Why Migrate from MySQL to Cassandra?

Table of Contents

Abstract

For 15+ years, Oracle's MySQL has been a de facto infrastructure piece in web applications, enjoying wide adoption. This is for good reason: MySQL provides a solid relational database that enables companies to build systems that perform well in many use cases. Yet, even its strongest supporters admit that it is not architected to tackle the new wave of big data applications. Modern businesses that need to manage big data use cases are helping to forge a different set of technologies to replace MySQL. This paper examines the why's and how's of migrating from Oracle's MySQL to these new big data technologies.

Introduction

Founded in 1995, MySQL has been one of the de facto infrastructure pieces in applications that target the Web. The database component of the Internet LAMP stack, MySQL has enjoyed wide success in adoption. This is for good reason: MySQL provides solid relational database management system (RDBMS) capabilities in an open source package that enables companies to build systems that perform well from a database perspective in many general purpose use cases.

MySQL was acquired by Sun Microsystems in 2008, and was afterwards formally acquired by Oracle (via its acquisition of Sun Microsystems) in January 2010. Now part of Oracle's stable of database products, MySQL continues to be promoted and sold through Oracle. In addition to Oracle, a number of other vendors have either forked MySQL to create another database offering or are using MySQL as part of a specialized service offering. Examples include Percona, Monty Program AB, Infobright, Calpont, and Amazon RDS.

While Oracle's MySQL remains a good RDBMS that performs well for the use cases it was designed for, even its strongest supporters admit that it is not architected to tackle the new wave of big data applications being developed today. In fact, in the same way that the needs of late 20th century web companies helped give birth to MySQL and drive its success, modern businesses today that need to manage big data use cases are helping to forge a different set of technologies that are replacing MySQL in many situations.

This paper examines the why's and how's of migrating from Oracle's MySQL to these new big data technologies, such as Apache Cassandra™ and Apache Hadoop™. It also looks at the benefits of moving from MySQL to a fully integrated data stack that combines those technologies and others together into a single big data platform like that found in DataStax Enterprise.

Why Stay with MySQL

Before continuing with a discussion on why it might be necessary to either develop new applications or migrate existing MySQL systems to another database platform, it first makes sense to understand why staying with MySQL as a database service may be the right choice. Because database migrations in particular can be resource-intensive, an IT professional should ensure they are making the right decision before they make such a move.

- ACID-compliant transactions, with nested transactions, commits/rollbacks, and full referential integrity required.
- A very denormalized data model that is well served by the Codd-Date relational design, and one where join operations cannot be avoided.
- Data is primarily structured with little to no unstructured or semi-structured data being present.
- Low to moderate data volumes that can be handled easily by the MySQL optimizer.
- Telco applications that require use of main memory solutions and whose data is primarily accessed via primary keys.
- Scale out architectures that are primarily read in nature, with no need to write to multiple masters or servers that exist in many different cloud zones or geographies.
- No requirement for a single database/cluster to span many different data centers.

- High availability requirements can be accomplished via a synchronous replication architecture that is primarily maintained at a single data center.

If an application presents these and similar requirements, then Oracle's MySQL may indeed be a good fit as a database platform.

Why Migrate from MySQL

One primary reason why various IT organizations are either already migrating away from Oracle's MySQL or planning to do so is the very visible rise of big data applications – and specifically, big data online transaction processing (OLTP) applications. These companies either have existing systems morphing into big data systems, or they are planning new applications that are big data in nature and need something "more" than MySQL for their database platform.

Although the top industry IT analyst groups may disagree on various technical and marketplace trends, they are, remarkably, in agreement on the following three things: (1) the upsurge of big data applications; (2) the definition of what constitutes big data, and (3) the need for new technology to deal with big data.

The momentum and growth of big data applications is unmistakable. Underscoring this is a recent survey of 600 IT professionals that revealed nearly 70 percent of organizations are now considering, planning, or running big data projects.[1]

As for a definition of big data, it is nearly universally agreed that big data involves one or all of the following:

1. Velocity – data coming in at extreme rates of speed.
2. Variety – the types of data needing to be captured (structured, semi-structured, and unstructured).
3. Volume – sizes that potentially involve terabytes to petabytes of data.
4. Complexity – involves everything from moving operational data into big data platforms, maintaining various data "silos," and the difficulty in managing data across multiple sites and geographies.

Although they may phrase it differently, top analyst groups and market observers agree that, to tackle big data applications that have the above characteristics, something more than standard RDBMSs like Oracle's MySQL is needed.

