



The Official Ten-Year
Retrospective of

NEWSQL

NEWSQL

adjective \ˈnū-sē-kwəl \

A category of relational DBMSs designed to support scalable workloads for operational applications.

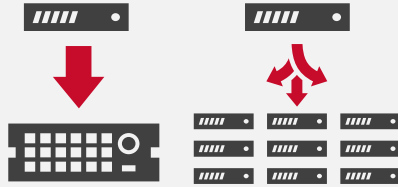


Ted Codd

A category of **relational** DBMSs designed to support scalable workloads for operational applications.



Ted Codd



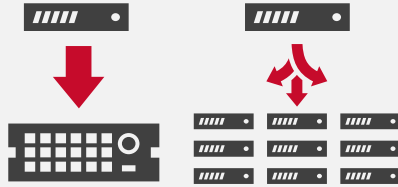
Vertical

Horizontal

A category of **relational** DBMSs designed to support **scalable** workloads for operational applications.



Ted Codd



Vertical

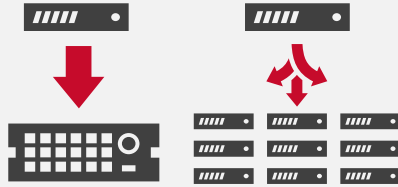
Horizontal

OLTP

A category of relational DBMSs designed to support scalable workloads for operational applications.



Ted Codd



Vertical

Horizontal



Fast



Repetitive



Small

A category of **relational** DBMSs designed to support **scalable** workloads for **operational** applications.

Outline

How did the **NewSQL** trend start?

Is the **NewSQL** term still relevant?

What is the future of OLTP DBMSs?



Twenty-First Century of DATABASE SYSTEMS

Early 2000s – The Internet Boom

It was now possible for a small organization to build an application that could be used by many concurrent users.

Database scalability challenges were no longer limited to major corporations.

Early 2000s – Legacy Systems

Single-node deployments
using old "elephant" DBMSs.

Viable open-source options
did not exist.

Can only scale vertically.

Expensive software + hardware.



ORACLE®

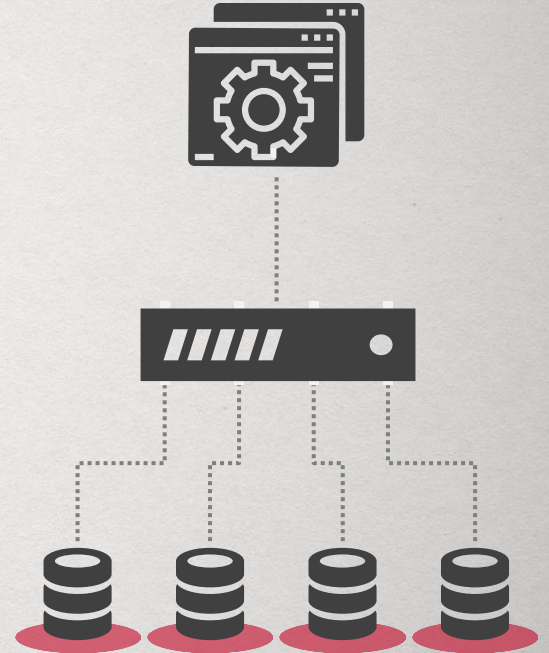
Microsoft®
SQL Server

Mid 2000s – Sharding Middleware

Combine multiple nodes into a logical database.

Route / rewrite queries to access data at specific nodes.

Development cost.
Limited functionality.



Late 2000s – NoSQL

Forgo DBMS-enforced protections to achieve high-availability and high-scalability.

Non-relational data models without schemas.

No transaction guarantees.

Custom query APIs.

Late 2000s – NoSQL

Forgo DBMS-enforced pro
high-availability and high-s

Non-relational data model

No transaction guarantees

Custom query APIs.

To appear in O

Bigtable is a distributed data storage system. It is designed to store petabytes of data on thousands of servers. Many people use it for web pages to state. Despite these various Google people data model. Dynamic control. scribe the design.

I. Introduction

Over the last two years, we have implemented, and for managing structured data and thousands of servers. Performance, and more than sixty million Google Analytics. We use Bigtable for jobs to latency. The Bigtable cluster of configuration servers, and store. In many ways, many implement relational databases [14].

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilson,hkerr,m3h,tushar,fikes,gruber}@google.com

Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jambani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels
Amazon.com

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

Categories and Subject Descriptors

D.4.2 [Operating Systems]: Storage Management; D.4.5 [Operating Systems]: Reliability; D.4.2 [Operating Systems]: Performance;

General Terms

Algorithms, Management, Measurement, Performance, Design, Reliability.

