An Oracle White Paper
April 2014

Back to the Future with Oracle Database 12*c*

Introduction

The IT business evolves in circles with a tendency to form a spiral. Every once in a while an idea or a technology moves back into the focus, which is not entirely new, but has evolved since the first time it appeared.

In October 2013, Gartner published their latest Magic Quadrant (MQ) for Operational Database Management Systems, formerly OLTP [Gartner 2013]. The number of players in the lower left quadrant, the so-called "Niche Players", may be surprising and may indicate that the IT business is about to enter a new cycle introducing changes that could elevate and transform the Operational Database Management Systems market significantly. It may, however, just be a minor cycle leading to a broad but rather insignificant adjustment of the market.

Currently, mainly the increased number of Niche Players, amongst them names such as Couchbase, VoltDB and mongoDB, surprises. All Niche Players are well behind the "Leaders" considering their ability to execute and / or their vision according to Gartner. Current leaders are IBM, Microsoft, SAP, and Oracle, which leads on both axes (ability to execute and vision). Some of the players, mostly the Niche Players, only offer one specific product with a very explicit set of features, while the Leaders can typically leverage a universal solution. The latter is the key to success for those players and ensures the success of their customers.

This paper is intended for technically interested parties that are involved in the operation, management as well as selection of a standard database management system. The paper is split into two parts. Part one provides a practical review of some database concepts and relates them to the current DBMS market as well as the history of three exemplary database management systems. Part two looks into the reasons for the recent popularity of NoSQL databases in order to determine their potential impact on the current DBMS market. It discusses two exemplary NoSQL products and compares them to the solutions offered by the leading RDBMS vendor: Oracle.

How Database Concepts Influence the Database Market

In order to understand the rise of Niche Players in the Operational Database Management Systems market as well as the recent increase in popularity of NoSQL (*N*ot *O*nly or No-*S*tructured *Q*uery *L*anguage) databases, one will have to review certain database concepts and see how they influenced the current database market. The background and the focus of nearly all database management systems (DBMS) available today is related to the requirements and technology that were available at the time they were first invented. The following three examples of database management systems will be used to illustrate this relation in the course of this discussion:

1. Apache Cassandra
   - Webpage: http://cassandra.apache.org/
   - Cassandra is an open-source project under the Apache Software Foundation license.
   - DataStax, which is part of Gartner's MQ, offers professional services for Cassandra.
2. mongoDB
   - mongoDB is originally also an open-source database (see http://www.mongoDB.org/).
   - http://www.mongoDB.com / mongoDB – the company – offers professional services
3. Oracle RDBMS
   1. Oracle Relational Database Management System; current version: Oracle Database 12*c*

Apache Cassandra and mongoDB were chosen as examples for open-source database products. Both offerings have found support by companies providing professional services for the respective solution. Both database offerings also address "agility and scalability" as well as "fault tolerance, performance and elasticity" in their own way, attributes that are typically also associated with the Oracle Database.

Both solutions represent a very different type of database compared to the Oracle Database, which is an example of a relational database management system. Cassandra as well as mongoDB are non-ACID, NoSQL, key-value pair databases using a sharding approach for horizontal scalability. In other words, they are built on guiding principles that are the opposite of those guiding the Oracle RDBMS.

In "Cassandra's words": "On the contrary to the strong consistency used in most relational databases (ACID for *Atomicity Consistency Isolation Durability*) Cassandra is at the other end of the spectrum (BASE for *Basically Available Soft-state Eventual consistency*). Cassandra weak consistency comes in the form of eventual consistency which means the database eventually reaches a consistent state."[1]

[1] http://wiki.apache.org/cassandra/ArchitectureOverview

Database concepts and fundamentals – a practical review

In order to understand statements like the definition of "strong consistency" that Cassandra uses and their practical impact, one needs to first understand the differences in architecture and technology that are available and used for today's DBMS solutions. For the purpose of this paper, the following three concepts will be considered and discussed:

1. NoSQL databases vs. Relational Database Management Systems
2. The ACID consistency vs. the BASE consistency model
    a. The CAP theorem in the context of distributed database systems
3. Shared Disk Clusters vs. Shared Nothing Clusters
    a. Sharding as an evolution of the Shared Nothing approach

1) NoSQL databases vs. Relational Database Management Systems

According to Wikipedia, "a NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases."[2] Wikipedia also mentions that "NoSQL databases are often highly optimized key–value stores intended primarily for simple retrieval and appending operations, whereas an RDBMS is intended as a general purpose data store." In simple terms, this means, that the definition of NoSQL databases is based on the idea that a NoSQL database uses a different approach than the "tabular relations" used by RDBMS.

The association of NoSQL databases with key–value stores, is implied, but in the absence of a NoSQL standard not compelling. NoSQL databases can be implemented in many ways. http://nosql-database.org/ currently lists over 150 NoSQL DBMS in more than ten categories. Those categories include, but are not limited to: Wide Column Store / Column Families, Document Store, Key Value / Tuple Store, Graph Databases, Multimodel Databases, Object Databases and quite a few more.

