Couchbase

Why NoSQL?

Three trends disrupting the database status quo

Interactive applications have changed dramatically over the last 15 years. In the late ‘90s, large web companies emerged with dramatic increases in scale on many dimensions:

- The number of concurrent users skyrocketed as applications increasingly became accessible via the web (and later on mobile devices).
- The amount of data collected and processed soared as it became easier and increasingly valuable to capture all kinds of data.
- The amount of unstructured or semi-structured data exploded and its use became integral to the value and richness of applications.

Dealing with these issues was more and more difficult using relational database technology. The key reason is that relational databases are essentially architected to run a single machine and use a rigid, schema-based approach to modeling data.

Google, Amazon, Facebook, and LinkedIn were among the first companies to discover the serious limitations of relational database technology for supporting these new application requirements. Commercial alternatives didn’t exist, so they invented new data management approaches themselves. Their pioneering work generated tremendous interest because a growing number of companies faced similar problems. Open source NoSQL database projects formed to leverage the work of the pioneers, and commercial companies associated with these projects soon followed.

Today, the use of NoSQL technology is rising rapidly among Internet companies and the enterprise. It’s increasingly considered a viable alternative to relational databases, especially as more organizations recognize that operating at scale is more effectively achieved running on clusters of standard, commodity servers, and a schema-less data model is often a better approach for handling the variety and type of data most often captured and processed today.

What's causing the move to NoSQL?

Three interrelated megatrends – Big Data, Big Users, and Cloud Computing – are driving the adoption of NoSQL technology.

Big Users

Not that long ago, 1,000 daily users of an application was a lot and 10,000 was an extreme case. Today, most new applications are hosted in the cloud and available over the Internet, where they must support global users 24 hours a day, 365 days a year. More than 2 billion people are connected to the Internet worldwide – and the amount time they spend online each day is steadily growing – creating an explosion in the number of concurrent users. Today, it's not uncommon for apps to have millions of different users a day.

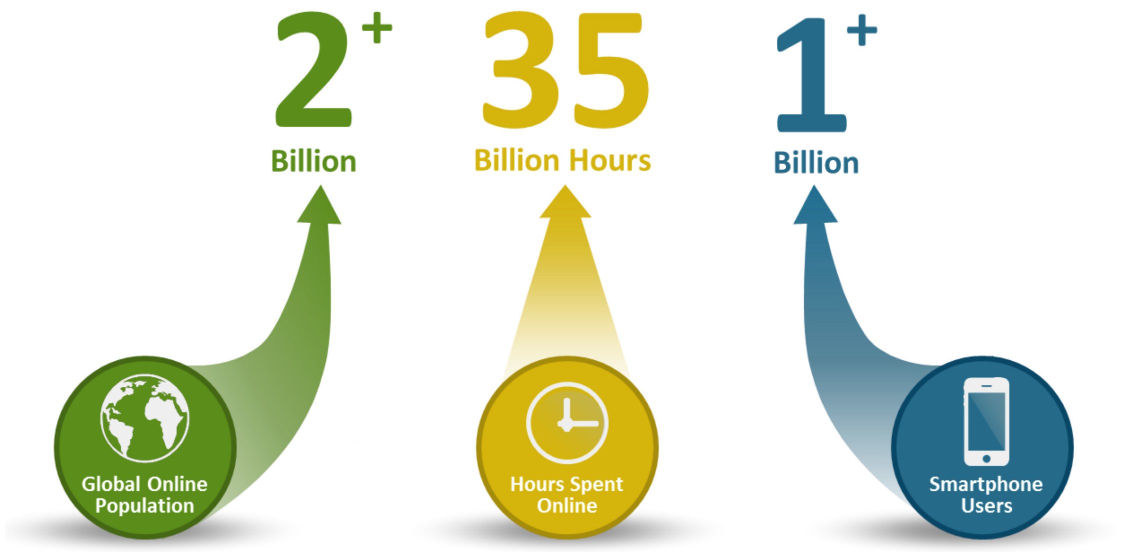


Figure 1. Big Users: With the growth in global internet use, the number of hours spent online, and the increase in smartphone users, it's not uncommon for apps to have millions of users per day.