For example, David Kellogg says big data is "too big to be reasonably handled by current/traditional technologies."[2] Consulting and research firm McKinsey & Company agrees with Kellogg's concept of big data and defines it as "datasets whose size is beyond the ability of typical database software tools to capture, store, manage, and analyze."[3]

IDC says: "Big data technologies describe a new generation of technologies and architectures, designed to economically extract value from very large volumes of a wide variety of data, by enabling high-velocity capture, discovery, and/or analysis."[4]

And finally, O'Reilly defines big data in this way: "Big data is data that exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or doesn't fit the strictures of your database architectures. To gain value from this data, you must choose an alternative way to process it."[5]

---

[1] "Survey: 70 Percent of Organizations Have Big Plans for Big Data," by Pedro Hernandez, EntepriseAppsToday.com, May 14, 2012: http://www.enterpriseappstoday.com/data-management/survey-70-perdent-enterprises-big-plans-for-big-data.html.

[2] "'Big data' has jumped the shark," DBMS2, September 11, 2011: http://www.dbms2.com/2011/09/11/big-data-has-jumped-the-shark/.

[3] Big Data: The next frontier for innovation, competition, and productivity, McKinsey Global Institute, May 2011, p. 11: http://www.mckinsey.com/Insights/MGI/Research/Technology_and_Innovation/Big_data_The_next_frontier_for_innovation.

[4] Extracting Value from Chaos, by John Gantz and David Reinsel, IDC, June 2011: http://idcdocserv.com/1142.

[5] "What Is Big Data? An Introduction to the Big Data Landscape," by Edd Dumbill, O'Reilly Radar, January 11, 2012: http://radar.oreilly.com/2012/01/what-is-big-data.html.

In particular, big data OLTP applications are nudging MySQL aside for other options. While MySQL initially gained its popularity through use of the MyISAM storage engine, the InnoDB transactional engine is arguably the most used today. But InnoDB isn't designed to handle the types of big data requirements discussed above.

What types of limitations, bottlenecks, and issues are MySQL users experiencing? Although the exact situations vary, a few of the most prevalent reasons that cause a move from MySQL are as follows:

Architectural Limitations

One reason modern businesses are switching from Oracle's MySQL to big data platforms is because the underlying architecture does not support key big data use cases. This is true regardless of which MySQL products are being considered – MySQL Community/Enterprise, MySQL Cluster, or database services such as Amazon RDS.

Some of the architectural issues that arise when MySQL is thrown into big data situations include:

- The traditional master-slave architecture of MySQL (one write master with 1-n slaves) prohibits "location independent" or "read/write anywhere" use cases that are very common in big data environments where a database cluster is spread out throughout many different geographies and data centers (and the cloud), with each node needing to support both reads and writes.
- The necessity to manually shard (i.e., partition) general MySQL systems to overcome various performance shortcomings becomes a very time-consuming, error-prone, and expensive proposition to support. It also places a heavy burden on development staff to support sharding logic in the application.
- Failover and failback situations tend to require manual intervention with generally replicated MySQL systems. Failback can be especially challenging.
- Although it provides automatic sharding and supports simple geographic replication, MySQL Cluster's dependence on synchronous replication can cause latency and transactional response time issues. Further, its geographic replication does not support multiple (i.e., >2) data centers in a way that either performs well or is easy to manage.
- Database services such as Amazon's RDS suffer from the same shortfalls above, as Amazon only supports either a simple standby server that is maintained in a different availability zone in Amazon's cloud, or a series of read replicas that are provisioned and used to help service increased query (not write) traffic.

Data Model Limitations

A big reason why many businesses are moving to NoSQL-based solutions is because the legacy RDBMS data model is not flexible enough to handle big data use cases that contain a mixture of structured, semi-structured, and unstructured data. While MySQL has good datatype support for traditional RDBMS situations that deal with structured data, it lacks the dynamic data model necessary to tackle high-velocity data coming in from machine-generated systems or time series applications, as well as cases needing to manage semi-structured and unstructured data.

Recently, Oracle announced it had introduced a NoSQL-type interface into its MySQL Cluster product that is key/value in design. While certainly helpful in some situations, such a design still falls short in key big data use cases like time series applications that require inserting data into structures that support tens of thousands of columns.

Scalability and Performance Limitations

Oracle's MySQL has long been touted as a scale-out database. However, those who know and use MySQL admit it has limitations that negate its use in big data situations where scalability is required. For example:

- More servers can be added to a general MySQL Community/Enterprise cluster to help service more reads, but writes are still bottlenecked via the main write master server. Moreover, if many read slave servers are required, latency issues can arise in the process of simply getting the data from the master server to all the slaves.

- Consumption of high-velocity data can be challenging, especially if the InnoDB storage engine is used, as the index-organized structure often does not handle high insert rates well. Third-party storage engine vendors, which are columnar in nature, typically cannot help in this case, as they rely on their proprietary high-speed loaders to load data quickly into a database.
- Data volumes over half a terabyte become a real challenge for the MySQL optimizer. To overcome this, a third-party storage engine vendor such as Calpont or Infobright must be used – but these vendors have limitations either in their SQL support, MPP capabilities, or both.