1. INTRODUCTION

Amazon runs a worldwide e-commerce platform that serves tens of millions of customers at peak times using tens of thousands of servers located in many data centers around the world. There are strict operational requirements on Amazon's platform in terms of performance, reliability and efficiency, and to support continuous growth the platform needs to be highly scalable. Reliability is one of the most important requirements because even the slightest outage has significant financial consequences and impacts customer trust. In addition, to support continuous growth, the platform needs to be highly scalable.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full copyright notice on the first page.

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service-oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornadoes. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

Dealing with failures in an infrastructure comprised of millions of components is our standard mode of operation; there are always a small but significant number of server and network components that are failing at any given time. As such Amazon's software systems need to be constructed in a manner that treats failure handling as the normal case without impacting availability or performance.

To meet the reliability and scaling needs, Amazon has developed a number of storage technologies, of which the Amazon Simple Storage Service (also available outside of Amazon and known as Amazon S3), is probably the best known. This paper presents the design and implementation of Dynamo, another highly available and scalable distributed data store built for Amazon's platform. Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the performance. Amazon's platform has a very diverse set of applications with different storage requirements. A select set of application designers configure their data store appropriately based on these tradeoffs to achieve high availability and guaranteed performance in the most cost effective manner.

There are many services on Amazon's platform that only need primary-key access to a data store. For many services, such as those that provide best seller lists, shopping carts, customer preferences, session management, sales rank, and product catalogs, the common pattern of using a relational database would lead to inefficiencies and limit scale and availability. Dynamo provides a simple primary-key only interface to meet the requirements of these applications.

Dynamo uses a synthesis of well known techniques.

Key/Value



←EROSPIKE→



Column-Family



APACHE
HBASE



Documents



RethinkDB

Late 2000s – NoSQL

Forgo DBMS-enforced protections to achieve high-availability and high-scalability.

Non-relational data models without schemas.

No transaction guarantees.

Custom query APIs.

Application must handle eventually consistent data, lack of transactions, and joins.

Late 2000s – NoSQL

Forgo DBMS-enforced protection of high-availability and high-scalability. Non-relational data models with No transaction guarantees. Custom query APIs.

Application must handle even more data, lack of transactions, and

Spanner: Google's Globally-Distributed Database

James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, Sarjay Ghemawat, Andrew Gubarev, Christopher Heiser, Peier Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mvasara, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, Dale Woodford

Google, Inc.

Abstract

Spanner is Google's scalable, multi-version, globally-distributed, and synchronously-replicated database. It is the first system to distribute data at global scale and support externally-consistent distributed transactions. This paper describes how Spanner is structured, its feature set, the rationale underlying various design decisions, and a novel time API that exposes clock uncertainty. This API and its implementation are critical to supporting external consistency and a variety of powerful features: non-blocking reads in the past, lock-free read-only transactions, and atomic schema changes, across all of Spanner.

1 Introduction

Spanner is a scalable, globally-distributed database designed, built, and deployed at Google. At the highest level of abstraction, it is a database that shards data across many sets of Paxos [21] state machines in datacenters spread all over the world. Replication is used for global availability and geographic locality; clients automatically failover between replicas. Spanner automatically reshards data across machines as the amount of data or the number of servers changes, and it automatically migrates data across machines (even across datacenters) to balance load and in response to failures. Spanner is designed to scale up to millions of machines across hundreds of datacenters and trillions of database rows.

Applications can use Spanner for high availability, even in the face of wide-area natural disasters, by replicating their data within or even across continents. Our initial customer was F1 [35], a rewrite of Google's advertising backend. F1 uses five replicas spread across the United States. Most other applications will probably replicate their data across 3 to 5 datacenters in one geographic region, but with relatively independent failure modes. That is, most applications will choose lower la-

tency over higher availability, as long as they can survive 1 or 2 datacenter failures.

Spanner's main focus is managing cross-datacenter replicated data, but we have also spent a great deal of time in designing and implementing important database features on top of our distributed-systems infrastructure. Even though many projects happily use Bigtable [9], we have also consistently received complaints from users that Bigtable can be difficult to use for some kinds of applications: those that have complex, evolving schemas, or those that want strong consistency in the presence of wide-area replication. (Similar claims have been made by other authors [37].) Many applications at Google have chosen to use Megastore [5] because of its semi-relational data model and support for synchronous replication, despite its relatively poor write throughput. As a consequence, Spanner has evolved from a Bigtable-like versioned key-value store into a temporal multi-version database. Data is stored in schematized semi-relational tables; data is versioned, and each version is automatically timestamped with its commit time; old versions of data are subject to configurable garbage-collection policies; and applications can read data at old timestamps. Spanner supports general-purpose transactions, and provides a SQL-based query language.