Supporting large numbers of concurrent users is important but, because application usage requirements are difficult to predict, it's just as important to dynamically support rapidly growing (or shrinking) numbers of concurrent users:

- A newly launched app can go viral, growing from zero to a million users overnight – literally.
- Some users are active frequently, while others use an app a few times, never to return.
- Seasonal swings like those around Christmas or Valentine's Day create spikes for short periods.
- New product releases or promotions can spawn dramatically higher application usage.

The large numbers of users combined with the dynamic nature of usage patterns is driving the need for more easily scalable database technology. With relational technologies, many application developers find it difficult, or even impossible, to get the dynamic scalability and level of scale they need while also maintaining the performance users demand. Many are turning to NoSQL for help.

Big Data

Data is becoming easier to capture and access through third parties such as Facebook, D&B, and others. Personal user information, geo location data, social graphs, user-generated content, machine logging data, and sensor-generated data are just a few examples of the ever-expanding array of data being captured. It's not surprising that developers find increasing value in leveraging this data to enrich existing applications and create new ones made possible by it. The use of the data is rapidly changing the nature of communication, shopping, advertising, entertainment, and relationship management. Applications that don't find ways to leverage it quickly will quickly fall behind.

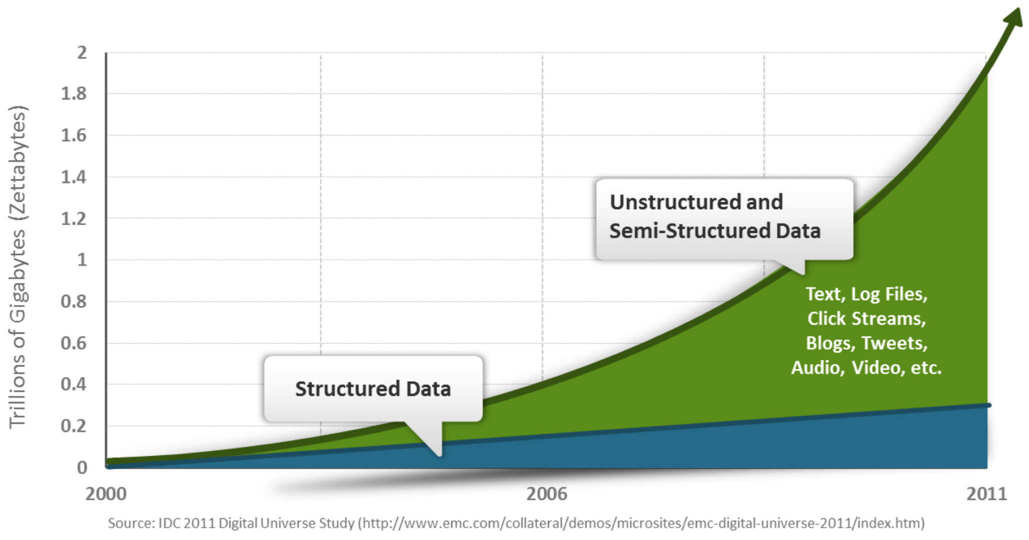


Figure 2. Big Data: The amount of data is growing rapidly, and the nature of data is changing as well. More than 80% of data generated today is unstructured or semi-structured.

The capture and use of the data creates the need for a very different type of database, however. Developers want a very flexible database that easily accommodates any new type of data they want to work with and is not disrupted by content structure changes from third-party data providers. Much of the new data is unstructured and semi-structured, so developers also need a database that is capable of efficiently storing it. Unfortunately, the rigidly defined, schema-based approach used by relational databases makes it impossible to quickly incorporate new types of data, and is a poor fit for unstructured and semi-structured data.

