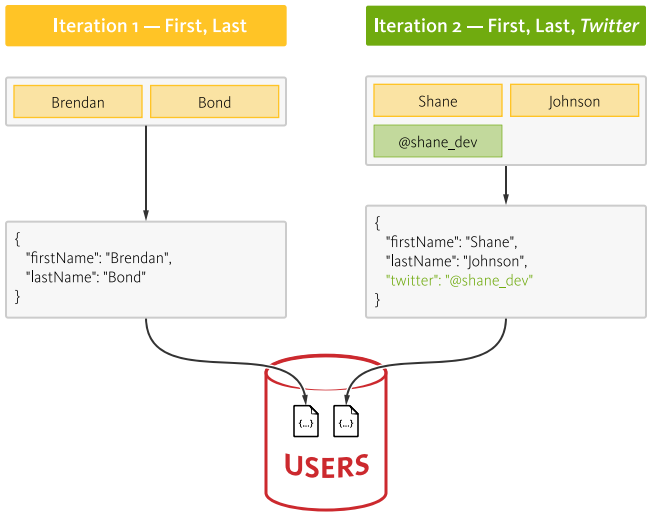

Figure 2: *JSON – The data model evolves as new attributes are added on demand.*

Simplicity for Easier Development

Another advantage of a document database that enables faster innovation is the ability for applications to directly read documents: There's no need for an object-relational mapping layer between the application model and the data.

Applications and services model data as objects (e.g. employee), multi-valued data as collections (e.g., roles), and related data as nested objects or collections (e.g. manager). 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. The problem with relational databases is that data is read and written by disassembling, or "shredding," and reassembling objects. This is the object-relational "impedance mismatch." The workaround is object-relational mapping frameworks, which are inefficient at best, problematic at worst.

As an example, consider an application for managing resumes. It interacts with resumes as an object, the user object. 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:

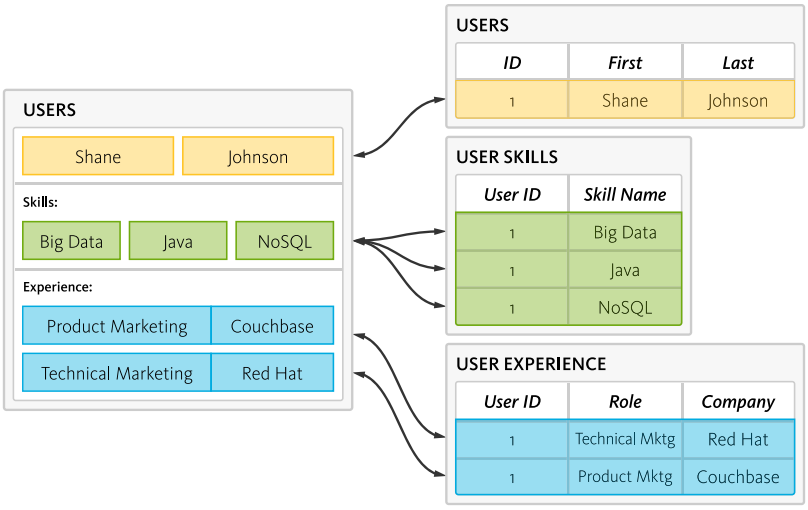

Figure 3: *RDBMS – Applications "shred" objects into rows of data stored in multiple tables.*

Reading this profile would require the application to read six rows from three tables:

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

Figure 4: *RMDBS – Queries return duplicate data, applications have to filter it out.*

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 eliminates the object-relational impedance mismatch, it eliminates 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:

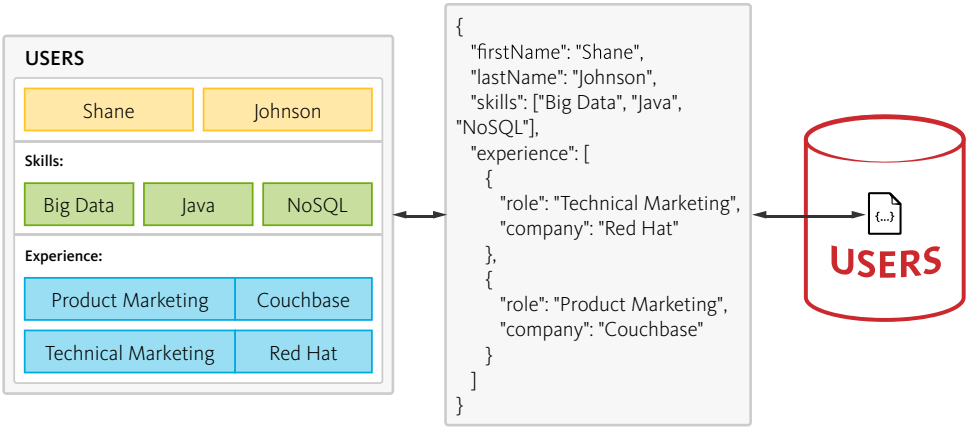

Figure 5: *JSON – Applications can store objects with nested data as single documents.*

What about querying and SQL?