In addition, Wikipedia mentions that "NoSQL systems are also referred to as "Not only SQL" to emphasize that they may in fact allow SQL-like query languages to be used." An example for such a DBMS is Cassandra, which provides CQL – the Cassandra Query Language.[3] On the other hand, an RDBMS can also provide a NoSQL-based way of accessing data. For example, VoltDB as well as the Oracle RDBMS both support "SQL and the flexibility of JSON"[4].

[2] Wikipedia, the free encyclopedia, 20th March 2014, *NoSQL*: http://en.wikipedia.org/wiki/NoSQL

[3] *The Cassandra Query Language*: http://cassandra.apache.org/doc/cql/CQL.html

[4] For VoltDB's support of SQL and JSON, see: http://voltdb.com/products/; for Oracle Database 12*c*, JSON support will be part of an upcoming release as discussed in this Oracle Open World 2013 Session: "*Storing and Querying JSON Data in Oracle Database 12c [CON9348]*", San Francisco, September 2013

For the purpose of this discussion, the fundamental differences between NoSQL databases and RDBMS do not represent a compelling theoretical reason to use either type. NoSQL databases are typically "highly optimized key–value stores" and often provide a variety of ways for accessing data. In contrast, RDBMS are typically designed as multi-purpose DBMS using the SQL standard to access and modify the data. These differing access methods are in practice the most distinguishing aspects that should be considered when choosing a standard (general purpose) DBMS.

2) The ACID consistency vs. the BASE consistency model

Another fundamental design difference distinguishing various DBMS implementations is their consistency model. In computer science, two different consistency models are distinguished:

1. ACID: ***A***tomicity, ***C***onsistency, ***I***solation, ***D***urability
2. BASE: ***B***asically ***A***vailable ***S***oft-state ***E***ventual consistency

which basically describe contrary approaches to how reliable database transactions are processed by a given DBMS. A DBMS that follows ACID design principles[5] ensures the highest level of reliability for database transactions, whereas DBMS that follow the BASE design principles follow what is often referred to as an optimistic replication approach, which means that "reads eventually return the same value"[6] and the DBMS does not make safety guarantees:

An eventually consistent system can return any value before it converges. The verb "converge" in this context refers to a particular aspect of the eventual consistency model pertaining to using this consistency model in distributed computing. It basically describes an informal guarantee "that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value… A system that has achieved eventual consistency is often said to have converged, or achieved replica convergence." [Petersen et. al. 1997)]

Most of the database management systems nowadays represent distributed DBMS since they use a distributed approach for scaling. Hence, the consistency model is an important selection criterion and should be considered carefully when choosing a standard DBMS.

[5] Wikipedia, the free encyclopedia, 20th March 2014, *ACID*: http://en.wikipedia.org/wiki/ACID

[6] Wikipedia, the free encyclopedia, 20th March 2014, *Eventual consistency*: http://en.wikipedia.org/wiki/Eventual_consistency

2a) The CAP theorem

In the context of discussing the consistency model, some DBMS vendors refer to the CAP theorem. The CAP theorem by Brewer [Brewer 2000] basically says that a distributed shared-data system can at the most have two of the following three properties: ***C***onsistency, ***A***vailability, and ***P***artition Tolerance

Cassandra for example is very clear on their implementation of the CAP theorem: "Cassandra values Availability and Partitioning tolerance (AP). Tradeoffs between consistency and latency are tunable in Cassandra. You can get strong consistency with Cassandra (with an increased latency). But, you can't get row locking: that is a definite win for HBase. Note: Hbase values Consistency and Partitioning tolerance (CP)"[7]

The first publication of Brewer's CAP theorem dates back to 2000, but he clarified some of his positions in may 2012 [Brewer 2012], when he found that "In the decade since its introduction, designers and researchers have used (and sometimes abused) the CAP theorem as a reason to explore a wide variety of novel distributed systems.

The NoSQL movement also has applied it as an argument against traditional databases." Brewer clarified that "The "2 of 3" formulation was always misleading because it tended to oversimplify the tensions among properties. Now such nuances matter. CAP prohibits only a tiny part of the design space: perfect availability and consistency in the presence of partitions, which are rare."

Practically, the nuances do not matter that much, since some DBMS vendors (especially, those, classified as Niche Players by Gartner) still stick to the idea that choosing two properties is mostly "sufficient", whereas most of the RDBMS vendors (especially, those, classified as Leaders by Gartner) have always striven to satisfy all three properties with their RDBMS products in various ways.

3) Shared Disk Clusters vs. Shared Nothing Clusters

One of the areas where the leading RDBMS vendors have traditionally been at odds is the way storage is organized for the shared-data, distributed DBMS. In the ideal case, a client does not have to be aware of the location of the data within the DBMS. As long as this assumption is fulfilled, it is up to the DBMS (RDBMS or general DBMS) to decide on how the data is persisted on storage.