Finally, with the rising importance of processing data, developers are increasingly frustrated with the "impedance mismatch" between the object-oriented approach they use to write applications and the schema-based tables and rows of a relational database. NoSQL provides a data model that maps better to the application's organization of data and simplifies the interaction between the application and the database, resulting in less code to write, debug, and maintain.

Cloud Computing

Not long ago, most consumer and many business applications were single-user applications that ran on your PC. Most data-intensive, multi-user business applications used a two-tier, client-server architecture that ran inside the firewall and supported a limited number of users. Today, most new applications (both consumer and business) use a three-tier Internet architecture, run in a public or private cloud, and support large numbers of users. Along with this shift in software architecture, new business models like software-as-a-service (SaaS) and advertising-based models have become more prevalent.

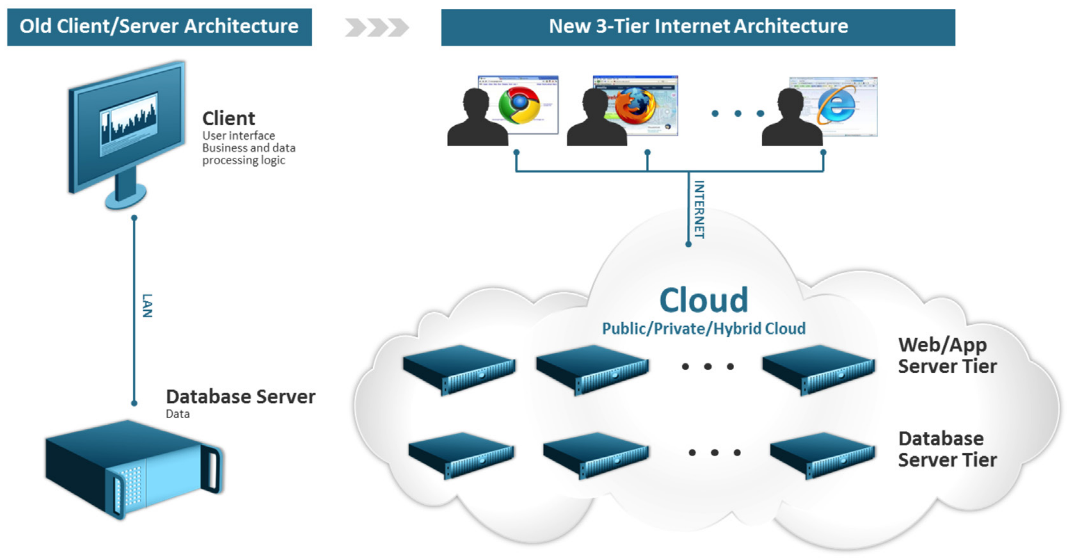

Old Client/Server Architecture
New 3-Tier Internet Architecture
Client
User interface
Business and data processing logic
LAN
Database Server
Data
INTERNET
Cloud
Public/Private/Hybrid Cloud
Web/App Server Tier
Database Server Tier

Figure 3. Applications today are increasingly developed using a three-tier internet architecture, requiring a horizontally scalable database tier that easily scales with the number of users and amount of data your application has.

In the three-tier architecture, applications are accessed through a web browser or mobile app that is connected to the Internet. In the cloud, a load balancer directs the incoming traffic to a scale-out tier of web/application servers that process the logic of the application. The scale-out architecture at the web/application tier works beautifully. For every 10,000 (or however many) new concurrent users, you simply add another commodity server to the web/application tier to absorb the load.

At the database tier, relational databases were originally the popular choice. Their use was increasingly problematic however, because they are a centralized, share-everything technology that scales up rather than out. This made them a poor fit for applications that require easy and dynamic scalability. NoSQL technologies have been built from the ground up to be distributed, scale-out technologies and therefore fit better with the highly distributed nature of the three-tier Internet architecture.