As a globally-distributed database, Spanner provides several interesting features. First, the replication configurations for data can be dynamically controlled at a fine grain by applications. Applications can specify constraints to control which datacenters contain which data, how far data is from its users (to control read latency), how far replicas are from each other (to control write latency), and how many replicas are maintained (to control durability, availability, and read performance). Data can also be dynamically and transparently moved between datacenters by the system to balance resource usage across datacenters. Second, Spanner has two features that are difficult to implement in a distributed database: it

Published in the Proceedings of OSDI 2012

Late 2000s – NoSQL

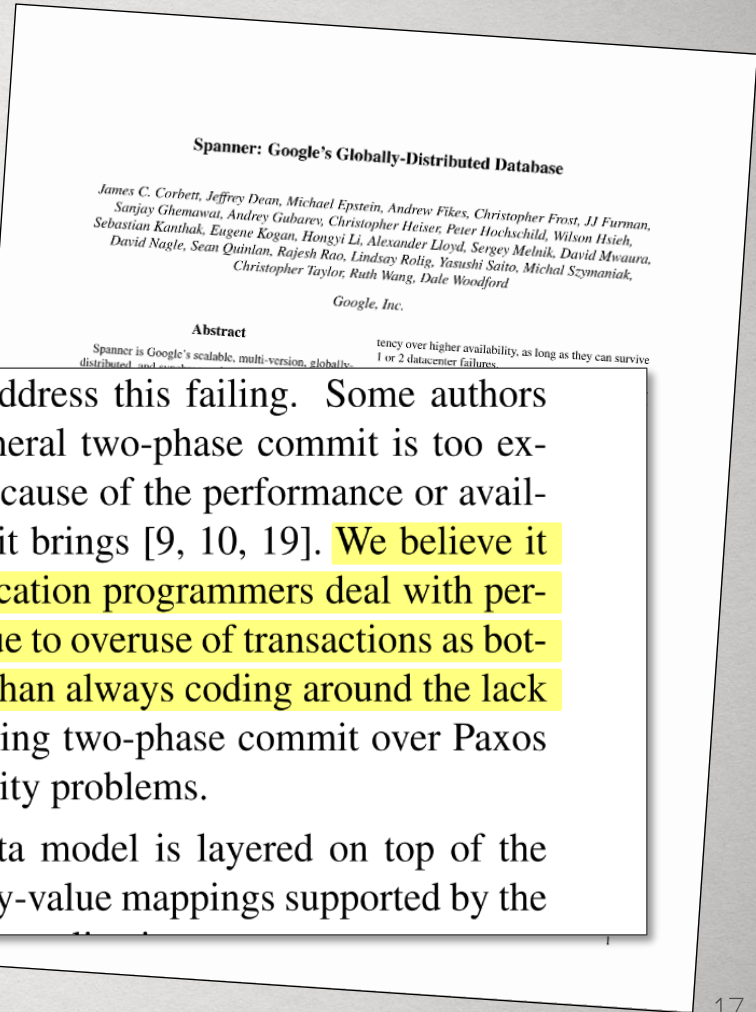
Forgo DBMS-enforced protection
high-availability and high-scalability

Non-relational data models

No transaction guarantees

Custom query APIs

*Application must handle
data, lack of transactions, and*





The Rise of NEWSQL SYSTEMS

Aslett Report (2011)

[Systems that] deliver the scalability and flexibility promised by NoSQL while retaining the support for SQL queries and/or ACID, or to improve performance for appropriate workloads.

Matt Aslett – 451 Group (April 4th, 2011)

<https://www.451research.com/report-short?entityId=66963>

the 451 group

451 TECHDEALMAKER

4 April 2011 – Sector IQ

How will the database incumbents respond to NoSQL and NewSQL?

Analyst: Matt Aslett

The acquisition of **MySQL AB** by **Sun Microsystems** in January 2008 appeared to signal that open source databases were on the brink of opening up a new battleground against the proprietary database giants. In announcing the deal, Sun signaled its intention to provide the support and development resources required for MySQL to challenge the established vendors in supporting mission-critical, high-performance applications on Web-based architectures. Needless to say, reality was somewhat different as Sun faced wider problems of its own and eventually succumbed to takeover by **Oracle** (Nasdaq: ORCL) in April 2009, in doing so handing ownership of the leading commercial open source database to the database heavyweight.

We had previously argued that MySQL was very much the crown jewel of the open source database world thanks to its focus on Web applications, its lightweight architecture and its fast read capabilities, which made it potentially complementary technology for all of the established database players. Additionally, if Oracle's major rivals were seeking an obvious alternative to MySQL in 2009, they were out of luck.

Just two years later, however, the database market is awash with open source databases with lightweight architectures targeted at Web applications. Not only have the likes of **Monty Program** and **SkysQL** emerged to provide alternative support for MySQL and its forks, but there are also a large number of products available under the banner of NoSQL, which emerged in mid-2009 as an umbrella term for a loosely affiliated collection of non-relational database projects. We have also seen the emergence of what we have termed 'NewSQL' database offerings, with companies promising to deliver the scalability and flexibility promised by NoSQL while retaining the support for SQL queries and/or ACID (atomicity, consistency, isolation and durability), or to improve performance for appropriate workloads to the extent that the advanced scalability promised by some NoSQL databases becomes irrelevant.