Why Migrate from MySQL

While a move from Oracle's MySQL may be necessary because of its inability to handle key big data use cases, why should that move involve a switch to Apache Cassandra and DataStax Enterprise?

The sections that follow describe why a move to Cassandra and DataStax Enterprise make both technical and business sense for MySQL users seeking alternatives.

A Technical Overview of Cassandra

Apache Cassandra, an Apache Software Foundation project, is an open source NoSQL distributed database management system. Cassandra is designed to handle big data OLTP workloads across multiple data centers with no single point of failure, providing enterprises with continuous availability without compromising performance.

In selecting an alternative to Oracle's MySQL, IT professionals will find Apache Cassandra is a standout among other NoSQL offerings for the following technical reasons:

- **Massively scalable architecture** – Cassandra's masterless, peer-to-peer architecture overcomes the limitations of master-slave designs and allows for both high availability and massive salability. Cassandra is the acknowledged NoSQL leader when it comes to comfortably scaling to terabytes or petabytes of data, while maintaining industry-leading write and read performance.
- **Linear scale performance** – Nodes added to a Cassandra cluster (all done online) increase the throughput of a database in a predictable, linear fashion for both read and write operations, even in the cloud where such predictability can be difficult to ensure.
- **Continuous availability** – Data is replicated to multiple nodes in a Cassandra database cluster to protect from loss during node failure and provide continuous availability with no downtime.
- **Transparent fault detection and recovery** – Cassandra clusters can grow into the hundreds or thousands of nodes. Because Cassandra was designed for commodity servers, machine failure is expected. Cassandra utilizes gossip protocols to detect machine failure and recover when a machine is brought back into the cluster – all without the application noticing.
- **Flexible, dynamic schema data modeling** – Cassandra offers the organization of a traditional RDBMS table layout combined with the flexibility and power of no stringent structure requirements. This allows data to be dynamically stored as needed without performance penalty for changes that occur. In addition, Cassandra can store structured, semi-structured, and unstructured data.
- **Guaranteed data safety** – Cassandra far exceeds other systems on write performance due to its append-only commit log while always ensuring durability. Users must no longer trade off durability to keep up with immense write streams. Data is absolutely safe in Cassandra; data loss is not possible.
- **Distributed, location independence design** – Cassandra's architecture avoids the hot spots and read/write issues found in master-slave designs. Users can have a highly distributed database (e.g., multi-geography, multi-data center) and read or write to any node in a cluster without concern over what node is being accessed.
- **Tunable data consistency** – Cassandra offers flexible data consistency on a cluster, data center, or individual I/O operation basis. Very strong or eventual data consistency among all participating nodes can be set globally and also controlled on a per-operation basis (e.g., per INSERT, per UPDATE)
- **Multi-data center replication** – Whether it's keeping data in multiple locations for disaster recovery scenarios or locating data physically near its end users for fast performance, Cassandra offers support for multiple data centers. Administrators simply configure how many copies of the data they want in each data center, and Cassandra handles the rest – replicating the data automatically. Cassandra is also rack-

aware and can keep replicas of data stored on different physical racks, which helps ensure uptime in the case of single rack failures.

- **Cloud-enabled** – Cassandra's architecture maximizes the benefits of running in the cloud. Also, Cassandra allows for hybrid data distribution where some data can be kept on-premise and some in the cloud.
- **Data compression** – Cassandra supplies built-in data compression, with up to an 80 percent reduction in raw data footprint. More importantly, Cassandra's compression results in no performance penalty, with some use cases showing actual read/write operations speeding up due to less physical I/O being required.
- **CQL (Cassandra Query Language)** – Cassandra provides a SQL-like language called CQL that mirrors SQL's DDL, DML, and SELECT syntax. CQL greatly decreases the learning curve for those coming from RDBMS systems because they can use familiar syntax for all object creation and data access operations.
- **No caching layer required** – Cassandra offers caching on each of its nodes. Coupled with Cassandra's scalability characteristics, nodes can be incrementally added to the cluster to keep as much data in memory as needed. The result is that there is no need for a separate caching layer.
- **No special hardware needed** – Cassandra runs on commodity machines and requires no expensive or special hardware.
- **Incremental and elastic expansion** – The Cassandra ring allows online node additions. Because of Cassandra's fully distributed architecture, every node type is the same, which means clusters can grow as needed without any complex architecture decisions.
- **Simple install and setup** – Cassandra can be downloaded and installed in minutes, even for multi-cluster installs.
- **Ready for developers** – Cassandra has drivers and client libraries for all the popular development languages (e.g., Java, Python)