Closer look at why developers are considering NoSQL databases

Big Users, Big Data, and Cloud Computing are changing the way many applications are being developed. The industry has been dominated by relational databases for 40 years, but application developers are increasingly turning to NoSQL databases to meet new challenges for three main reasons:

1. Better application development productivity through a more flexible data model;
2. Greater ability to scale dynamically to support more users and data;
3. Improved performance to satisfy expectations of users wanting highly responsive applications and to allow more complex processing of data.

NoSQL's more flexible data model

Relational and NoSQL data models are very different. The relational model takes data and separates it into many interrelated tables. Each table contains rows and columns where a row might contain lots of information about a person and each column might contain a value for a specific attribute associated with that person, like his age. Tables reference each other through foreign keys that are stored in columns as well.

The relational model minimizes the amount of storage space required, because each piece of data is only stored in one place – a key requirement during when relational databases were created and disk storage was very. However, space efficiency comes at expense of increased complexity when looking up data. The desired information needs to be collected from many tables (often hundreds in today's enterprise applications) and combined before it can be provided to the application. Similarly, when writing data, the write needs to be coordinated and performed on many tables.

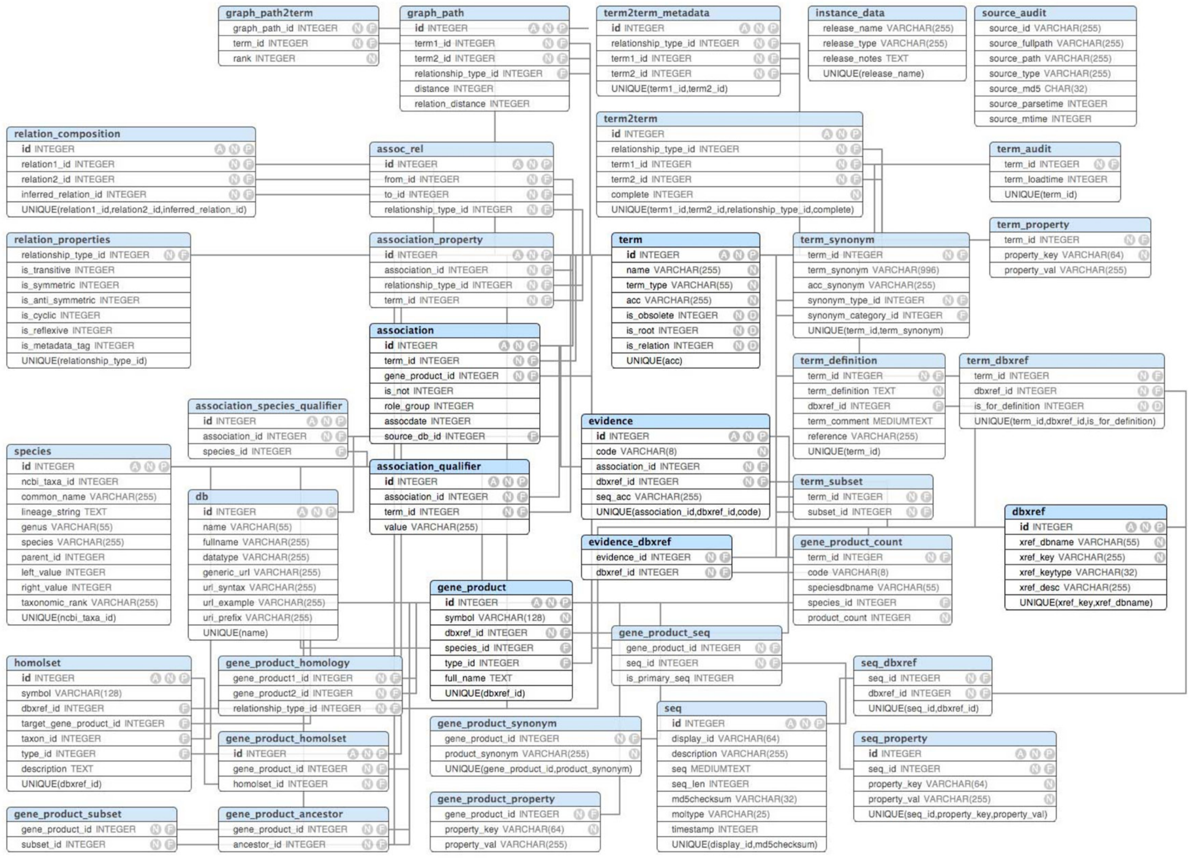
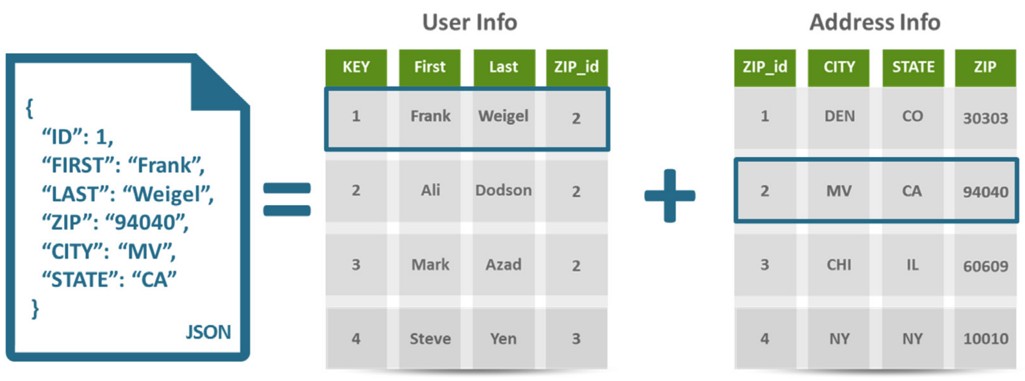