It's important to remember that "NoSQL" stands for "not only SQL." SQL is a mature query technology used by millions of developers. Virtually every programming language and framework, as well as nearly all BI and reporting tools, support SQL. So it's important that developers should be able to leverage their SQL skills and tools when working with a NoSQL database.

Couchbase Server 4.0 introduced N1QL, a powerful query language that extends SQL to JSON, enabling developers to leverage both the power of SQL and the flexibility of JSON. It not only supports standard SELECT / FROM / WHERE statements, it also supports aggregation (GROUP BY), sorting (SORT BY), joins (LEFT OUTER / INNER), as well as querying nested arrays and collections. In addition, query performance can be improved with composite, partial, covering indexes, and more.

SQL

```
SELECT      breweries.name AS brewery,
            count(*) AS cnt
FROM        beers
INNER JOIN breweries
ON          beer.brewery_id = breweries.id
WHERE       beers.type = "beer" AND
            breweries.type = "brewery" AND
            beers.style = "American-Style Imperial Stout"
GROUP BY    breweries.name
HAVING      count(*) > 2
ORDER BY    cnt DESC;
```

N1QL

```
SELECT      breweries.name AS brewery,
            count(*) AS cnt
FROM        `beer-sample` beers
INNER JOIN `beer-sample` breweries
ON KEYS     beers.brewery_id
WHERE       beers.type = "beer" AND
            breweries.type = "brewery" AND
            beers.style = "American-Style Imperial Stout"
GROUP BY    breweries.name
HAVING      count(*) > 2
ORDER BY    cnt DESC;
```

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 number 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 scale to maintain performance, and they have to do it efficiently.

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., adding more processors, memory, and storage to a single physical server. 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.

This leads to under- and over-provisioning. If the server turns about to be too big, the excess capacity is an unnecessary cost. If it turns out to be too small, degraded performance results in a poor user experience.

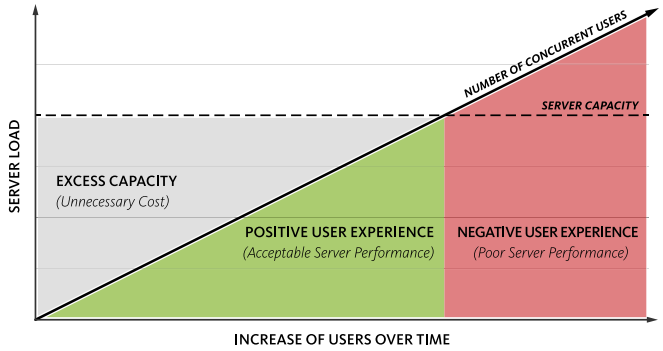

Figure 6: *RDBMS – The server is too big or too small, leading to unnecessary costs or poor performance.*

NOTE:

Alternatives such as manual sharding require custom application logic, introducing unnecessary complexity, and while some relational databases can be deployed as a cluster, they are still limited to scaling up. Oracle RAC, for example, relies on shared storage (e.g. SAN). While adding nodes will scale reads to a point, scaling writes and storage will ultimately require bigger hardware. In addition, the shared storage is not only a performance bottleneck, it's a single point of failure. Further, Oracle RAC is complex, requiring Oracle Grid Infrastructure / Clusterware and Oracle Automatic Storage Management, and expensive – requiring high-end interconnect for storage as well as high-end interconnect for Oracle Cache Fusion.

A distributed NoSQL database, however, leverages commodity hardware to scale out – i.e., add more resources simply by adding more servers. 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) leveraging less expensive hardware and/or cloud infrastructure; and (c) scaling on-demand and without downtime.

In addition to being able to scale effective and efficiently, distributed NoSQL databases are easy to install, configure, and scale. They were engineered to distribute reads, writes, and storage, and they were engineered to operate at any scale – including the management and monitoring of clusters small and large.