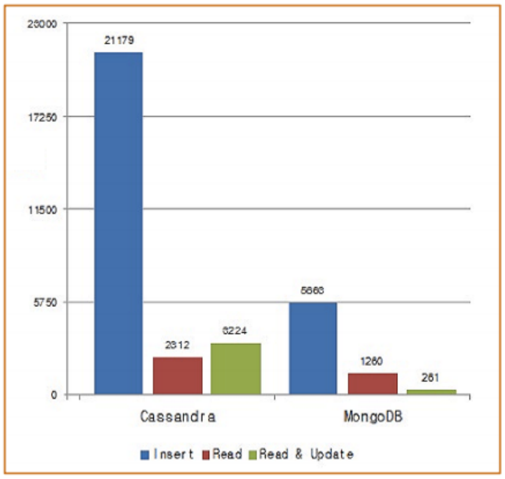
Given these technical features and benefits, the following are typical big data use cases handled well by Cassandra in the enterprise:

- Big data OLTP situations
- Time series data management
- High-velocity device data ingestion and analysis
- Healthcare system input and analysis
- Media streaming management (e.g., music, movies)
- Social media (i.e., unstructured data) input and analysis
- Online web retail (e.g., shopping carts, user transactions)
- Real-time data analytics
- Online gaming (e.g., real-time messaging)
- Software as a Service (SaaS) applications that utilize web services
- Write-intensive systems

Cassandra vs. Other NoSQL Solutions

What does the performance of Cassandra look like compared to other NoSQL options? While each use case is different, external benchmarks such as the YCSB test[6] show Cassandra to outperform its rivals in a number of situations.

This particular benchmark (right) shows Cassandra delivering nearly 4x the write performance, 2x the read performance, and better than 12x overall performance in a mixed workload use case over another leading NoSQL provider.



6 "NoSQL Benchmarking," CUBRID: http://blog.cubrid.org/dev-platform/nosql-benchmarking/?utm_source=NoSQL+Weekly+List&utm_campaign=143fae86b2-NoSQL_Weekly_Issue_41_September_8_2011&utm_medium=email.

Who's Using Cassandra?

One benefit MySQL users have enjoyed is a large community of users who have deployed the database in many production environments. Cassandra, likewise, is used in many industries for modern applications that need scale, fast performance, data flexibility, and easy data distribution. Below is a snapshot of some of the companies and organizations that use Cassandra in production. (Note: There are many other household name companies with production implementations that cannot be published due to NDA restrictions.)

A Quick Look at DataStax

A viable company behind an open source offering is vital for enterprises wanting to enjoy the benefits provided by open source software, but also needing the professional requirements supplied by proprietary software offerings.

DataStax is the leading provider of modern enterprise database software products and services based on Apache Cassandra. It employs the Apache chair of Cassandra as well as most of the project's committers. At the time of this writing, DataStax has nearly 400 employees and over 500 customers, though both of these statistics are growing rapidly.

DataStax provides free and open source NoSQL products as well as commercial solutions aimed at production big data environments, with its flagship solution being DataStax Enterprise.

DataStax Enterprise – The Choice for Production Big Data Deployments

Apache Cassandra can be likened to MySQL's community server in that both are free and open source. By contrast, DataStax Enterprise is more akin to MySQL Enterprise or MySQL Cluster's Carrier Grade editions in that it is designed for production deployments that power key business systems.

DataStax Enterprise is tailor-made to manage big data effectively. The solution inherits all of Cassandra's powerful feature set for servicing modern big data OLTP applications, and smartly integrates a fault-tolerant, analytics platform that provides Hadoop™ MapReduce, Hive, Pig, Mahout, and Sqoop support for business intelligence systems. It also includes enterprise search capabilities via Apache Solr™, which is the most popular open source search software in use today.

A key differentiator of DataStax Enterprise over other big data providers is that real-time, analytic, and search workloads are intelligently isolated across a distributed DataStax Enterprise database cluster, so that no competition for underlying compute resources or data occurs.

DataStax Enterprise is comprised of three components:

- **The DataStax Enterprise Server** – built on Apache Cassandra, the server manages real-time data with Cassandra, analytic data with Hadoop, and enterprise search data with Apache Solr.
- **OpsCenter Enterprise** – a visual, browser-based solution for managing and monitoring Cassandra and the DataStax Enterprise server.
- **Production Support** – full 24x7x365 support from the big data experts at DataStax.

With Hadoop and Solr, the types of use cases that can be tackled with DataStax Enterprise grow exponentially beyond those previously covered with Cassandra alone and include:

- Social media input and analysis
- Web clickstream analysis
- Buyer event and behavior analytics
- Fraud detection and analysis
- Risk analysis and management
- Supply chain analytics
- Web product searches
- Internal document search (e.g., law firms)
- Real estate/property searches
- Social media matchups
- Web and application log management/analysis

What About Cost?