Figure 4: With relational databases, any operation requires the collection and processing of data from across tens or hundreds of interrelated tables, greatly hindering performance.

NoSQL databases have a very different model. For example, a document-oriented NoSQL database takes the data you want to store and aggregates it into documents using the JSON format. Each JSON document can be thought of as an object to be used by your application. A JSON document might, for example, take all the data stored in a row that spans 20 tables of a relational database and aggregate it into a single document/object. Aggregating this information may lead to duplication of information, but since storage is no longer cost prohibitive, the resulting data model flexibility, ease of efficiently distributing the resulting documents and read and write performance improvements make it an easy trade-off for web-based applications.



```
{
  "ID": 1,
  "FIRST": "Frank",
  "LAST": "Weigel",
  "ZIP": "94040",
  "CITY": "MV",
  "STATE": "CA"
}
JSON
```

User Info

KEY	First	Last	ZIP_id
1	Frank	Weigel	2
2	Ali	Dodson	2
3	Mark	Azad	2
4	Steve	Yen	3

Address Info

ZIP_id	CITY	STATE	ZIP
1	DEN	CO	30303
2	MV	CA	94040
3	CHI	IL	60609
4	NY	NY	10010

Figure 5: Unlike relational databases, which must store and retrieve data from scores of interrelated tables, document databases can store an entire object in a single JSON document, making it faster to retrieve.

Developers generally use object-oriented programming languages to build applications. It's usually most efficient to work with data that's in the form of an object with a complex structure consisting of nested data, lists, arrays, etc. The relational data model provides a very limited data structure that doesn't map well to the object model. Instead data must be stored and retrieved from tens or even hundreds of interrelated tables. Object-relational frameworks provide some relief but the fundamental impedance mismatch still exists between the way an application would like to see its data and the way it's actually stored in a relational database.

Document databases, on the other hand, can store an entire object in a single JSON document and support complex data structures. This makes it easier to conceptualize data as well as write, debug, and evolve applications, often with fewer lines of code.

Another major difference is that relational technologies have rigid schemas while NoSQL models are schemaless. Relational technology requires strict definition of a schema prior to storing any data into a database. Changing the schema once data is inserted is a big deal. Want to start capturing new information not previously considered? Want to make rapid changes to application behavior requiring changes to data formats and content? With relational technology, changes like these are extremely disruptive and frequently avoided, which is the exact opposite of the behavior desired in the Big Data era, where application developers need to constantly – and rapidly – incorporate new types of data to enrich their applications.