From MySQL to NoSQL

Despite being a good match for many read-intensive applications, MySQL does not provide predictable performance at scale, particularly with a few writes thrown into the mix. The memcached distributed memory object-caching system can be used – and has been widely adopted – to improve performance but does not provide any persistence and lacks consistency. To some extent, the rise of NoSQL has been driven by the inadequacies of

451 TechDealMaker
Copyright 2011 The 451 Group

4 April 2011
Page 1 of 5

Stonebraker Article (2011)

*SQL as the primary interface.
ACID support for transactions
High per-node performance.
Non-locking concurrency control.
Shared-nothing architecture.*

Mike Stonebraker – Blog@CACM (June 16th, 2011)
<http://cacm.acm.org/blogs/blog-cacm/109710>

COMMUNICATIONS
OF THE
ACM TRUSTED INSIGHTS FOR COMPUTING'S LEADING PROFESSIONALS


Home » Blogs » [BLOG@CACM](#) » [New SQL: An Alternative to NoSQL and Old SQL for New...](#) » [Full Text](#)

[BLOG@CACM](#)

New SQL: An Alternative to NoSQL and Old SQL for New OLTP Apps

Michael Stonebraker

June 16, 2011



Historically, Online Transaction Processing (OLTP) was performed by customers submitting traditional transactions (order something, withdraw money, cash a check, etc.) to a relational DBMS. Large enterprises might have dozens to hundreds of these systems. Invariably, enterprises wanted to consolidate the information in these OLTP systems for business analysis, cross selling, or some other purpose. Hence, Extract-Transform-and-Load (ETL) products were used to convert OLTP data to a common format and load it into a data warehouse. Data warehouse activity rarely shared machine resources with OLTP because of lock contention in the DBMS and because business intelligence (BI) queries were so resource-heavy that they got in the way of timely responses to transactions.

This combination of a collection of OLTP systems, connected to ETL, and connected to one or more data warehouses is the gold standard in enterprise computing. I will term it "Old OLTP." By and large, this activity was supported by the traditional RDBMS vendors. In the past I have affectionately called them "the elephants"; in this posting I refer to them as "Old SQL."

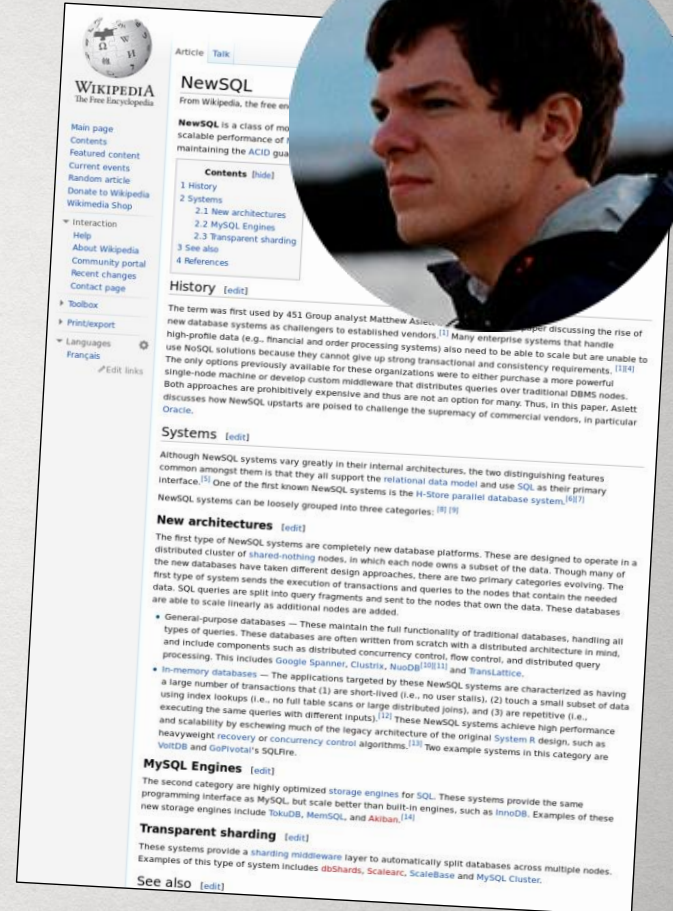
As noted by most pundits, "the Web changes everything," and I have noticed a very different collection of OLTP requirements that are emerging for Web properties, which I will term "New OLTP." These sites seem to be driven by two customer requirements:

The need for far more OLTP throughput. Consider new Web-based applications such as multi-player games, social networking sites, and online gambling networks. The aggregate number of interactions per second is skyrocketing for the successful Web properties in this category. In addition, the explosive growth of smartphones has created a market for applications that use the phone as a geographic sensor and provide location-based services. Again, successful

Wikipedia Article (2012)

A class of modern relational DBMSs that provide the same scalable performance of NoSQL systems for OLTP workloads while still maintaining the ACID guarantees of a traditional DBMS.