There are many technical benefits in moving from Oracle's MySQL to Cassandra and DataStax Enterprise, but what about cost? How does DataStax compare with Oracle in that regard? As of May 2012, the list prices for Oracle's MySQL products were priced by subscription, per server/socket, and were as follows[7]:

Product	**List Price**
MySQL Standard Edition Subscription (1-4 socket server)	$2,000
MySQL Standard Edition Subscription (5+ socket server)	$4,000
MySQL Enterprise Edition Subscription (1-4 socket server)	$5,000
MySQL Enterprise Edition Subscription (5+ socket server)	$10,000
MySQL Cluster Carrier Grade Edition Subscription (1-4 socket server)	$10,000
MySQL Cluster Carrier Grade Edition Subscription (5+ socket server)	$20,000

For feature differences in the MySQL products listed above, see http://www.mysql.com/products/.

Like the MySQL Community Edition, DataStax provides a free edition of Apache Cassandra (the DataStax Community Edition), which comes with the latest version of Cassandra, a free version of DataStax's OpsCenter management and monitoring tool, and quick-start developer aids.

[7] "MySQL Global Price List, Software Investment Guide," Oracle, November 1, 2010: http://www.oracle.com/us/corporate/pricing/price-lists/mysql-pricelist-183985.pdf

From a production, enterprise perspective, a subscription to DataStax Enterprise Edition may be compared to the MySQL Enterprise and MySQL Cluster Carrier Grade Edition subscriptions. DataStax does not currently price by socket – only by server, with pricing either being on par with MySQL Enterprise for small boxes or, with beefier machines, substantially less (e.g., 50 to 70 percent).

How to Migrate from MySQL to Cassandra

The first step in migrating from MySQL to Cassandra is to understand that data modeling is handled differently in NoSQL solutions vs. RDBMSs. In traditional databases such as MySQL, data is modeled in standard "third normal form" design without the need to know what questions will be asked of the data. By contrast, in NoSQL, the questions asked of the data are what drive the data model design and the data is highly denormalized.

If a developer is simply interested in porting MySQL schema and data over into a Cassandra keyspace (analogous to a database in MySQL) just for testing or preliminary development purposes, there are two primary options:

1. Use the Sqoop interface to move MySQL tables and data.
2. Use ETL (extract-transform-load) tools such as Pentaho's Kettle to move schema and data.

Using Sqoop to Migrate from MySQL

DataStax Enterprise supports Sqoop, which is a utility designed to transfer data between an RDBMS and Hadoop. Given that DataStax Enterprise combines Cassandra, Hadoop, and Solr together into one big data platform, a developer can move data not only to a Hadoop system with Sqoop, but also Cassandra.

The DataStax Enterprise installation package includes a sample/demo of how to move MySQL schema and data into Cassandra. Because Sqoop works via Java Database Connectivity (JDBC), the only prerequisite is that the JDBC driver for MySQL must be downloaded from the MySQL website and placed in a directory where Sqoop has access to it (the /sqoop subdirectory of the main DataStax Enterprise installation is recommended).

Each MySQL table is mapped to a Cassandra column family. Column families are a Google Bigtable structure, with rows and columns like MySQL but much more dynamic and flexible. The migration is done via a command line utility that accepts a number of different parameters. For example, the following code migrates a MySQL table contained on a server with IP address `127.0.0.1` that's in the dev database. It connects to MySQL with the `root` ID and uses no password. It migrates a table called `npa_nxx` from MySQL into a Cassandra keyspace named dev, to a column family named `npa_nxx_cf`, identifies the MySQL column `npa_nxx_key` as the primary key, names the Cassandra server's host IP, and lastly, asks for the schema to be created before the data is imported:

```
./dse sqoop import --connect jdbc:mysql://127.0.0.1/dev \
         --username root                               \
         --table npa_nxx                               \
         --cassandra-keyspace dev                      \
         --cassandra-column-family npa_nxx_cf          \
         --cassandra-row-key npa_nxx_key               \
         --cassandra-thrift-host 127.0.0.1             \
         --cassandra-create-schema
```