In comparison, document databases are schemaless, allowing you to freely add fields to JSON documents without having to first define changes. The format of the data being inserted can be changed at any time, without application disruption. This allows application developers to move quickly to incorporate new data into their applications.

The stark differences between relational and NoSQL data models have caught the attention of application development teams. Whether or not they ever scale beyond a single machine, growing numbers of development teams feel they can be far more productive using the data models embodied in NoSQL databases.

NoSQL's scalability and performance advantage

To deal with the increase in concurrent users (Big Users) and the amount of data (Big Data), applications and their underlying databases need to scale using one of two choices: scale up or scale out. Scaling up implies a centralized approach that relies on bigger and bigger servers. Scaling out implies a distributed approach that leverages many standard, commodity physical or virtual servers.

Scale out: an excellent approach at the web/application tier

At the web/application tier of the three-tier Internet architecture, a scale out approach has been the default for many years and worked extremely well. (See Figure 6.) As more people use an application, more commodity servers are added to the web/application tier, performance is maintained by distributing load across an increased number of servers, and the cost scales linearly with the number of users.

While the performance and cost curves of this approach are attractive, the flexibility of the approach is an equally big win. As users come and go, commodity servers (or virtual machines) can be quickly added or removed from the server pool, matching capital and operating costs to the difficult-to-predict size and activity level of the user population. And, by distributing the load across many servers (even across geographies) the system is inherently fault tolerant, supporting continuous operations.

Another advantage is that new software upgrades can be rolled out gradually across subsets of the overall server pool. Facebook, as an example, slowly dials up new functionality by rolling out new software to a subset of their entire application server tier (and user population) in a stepwise manner. If issues crop up, servers can be quickly reverted to the previous known good version. All this can be done without ever taking the application offline.