Wikipedia (October 2012)
<http://en.wikipedia.org/wiki/NewSQL>



WIKIPEDIA
The Free Encyclopedia

Article Talk

NewSQL

From Wikipedia, the free encyclopedia

NewSQL is a class of modern relational database management systems (DBMSs) that aim to provide the scalability and performance of NoSQL systems for OLTP workloads while still maintaining the ACID guarantees of a traditional DBMS.

Contents [hide]

- History
- Systems
 - 2.1 New architectures
 - 2.2 MySQL Engines
 - 2.3 Transparent sharding
- See also
- References

History

The term was first used by 451 Group analyst Matthew Aslett in a paper discussing the rise of new database systems as challenges to established vendors.^[1] Many enterprise systems that handle high-profile data (e.g., financial and order processing systems) also need to be able to scale but are unable to use NoSQL solutions because they cannot give up strong transactional and consistency requirements.^{[1][8]} The only options previously available for these organizations were to either purchase a more powerful single-node machine or develop custom middleware that distributes queries over traditional DBMS nodes. Both approaches are prohibitively expensive and thus are not an option for many. Thus, in this paper, Aslett discusses how NewSQL startups are poised to challenge the supremacy of commercial vendors, in particular Oracle.

Systems

Although NewSQL systems vary greatly in their internal architectures, the two distinguishing features common amongst them is that they all support the relational data model and use SQL as their primary interface.^[5] One of the first known NewSQL systems is the H-Store parallel database system.^{[6][7]} NewSQL systems can be loosely grouped into three categories:^[9] [9]

New architectures

The first type of NewSQL systems are completely new database platforms. These are designed to operate in a distributed cluster of shared-nothing nodes, in which each node owns a subset of the data. Though many of the new databases have taken different design approaches, there are two primary categories evolving. The first type of system sends the execution of transactions and queries to the nodes that contain the needed data. SQL queries are split into query fragments and sent to the nodes that own the data. These databases are able to scale linearly as additional nodes are added.

- General-purpose databases** — These maintain the full functionality of traditional databases, handling all types of queries. These databases are often written from scratch with a distributed architecture in mind, and include components such as distributed concurrency control, flow control, and distributed query processing. This includes *Google Spanner*, *Clustrix*, *NuoDB*^{[10][11]} and *TransLattice*.
- In-memory databases** — The applications targeted by these NewSQL systems are characterized as having a large number of transactions that (1) are short-lived (i.e., no user stalls), (2) touch a small subset of data using index lookups (i.e., no full table scans or large distributed joins), and (3) are repetitive (i.e., executing the same queries with different inputs).^[12] These NewSQL systems achieve high performance and scalability by eschewing much of the legacy architecture of the original System R design, such as heavyweight recovery or concurrency control algorithms.^[13] Two example systems in this category are *VertiDB* and *GoPivotal's SQLFire*.

MySQL Engines

The second category are highly optimized storage engines for SQL. These systems provide the same programming interface as MySQL but scale better than built-in engines, such as *InnoDB*. Examples of these new storage engines include *TokuDB*, *MemSQL*, and *Akiban*.^[14]

Transparent sharding

These systems provide a sharding middleware layer to automatically split databases across multiple nodes. Examples of this type of system includes *distdards*, *Scalearc*, *ScaleBase* and *MySQL Cluster*.

See also [edit]



SIGMOD Rec Article (2016)

Refinement of NewSQL categories and properties.

New Architectures Middleware Database-as-a-Service

Andy Pavlo, Matt Aslett – SIGMOD Record (June 2016)
<https://dl.acm.org/doi/10.1145/3003665.3003674>

What's Really New with NewSQL?

Andrew Pavlo
Carnegie Mellon University
pavlo@cs.cmu.edu

Matthew Aslett
451 Research
matthew.aslett@451research.com

ABSTRACT

A new class of database management systems (DBMSs) called **NewSQL** uses their ability to scale modern on-line transaction processing (OLTP) workloads in a way that is not possible with legacy systems. The term NewSQL was first used by one of the authors of this article in a 2011 business analysis report discussing the rise of new database systems as challengers to these established vendors (Oracle, IBM, Microsoft). The other author was working on what became one of the first examples of a NewSQL DBMS. Since then several companies and research projects have used this term (rightly and wrongly) to describe their systems.

Given that relational DBMSs have been around for over four decades, it is justifiable to ask whether the claim of NewSQL's superiority is actually true or whether it is simply marketing. If they are indeed able to get better performance, then the next question is whether there is anything scientifically new about them that enables them to achieve these gains or is it just that earlier years are no longer a problem.