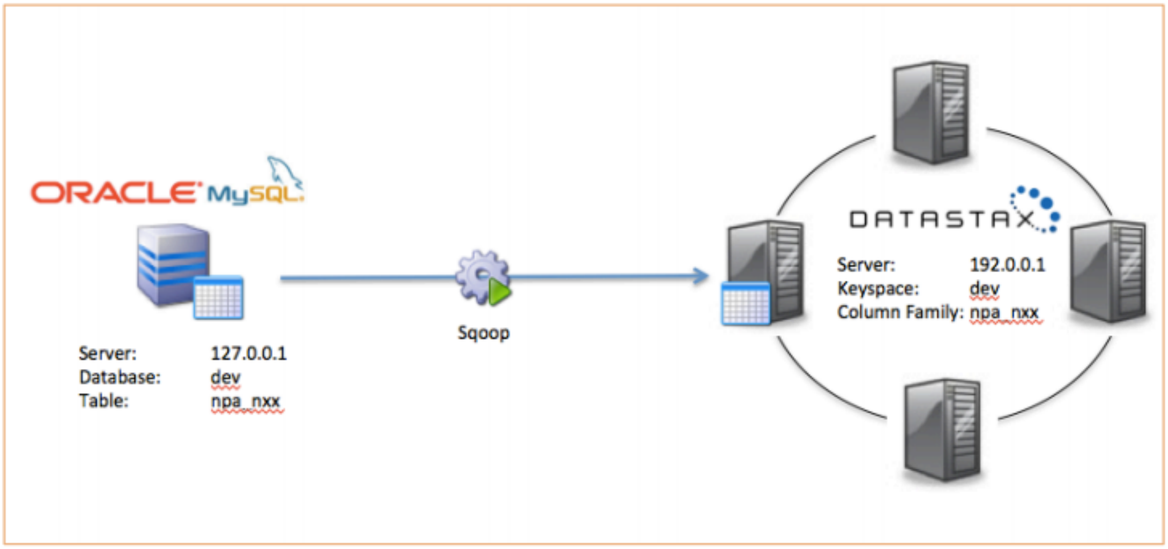


Figure 1 - Using Sqoop to move data from MySQL to DataStax Enterprise

Using Pentaho Kettle to Migrate from MySQL

Another way to migrate MySQL tables and data to Cassandra is by using a number of ETL tools on the market such as Pentaho's Data Integration product, also known as Kettle. Pentaho makes two editions of their ETL tool available: a free community edition and a paid enterprise edition. For core ETL tasks such as moving MySQL schema and data to Cassandra, the community edition should provide everything that is needed.

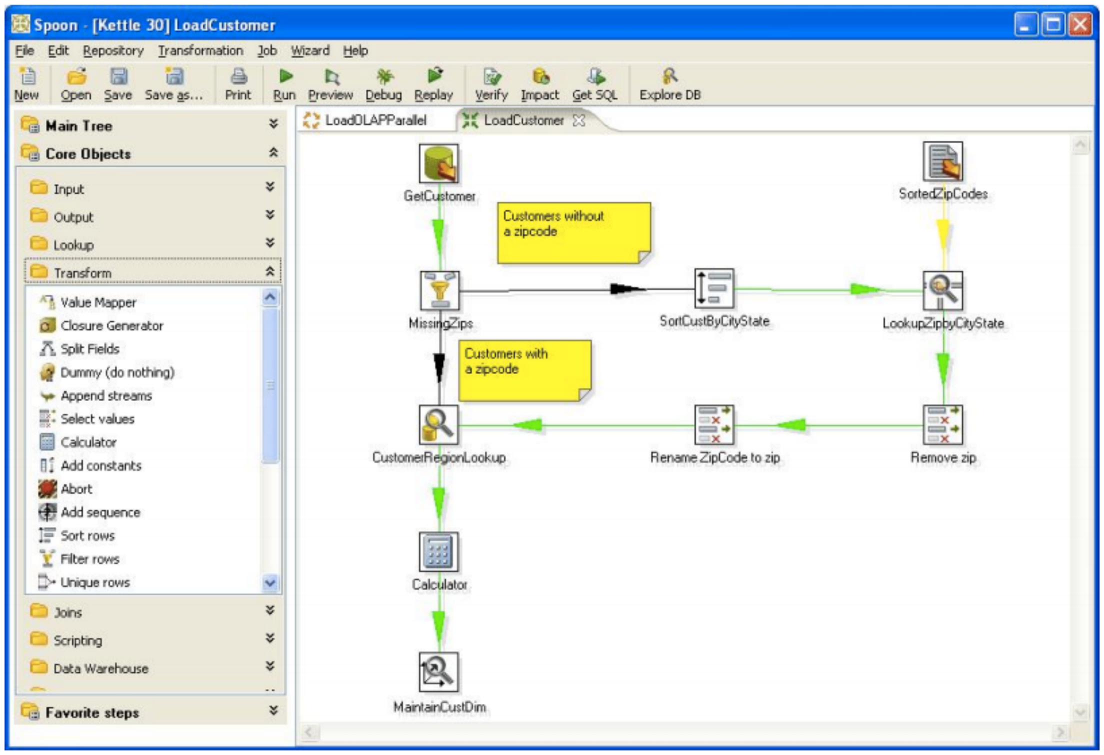


Figure 2 - Kettle's visual interface for performing ETL operations

Pentaho's Kettle product provides an easy-to-use graphical user interface (GUI) that allows a developer to visually design their MySQL migration tasks. Unlike the Sqoop utility, which just does extract-load, Pentaho's product allows a developer to create simple to sophisticated transformation routines to customize how a MySQL schema and data are moved to Cassandra. In addition, the data movement engine of Kettle is quite efficient, so medium to semi-large data volumes can be moved in a high-performance manner.