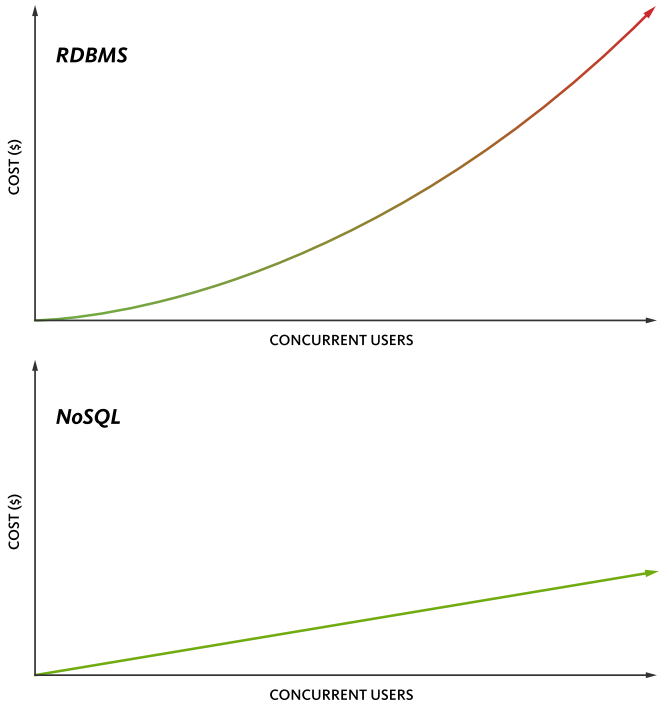

Figure 7: *NoSQL – Add commodity servers on demand so the hardware resources matche the application load.*

Availability for Always-on, Global Deployment

As more and more customer engagements take place online via web and mobile apps, availability becomes a major, if not primary, 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 deployed as a single server and it fails, or as a cluster and the shared storage fails, the database becomes unavailable.

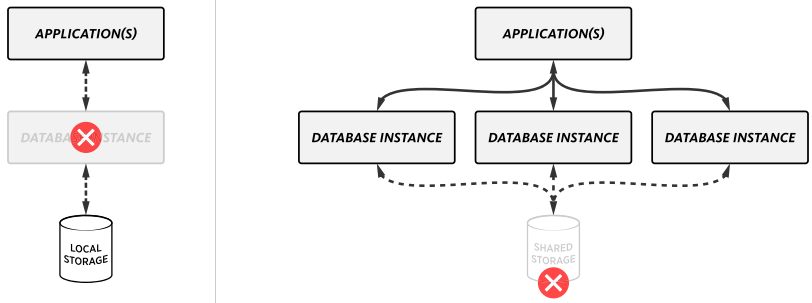

Figure 8: *RMDBS – The failure of a server or storage device brings down the entire database.*

In contrast to relational technology, a distributed, NoSQL database partitions and distributes data to multiple database instances with no shared resources. In addition, the data can be replicated to one or more instances for high availability (intercluster replication). 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. 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.

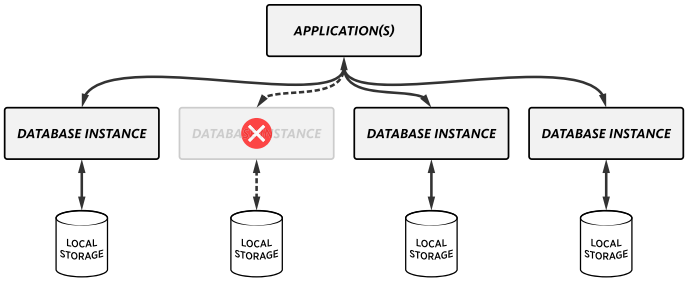

Figure 9: *NoSQL – If an instance fails, the application can send requests to a different one.*

As customer engagements move online, the need to be available in multiple countries and/or regions becomes critical. While deploying a database to multiple data centers increases availability and helps with disaster recovery, it also has the benefit of increasing performance too, because 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 increases 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.