To do this, we first discuss the history of databases to understand how NewSQL systems came about. We then provide a detailed explanation of what the term NewSQL means and the different categories of systems that fall under this definition.

1. A BRIEF HISTORY OF DBMS

The first DBMSs came on-line in the mid 1960s. One of the first was IBM's IMS that was built to keep track of the supplies and parts inventory for the Saturn V and Apollo space exploration process. It helped introduce the idea that an application code should be separate from the data that it operates on. This allows developers to write applications that only focus on the access and manipulation of data, and not the complications and overhead associated with how to actually perform these operations. IMS was later followed by the pioneering work in the early 1970s on the first relational DBMSs, IBM's System R and the University of California's INGRES. INGRES was soon adopted at other universities for their information systems and was subsequently commercialized in the late 1970s. Around the same time, Oracle released the first version of their DBMS that was similar to System R's design. Other competitors of the first commercial DBMSs, including Sybase and Informix. Although IBM never made System R available to the public, it later released a new relational DBMS (DB2) in 1983 that used parts of the System R code base.

The late 1980s and early 1990s brought about a new class of DBMSs that were designed to overcome the much touted impedance mismatch between the relational model and object-oriented programming languages [65]. These object-oriented DBMSs, however, never saw wide-spread market adoption because they lacked a standard interface like SQL. But many initial DBMSs when the major vendors added object and XML/NoSQL systems over two decades later.

The other notable event during the 1990s was the start of today's two major open-source DBMS projects. MySQL was started in Sweden in 1995 based on the earlier ISAM based MySQL system. PostgreSQL began in 1994 when two Berkeley graduate students forked the original SQL-based Postgres code from the 1980s to add support for SQL.

The 2000s brought the arrival of Internet applications that had more challenging resource requirements than applications from previous years. They needed to scale to support large number of concurrent users and had to be on-line all the time. But the database for these new applications was consistently much greater than when DBMSs and hardware could support their DBMS vertically by moving the database to a machine with better hardware. This, however, only improves performance so much and has diminishing returns. Furthermore, moving the database from one machine to another is a complex process and often requires significant downtime, which is unacceptable for these Web-based applications. To overcome this problem, some companies created custom *middleware* to shard single-node DBMSs over a cluster of less expensive machines. Such middleware presents a single logical database to the application that is stored across multiple physical nodes. When the application issues queries against this database, the middleware redirects and/or rewrites them to distribute their cost these queries and send the results back to the middleware. The nodes execute them and send the results back to the middleware. The nodes which then coalesce them into a single response to the application. Two notable examples of this middleware approach based cluster [54]. This approach was later adopted by Facebook for their own MySQL cluster that is still used today.

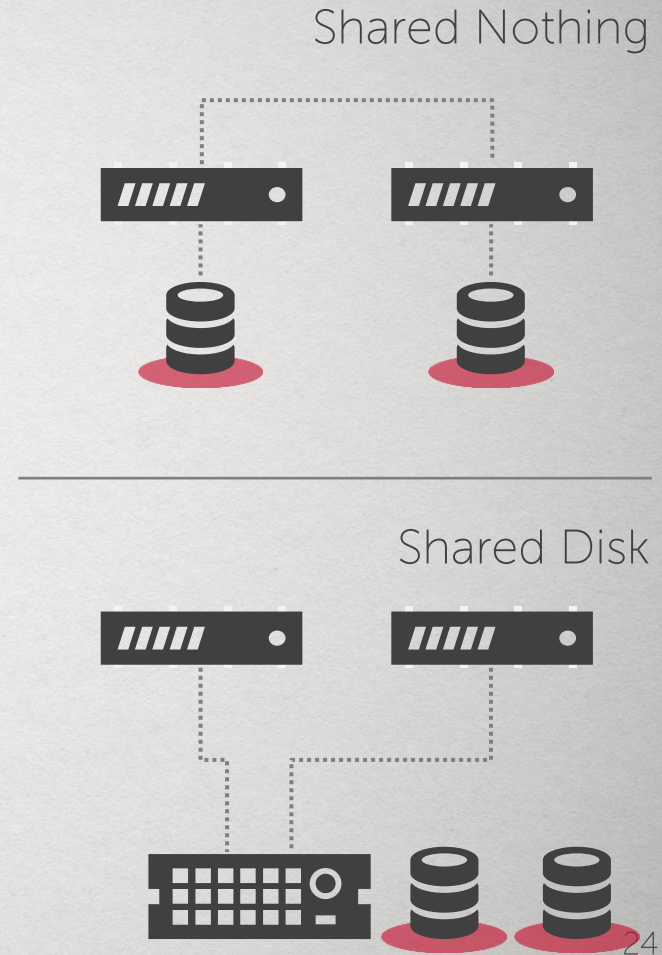
Sharding middleware works well for simple operations like reading or updating a single record. It is more difficult, however, to execute queries that update more than one record in a transaction or join tables. As such, these early middleware

New Architectures

New codebase written from scratch without architectural baggage of legacy systems.