More information about Kettle and free downloads can be found at: http://kettle.pentaho.com/.

Examples of Customers Who Have Switched from MySQL

Whether it's by using Sqoop, ETL tools, or other in-house-developed options, a move from Oracle's MySQL to big data platforms like Cassandra and DataStax Enterprise is not difficult at all. DataStax has many customers who have made just such a switch, but do not discuss it from an external-facing standpoint. However, the following are a few customers who were kind enough to let us share their use cases publicly.

Mahalo

Mahalo is a social media learning business that has a top 200 web ranking and experiences 12 million visits per month. Mahalo needed a database provider to manage the activity log for every single customer interaction as well as store information for their Q&A topic database. Mahalo started using Oracle's MySQL. However, performance and availability issues necessitated a move to a database that could scale and handle their heavy write workload. Further, the company needed a more dynamic data model to store the variety of data that was coming in.

Mahalo chose Cassandra and DataStax as their MySQL replacement. Mahalo's chief technology officer (CTO), Jason Burch, says: "With the Cassandra conversion completed and running smoothly, we're now free to focus on our primary mission, knowing that we'll be able to deliver the excellent responsiveness and capabilities that our user community has come to expect."

Pantheon Systems

Pantheon Systems provides a cloud-based web development platform for websites made with Drupal. Pantheon needed a primary database platform that would hold all metadata information that supports their main application platform and also all of their media storage.

Pantheon began with MySQL, but found it was unable to handle its requirements of managing both structured and unstructured data. Moreover, MySQL could not scale or support Pantheon's need for continuous availability across multiple data centers.

Pantheon switched to Cassandra and DataStax, which met all of their specific requirements. David Strauss, Pantheon's CTO, describes the company's use of Cassandra this way: "All the actual platform data in Pantheon is persisted primarily to Cassandra. We could wipe out pretty much everything on Pantheon, but as long as the Cassandra store is there, we have our data."

Conclusion

There is no argument that Oracle's MySQL is a good RDBMS – and one that well serves the use cases for which it was originally designed. But for IT professionals who are either planning new big data applications or have existing MySQL systems that have begun to break down under big data workloads, a move to DataStax Enterprise and Cassandra makes both business and technical sense.

Switching to a modern, big data platform like DataStax Enterprise will future-proof any application, and provides confidence that the system will scale and perform well both now and into a demanding future.

For more information on DataStax Enterprise and Cassandra, visit www.datastax.com.

About DataStax

DataStax is the fastest, most scalable distributed database technology, delivering Apache Cassandra to the world's most innovative enterprises. DataStax is built to be agile, always-on, and predictably scalable to any size.

With more than 500 customers in 45 countries, DataStax is the database technology and transactional back- bone of choice for the worlds most innovative companies such as Netflix, Adobe, Intuit, and eBay. Based in San- ta Clara, Calif., DataStax is backed by industry-leading investors including Lightspeed Venture Partners, Meritech Capital, and Crosslink Capital. For more information, visit DataStax.com or follow us @DataStax. © 2014 DataStax, All Rights Reserved.

Appendix A – FAQ on Switching from MySQL to DataStax Enterprise/Cassandra

This appendix supplies answers to frequently asked questions about migrating from Oracle's MySQL to DataStax Enterprise/Cassandra.

Do I lose transaction support when moving from MySQL to Cassandra?

MySQL's InnoDB supplies ACID transaction support, whereas Cassandra provides AID transaction support. The "C" or consistency part of transaction support does not apply to Cassandra, as there is no concept of referential integrity or foreign keys in a NoSQL database. There is also no concept of commit/rollback in Cassandra.

Batch operations are supported in Cassandra via the BATCH option in CQL.

What type of data consistency does DataStax Enterprise/Cassandra support?

DataStax Enterprise and Cassandra support "tunable data consistency". This type of consistency is the kind represented by the "C" in the CAP theorem, which concerns distributed systems.

Cassandra extends the concept of "eventual consistency" in NoSQL databases by offering tunable consistency. For any given read or write operation, the client application decides how consistent the requested data should be.

Consistency levels in Cassandra can be set on any read or write query. This allows application developers to tune consistency on a per-query/operation basis depending on their requirements for response time versus data accuracy. Cassandra offers a number of consistency levels for both reads and writes.

What parts of my MySQL database cannot be migrated to DataStax Enterprise/Cassandra?

Schema, data, and general indexes may be migrated, but objects that currently cannot be migrated include:

- Stored procedures
- Views
- Triggers
- Functions
- Security privileges
- Referential integrity constraints
- Rules
- Partitioned table definitions

Do I need to use a MySQL caching layer (like memcached) with Cassandra?