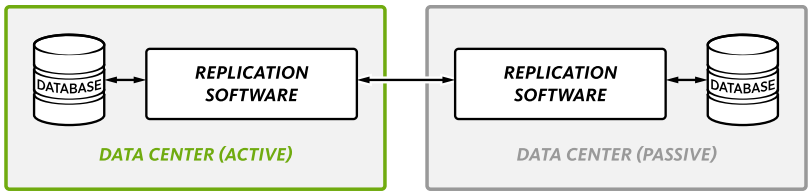

Figure 10: *RDBMS – Requires separate software to replicate data to other data centers.*

A distributed, NoSQL database includes built-in replication between data centers – no separate software is required. In addition, some include both unidirectional and bidirectional replication enabling full active-active deployments to multiple data centers – enabling the database to be deployed in multiple countries and/or regions and to provide local data access to local applications and their users. This not only improves performance, it enables immediate failover via hardware routers – applications don't have to wait for the database to discover the failure and perform its own failover.

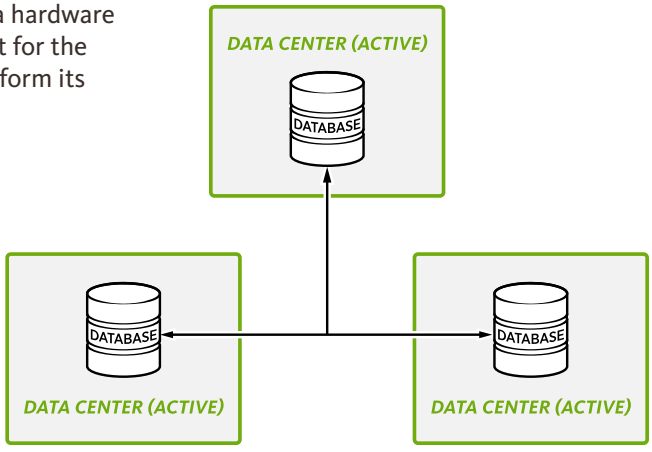

Figure 11: *NoSQL – Replication between data centers is fully built-in, and can be bi-directional.*

NoSQL is a better fit for Digital Economy requirements

As enterprises shift to the Digital Economy – enabled by cloud, mobile, social media, and big data technologies – developers and operations teams have to build and maintain web, mobile, and Internet of Things (IoT) applications faster and faster, and at 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 smaller businesses and start-ups, have adopted NoSQL. For many, the use of NoSQL started with a cache, proof of concept or a small application, then expanded to targeted mission-critical applications, and is now the foundation for all application development.

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 Digital Economy businesses.

Next Steps — Additional Whitepapers

Moving from Relational to NoSQL: How to Get Started

This white paper will help you introduce NoSQL into your infrastructure by highlighting lessons learned from enterprises that have successfully adopted NoSQL. We explore key considerations and strategies for transitioning from a relational database to a NoSQL database, in particular, a document database (Couchbase Server), with notes and hints for the transition from Oracle and other relational databases. There will be times where NoSQL databases are not a replacement for, but a complement to existing infrastructure.

We start with recommendations for identifying and selecting the right application. Next, we cover strategies for modeling relational data as documents, how to access them within your application, and how to migrate data from a relational database. Finally, we highlight the basics of operating a NoSQL database in comparison to a relational database, and provide guidance on how to conduct a successful NoSQL proof of concept.

NoSQL Database Evaluation Guide: How Leading NoSQL Databases Compare Across the Eight Core Requirements

This guide defines and details the eight core requirements for an effective NoSQL database. Based on those requirements, the guide articulates how databases do or do not meet those requirements, and points out what to look for and what to avoid. It begins with Data Access, because the key requirement for Phase III applications in the Digital Economy is the ability to query data with an expressive language that enables developers to query any type of data independent of how it is modeled.

About Couchbase

Couchbase delivers the world's highest performing NoSQL distributed database platform. Developers around the world use the Couchbase platform to build enterprise web, mobile, and IoT applications that support massive data volumes in real time. The Couchbase platform includes Couchbase Server, Couchbase Lite - the first mobile NoSQL database, and Couchbase Sync Gateway. Couchbase is designed for global deployments, with configurable cross data center replication to increase data locality and availability. All Couchbase products are open source projects. Couchbase customers include industry leaders like AOL, AT&T, Bally's, Beats Music, BSkyB, Cisco, Comcast, Concur, Disney, eBay, KDDI, Nordstorm, Neiman Marcus, Orbitz, PayPal, Rakuten / Viber, Tencent, Verizon, Wells Fargo, Willis Group, as well as hundreds of other household names. Couchbase investors include Accel Partners, Adams Street Partners, Ignition Partners, Mayfield Fund, North Bridge Venture Partners, and West Summit.

2440 West El Camino Real | Ste 600
Mountain View, California 94040

1-650-417-7500

www.couchbase.com