Almost all DBMSs use a **shared-nothing** architecture.

Our mistake was to not include **shared-disk** DBMSs.



New Architectures

New codebase written from scratch without architectural baggage of legacy systems.

Almost all DBMSs use a **shared-nothing** architecture.

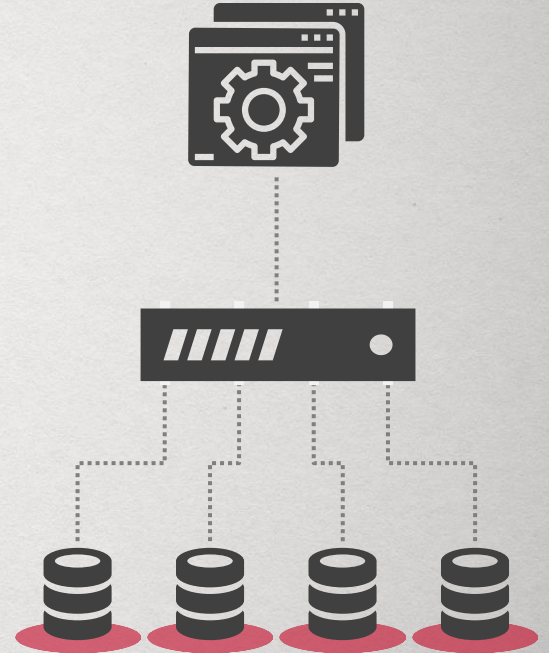
Our mistake was to not include **shared-disk** DBMSs.



Middleware

Transparent data sharding and query redirecting over cluster of single-node DBMSs.

Usually support MySQL or PostgreSQL wire protocol.



Middleware

Transparent data sharding and query redirecting over cluster of single-node DBMSs.

Usually support MySQL or PostgreSQL wire protocol.

The logo for ScaleBase, featuring the word "Scale" in blue and "Base" in grey.The logo for ScaleArc, featuring a colorful circular icon with red, blue, and yellow segments, followed by the word "Scale" in black and "Arc" in grey.The logo for codeFutures, featuring a red square icon followed by the word "code" in black and "Futures" in red.The logo for citusdata, featuring a green circular icon with a white swirl, followed by the word "citus" in green and "data" in blue.The logo for continuent, featuring a blue stylized bird or leaf icon above the word "continuent" in blue.

Database-as-a-Service

Distributed architecture designed specifically for cloud-native deployment.

Most of them use MySQL for single-node storage.

The logo for Xeround, featuring the word "xeround" in a lowercase, sans-serif font. The "x" is red, and the rest of the letters are dark grey.The logo for ClearDB, featuring a blue square icon with a white "B" inside, followed by the word "CLEARDB" in a blue, uppercase, sans-serif font.The logo for GenieDB, featuring a stylized icon of three horizontal bars in green, blue, and yellow, followed by the word "GenieDB" in a bold, black, sans-serif font.The logo for TransAttice, featuring the word "TRANS" in black and "ATTICE" in red, both in a bold, uppercase, sans-serif font, separated by a vertical slash.The logo for FathomDB, featuring a blue circular icon with a white stylized "F" inside, followed by the word "FATHOMDB" in a blue, uppercase, sans-serif font.

Why Not MySQL Engines?

the 451 group

451 TECHDEALMAKER

4 April 2011 - Sector IQ

How will the database incumbents respond to NoSQL and NewSQL?

Analyst: Matt Aslett

The acquisition of **MySQL AB** by **Sun Microsystems** in January 2008 appeared to signal that open source databases were on the brink of opening up a new battleground against the proprietary database giants. In announcing the deal, Sun signaled its intention to provide the support and development resources required for MySQL to challenge the established vendors in supporting mission-critical, high-performance applications on Web-based architectures. Needless to say, reality was somewhat different as Sun faced wider problems of its own and eventually succumbed to takeover by **Oracle** (Nasdaq: ORCL) in April 2009, in doing so handing ownership of the leading commercial open source database to the database heavyweight.

We had previously argued that MySQL was very much the crown jewel of the open source database world thanks to its focus on Web applications, its lightweight architecture and its fast read capabilities, which made it potentially complementary technology for all of the established database players. Additionally, if Oracle's major rivals were seeking an obvious alternative to MySQL in 2009, they were out of luck.

Just two years later, however, the database market is awash with open source databases with lightweight architectures targeted at Web applications. Not only have the likes of **Monty Program** and **SkySQL** emerged to provide alternative support for MySQL and its forks, but there are also a large number of products available under the banner of NoSQL, which emerged in mid-2009 as an umbrella term for a loosely affiliated collection of non-relational database projects. We have also seen the emergence of what we have termed 'NewSQL' database offerings, with companies promising to deliver the scalability and flexibility promised by NoSQL while retaining the support for SQL queries and/or ACID (atomicity, consistency, isolation and durability), or to improve performance for appropriate workloads (many of which are also supported by some NoSQL databases becomes

Tokutek®

deap® db

SCHOONER
INFORMATION TECHNOLOGY



ScaleDB

memSQL 



RethinkDB

Why Not MySQL Engines?