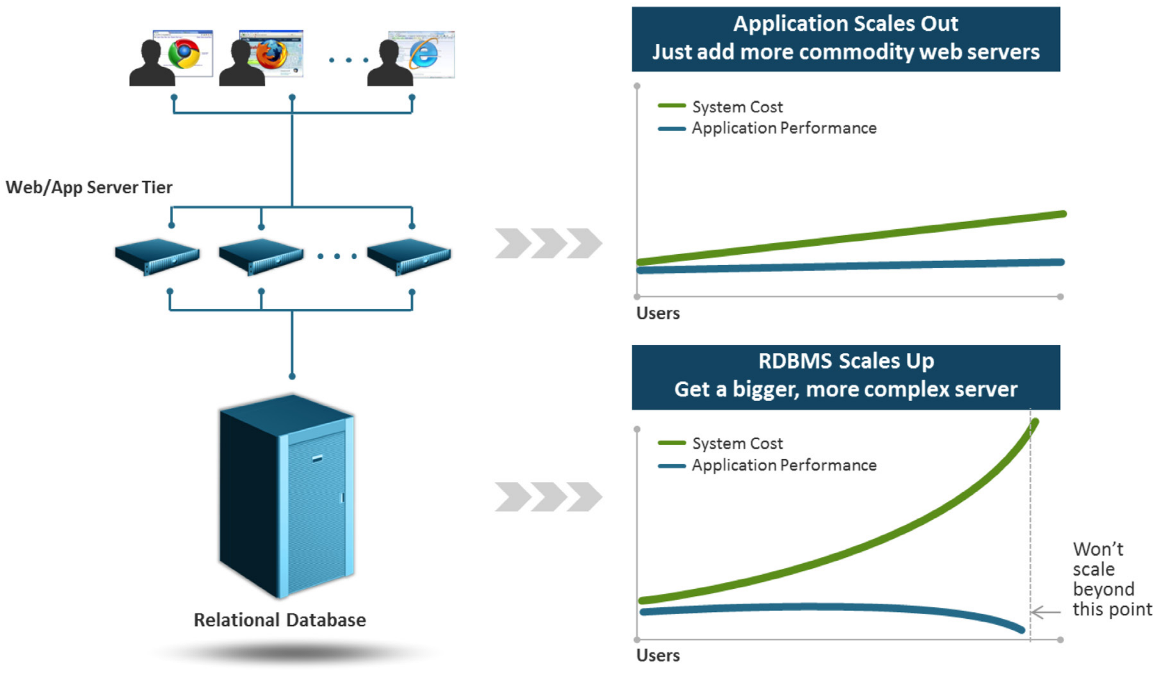


Figure 6. With relational databases, to support more users or store more data, you need a bigger server with more CPUs, more memory, and more disk storage.

Scale up with relational technology: limitations at the database tier

Prior to NoSQL databases, the default scaling approach at the database tier was to scale up. This was dictated by the fundamentally centralized, shared-everything architecture of relational database technology. To support more concurrent users and/or store more data, you need a bigger and bigger server with more CPUs, more memory, and more disk storage to keep all the tables. Big servers tend to be highly complex, proprietary, and disproportionately expensive, unlike the low-cost, commodity hardware typically used so effectively at the web/application server tier.

(Clustered relational databases, like Oracle RAC, work on the concept of a shared disk subsystem. They use a cluster-aware file system that writes to a highly available disk subsystem – but this means the cluster still has the disk sub-system as a single point of failure.)

With relational technology, upgrading a server is an exercise that requires planning, acquisition and application downtime to complete. Given the relatively unpredictable user growth rate of today's interactive applications, it's hard to avoid over- or under-provisioning resources. Too much and you've overspent; too little and users can have a bad experience or the application can outright fail. And, since all the eggs are in a single, centralized database basket, you have to get your fault tolerance and high-availability strategies exactly right.

Scale out with NoSQL technology at the database tier

Techniques used to extend the useful scope of relational technology – sharding, denormalizing, distributed caching – fight symptoms but not the disease itself In fact, they attempt to disguise one simple fact: relational technology is suboptimal for many modern interactive applications.

NoSQL databases were developed from the ground up to be distributed, scale out databases. They use a cluster of standard, physical or virtual servers to store data and support database operations. To scale, additional servers are joined to the cluster and the data and database operations are spread across the larger cluster. Since commodity servers are expected to fail from time-to-time, NoSQL databases are built to tolerate and recover from such failure making them highly resilient.

NoSQL databases provide a much easier, linear approach to database scaling. If 10,000 new users start using your application, simply add another database server to your cluster. Add ten thousand more users and add another server. There's no need to modify the application as you scale since the application always sees a single (distributed) database.

At scale, a distributed scale out approach also usually ends up being cheaper than the scale up alternative. This is a consequence of large, complex, fault tolerant servers being expensive to design, build and support. Licensing costs of commercial relational databases can also be prohibitive because they are priced with a single server in mind. NoSQL databases on the other hand are generally open source, priced to operate on a cluster of servers, and relatively inexpensive.