No. Cassandra negates the need for extra software caching layers like memcached through its distributed architecture, fast write throughput capabilities, and internal memory caching structures. When you want more memory cache available to your cluster, you simply add more nodes and it will handle the rest for you.

Is data absolutely safe in Cassandra?

Yes. First, data durability is fully supported in Cassandra, so any data written to a database cluster is first written to a commit log in the same fashion as nearly every popular RDBMS does.

Second, Cassandra offers tunable data consistency. This means a developer or administrator can choose how strong they wish consistency across nodes to be. The strongest form of consistency is to mandate that any data modifications be made to all nodes, with any unsuccessful attempt on a node resulting in a failed data operation. Cassandra provides consistency in the CAP sense, in that all readers will see the same values.

How is data written and stored in Cassandra?

Cassandra has been architected for consuming large amounts of data as fast as possible. To accomplish this, Cassandra first writes new data to a commit log to ensure it is safe. After that, the data is written to an in-memory structure called a memtable. Cassandra deems the write successful once it is stored on both the commit log and a memtable, which provides the durability required for mission-critical systems.

Once a memtable's memory limit is reached, all writes are then written to disk in the form of an SSTable (sorted strings table). An SSTable is immutable, meaning it is not written to ever again. If the data contained in the SSTable is modified, the data is written to Cassandra in an upsert fashion and the previous data automatically removed.

Because SSTables are immutable and only written once the corresponding memtable is full, Cassandra avoids random seeks and instead only performs sequential I/O in large batches, resulting in high write throughput.

A related factor is that Cassandra doesn't have to do a read as part of a write (i.e., check index to see where current data is). This means that insert performance remains high as data size grows, while with b-tree based engines (e.g., MongoDB) it deteriorates.

What kind of query language is provided in Cassandra? Is it like SQL in MySQL?

Cassandra supplies the Cassandra Query Language (CQL), which is very SQL-like. Queries are done via the standard `SELECT` command, while DML operations are accomplished via the familiar `INSERT`, `UPDATE`, `DELETE`, and `TRUNCATE` commands. DDL commands such as `CREATE` are used to create new keyspaces and column families.

Although CQL has many similarities to SQL, it does not change the underlying Cassandra data model. There is no support for JOINs, for example.

Appendix B – Comparing MySQL and DataStax Enterprise/Cassandra

This technical appendix provides brief comparisons between Oracle's MySQL and DataStax Enterprise/Cassandra.

General Comparisons

Feature/Function	MySQL	DataStax Enterprise/Cassandra
Platform support Data model	Linux, Windows, Solaris, Unix, Mac Relational/tabular	Linux, Windows, Mac Google Bigtable
Primary data object	Table	Column family
Data variety support	Primarily structured	Structured, semi-structured, unstructured
Data partitioning/sharding mode	Manual in general MySQL; automatic in MySQL Cluster	Automatic
Logical database container	Database	Keyspace
Indexes	Primary, secondary, clustered, full-text	Primary, secondary
Distribution architecture CAP consistency model Multi-data center support	Master/slave or synchronous replication with MySQL Cluster Synchronous with MySQL Cluster Basic	Peer-to-peer with replication Tunable consistency per operation Multi-data center and cloud, with rack awareness
Transaction support	ACID	AID (no "C" as there are no foreign keys)
Memory usage model	General caches, query cache, main memory option with MySQL Cluster	Distributed object/row caches across all nodes in a cluster
Core language	SQL	CQL
Primary query utilities	mysql command line client	CQL shell; CLI
Development language support Large data volume support (TBs+) Data compression Analytic support Search support	Many (e.g., Java, Python) Low TBs done with third-party storage engines Built into some storage engines Some analytic functions Full-text indexes	Many (e.g., Java, Python) Native, TB-PB support Built in Done via Hadoop (MapReduce, Hive, Pig, Mahout) Done via Solr integration
Geospatial support	Spatial extensions	Done via Solr integration
Logging (e.g., web, application) data support Mixed workload support	Nothing built in Must separate/ETL data between OLTP, analytic, search	Handled via log4j All handled in one cluster with built-in workload isolation
Backup/recovery Enterprise management/monitoring	Online, point-in-time restore MySQL Enterprise Monitor	Online, point-in-time restore DataStax OpsCenter

Datatype Comparisons

CQL datatype	MySQL datatype	Description
blob	blob	Arbitrary hexadecimal bytes (no validation)
ascii	char	US-ASCII character string
text, varchar	text, varchar	UTF-8 encoded string
varint	numeric	Arbitrary-precision integer
int, bigint	int, bigint	8-bytes long
uuid	None	Type 1 or type 4 UUID
timestamp	timestamp	Date plus time, encoded as 8 bytes since epoch
boolean	bit	true or false
float	float	4-byte floating point
double	double	8-byte floating point
decimal	decimal	Variable-precision decimal
counter	None	Distributed counter value (8-bytes long)