InnoDB is an excellent OLTP engine for single-node MySQL instances.

Nobody has built a long-term successful business replacing it.

Tokutek[®]

deap[®] db

SCHOONER
INFORMATION TECHNOLOGY 





What Went
WRONG?

What Went Wrong?

Almost every NewSQL company from the last decade has closed, sold for scraps, or pivoted to other markets.



Selling an OLTP DBMS is Hard

Existing Application:

- ▶ *People are risk adverse in replacing an OLTP DBMS even if it is slow or expensive.*

Greenfield Application:

- ▶ *The engineering start-up "cost" of a relational DBMS is higher than shoving JSON into a NoSQL DBMS.*



Existing DBMSs Are Really Good

The two most popular open-source DBMSs got even better in the last decade.

Most new applications don't have any data, so a single node DBMS is good enough.



Cloud Disruption

Most NewSQL companies started with selling on-prem and missed shift to cloud.

Difficult to compete with major cloud vendors on cost and technology.



amazon



Microsoft



Google

Lack of Open-Source

Few of the NewSQL DBMSs were open-source.

This may have inhibited their adoption, especially with developers building new applications.

VOLTDB

TRANSATTICE

citusdata





The AFTERMATH

Distributed SQL

New vendors are promising the same benefits of earlier NewSQL systems and seeing better adoption.

Many of their core concepts are similar to earlier systems.



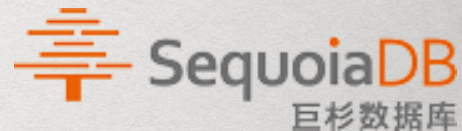
"Not Only SQL"

The vanguard NoSQL systems that touted their lack of SQL, joins, and transactions now include these features.



Remnants of NewSQL

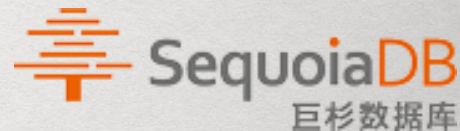
As of 2021, the term NewSQL seems to be only used by Chinese start-up database companies.



泽拓科技 昆仑数据库
Kunlun Database

Remnants of NewSQL

As of 2021, the term NewSQL seems to be only used by



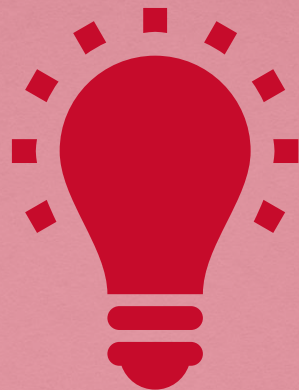
泽拓科技 昆仑数据库
Kunlun Database

泽拓科技 昆仑数据库 Kunlun Database

高性能 NewSQL OLTP 水平弹性扩容 容灾 高可用 强一致 分布式事务处理 分布式查询处理 Highly Performant, Crash Safe, Strongly Consistent,
Highly Available, Highly Scalable, Distributed NewSQL, BD RMS

首页 公司

RadonDB is a new generation of distributed relational database based on MySQL, we call it **MyNewSQL**. It was designed to create a database that capable to satisfy the requirement of large-scale transaction workload with high availability and reliability. RadonDB is architected to two independent cluster layers: SQL Layer and Transaction Layer, and the following guide show the detail of the inner-workings:



**What Comes
NEXT?**

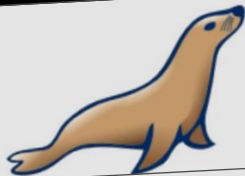
Random Prognostications

an evening with Andy Pavlo



- Fracturing will hurt perception & adoption.
 - See CouchDB vs. Couchbase.
 - This may be intentional.

16



- Fracturing will hurt p adoption.
 - See CouchDB vs. Couch
 - This may be intentional



mongoDB

- Will become first choice in new start-ups & Web apps.
 - Flush with cash, with more on the way.
 - MySQL-like growing pains.

The Next 10 Years

It will be difficult to supplant existing OLTP DBMSs unless there is another major hardware transition.

It will also be difficult to upend major cloud vendors on pricing unless...

The Next 10 Years

You still need humans to design, configure, and optimize logical/physical aspects of a database.

Humans are expensive.

Automation is the future.

ORACLE®

 Microsoft

 **OTTERTUNE**

Conclusion

NewSQL is dead.

From an academic view, the NewSQL movement was a success.

From a business view, it was a failure for those that embraced the NewSQL mantle.

END

@ANDY_PAVLO