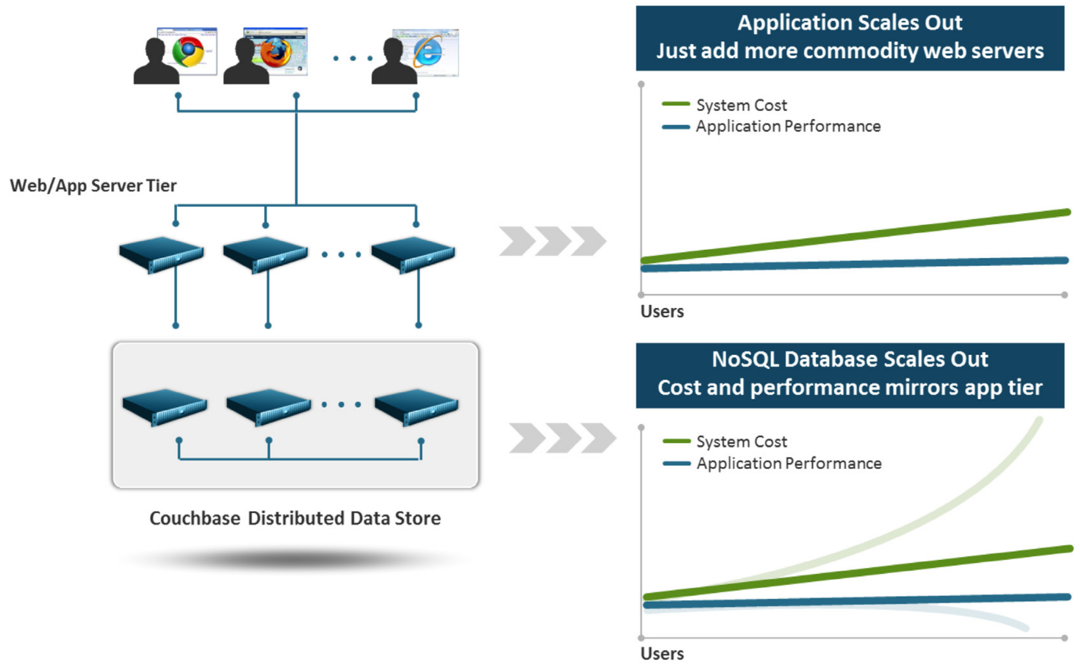

Web/App Server Tier
Couchbase Distributed Data Store
Application Scales Out
Just add more commodity web servers
System Cost
Application Performance
Users
NoSQL Database Scales Out
Cost and performance mirrors app tier
System Cost
Application Performance
Users

Figure 7: NoSQL databases provide a more linear, scalable approach to scaling than do relational databases.

While implementations differ, NoSQL databases share some characteristics with respect to scaling and performance:

- Auto-sharding – A NoSQL database automatically spreads data across servers, without requiring applications to participate. Servers can be added or removed from the data layer without application downtime, with data (and I/O) automatically spread across the servers. Most NoSQL databases also support data replication, storing multiple copies of data across the cluster, and even across data centers, to ensure high availability and support disaster recovery. A properly managed NoSQL database system should never need to be taken offline, for any reason, supporting 24x365 continuous operation of applications.

- Distributed query support – "Sharding" a relational database can reduce, or eliminate in certain cases, the ability to perform complex data queries. NoSQL database systems retain their full query expressive power even when distributed across hundreds of servers.

- Integrated caching – To reduce latency and increase sustained data throughput, advanced NoSQL database technologies transparently cache data in system memory. This behavior is transparent to the application developer and the operations team, compared to relational technology where a caching tier is usually a separate infrastructure tier that must be developed to, deployed on separate servers, and explicitly managed by the ops team.

Summary

Application needs have been changing dramatically, due in large part to three trends: growing numbers of users that applications must support (along with growing user expectations for how applications perform); growth in the volume and variety of data that developers must work with; and the rise of cloud computing, which relies on a distributed three-tier Internet architecture. NoSQL technology is rising rapidly among Internet companies and the enterprise because it offers data management capabilities that meet the needs of modern application:

- Better application development productivity through a more flexible data model;

- Greater ability to scale dynamically to support more users and data;

- Improved performance to satisfy expectations of users wanting highly responsive applications and to allow more complex processing of data.

NoSQL is increasingly considered a viable alternative to relational databases, and should be considered particularly for interactive web and mobile applications.

About Couchbase

We're the company behind the Couchbase open source project, a vibrant community of developers and users of Couchbase document-oriented database technology. Our flagship product, Couchbase Server, is a packaged version of Couchbase technology that's available in Community and Enterprise Editions. We're known for our easy scalability, consistent high performance, 24x365 availability, and a flexible data model. Companies like AOL, Cisco, Concur, LinkedIn, Orbitz, Salesforce.com, Shuffle Master, Zynga and hundreds of others around the world use Couchbase Server for their interactive web and mobile applications. www.couchbase.com