Traditionally, two approaches have been used by various DBMS: 1) Shared Disk 2) Shared Nothing

In a shared disk system, all servers that run a DBMS instance have (concurrent) access to all the data on storage, whereas in case of a shared nothing system, the data is partitioned and each server that runs an instance of the DBMS system accesses and manages only a subset of the data. Figure 1 illustrates the differences between these two architectures from a managerial point of view.

[7] http://wiki.apache.org/cassandra/ArchitectureOverview

Stonebraker [Stonebraker 1986] was one of the first ones that mentioned and advocated the shared nothing (SN) approach in his paper titled “The Case for Shared Nothing”. In very simple terms, he concluded that the SN architecture “will have no apparent disadvantages compared to the other alternatives. Hence the SN architecture adequately addresses the common case.” This basically means that according to his study, the SN architecture is no worse than any other alternative, while he clearly saw an availability advantage in the SN architecture.

De facto however, it has never incontrovertibly been proven that shared nothing always provides better availability or scalability. As summarized by [Michalewicz 2003], it is the management layer of the DBMS that can turn any DBMS into a potential single point of failure, not the shared disk based approach by itself, at least not in case of a DBMS that also implements an ACID consistency model and uses disk redundancy.

It is also the management layer and its ability to adapt to various use cases that mainly determines the scalability of the DBMS, not the way the storage is organized for the shared-data, distributed DBMS.

Finally, it needs to be noted that in any system shown in figure 1 disk-protection (redundancy) is required, as a failure of an unprotected disk would otherwise still lead to data loss. The amount of data that is affected by the loss of a (single) disk and the impact on the overall system differs between the three solutions. For the SN cluster, it may be assumed that there is a failover mechanism in place that allows a server in the cluster to manage an additional subset of data should the primary server fail.

3a) Sharding as an evolution of the Shared Nothing approach

Quite a few DBMS vendors use a variation of the shared nothing approach called sharding. Sharding combines two techniques: 1) Horizontal Partitioning & 2) Vertical Partitioning (illustrated in figure 1).

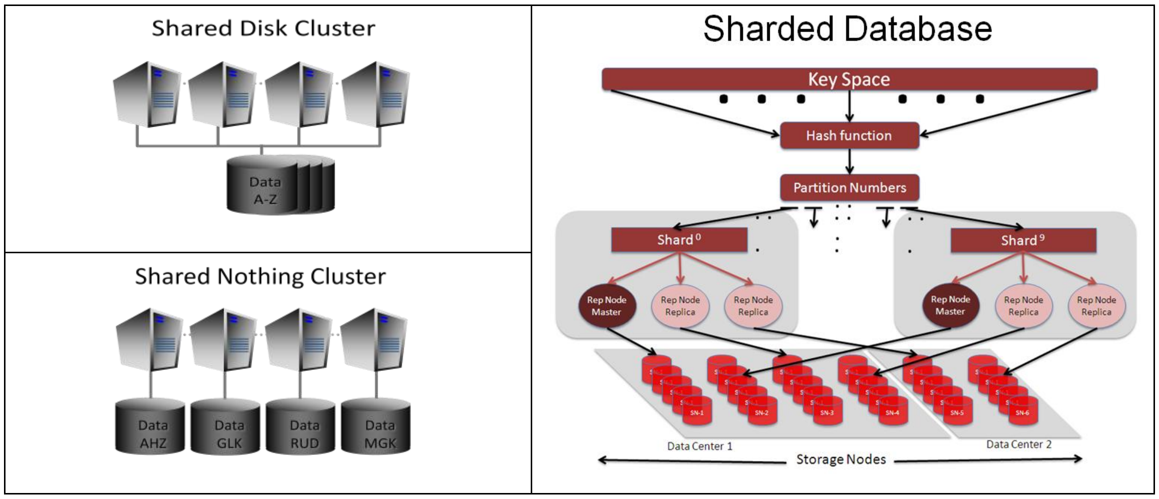

Figure 1: Shared Disk, Shared Nothing, and Sharding in comparison

Unlike shared nothing, sharding allows for shards to be independent databases, while the data remains to be partitioned. Sharding can therefore be considered an evolutionary step compared to shared nothing, but introduces further administrative challenges which should be considered for production databases. For example, if shards are distributed across data centers, the system as a whole needs to be made aware of this separation, especially assuming that replication is used as illustrated in figure 1.

From the same illustration, it also becomes apparent why most shard-based DBMS implement a BASE consistency model; it simplifies the recovery from partial failures in the system. On the other hand, using a BASE consistency model will make it more difficult to perform a consistent backup across all shards that constitute the sharded database.

From a practical point of view, sharding is simpler to implement as far as the DBMS is concerned and will provide scalability for certain applications, such as read-mostly or strict key-value pair-based applications. It also allows for flexible architectures, but requires a well thought-through system layout as well as continuous maintenance and product-specific training.

Exemplary history: Cassandra, mongoDB & the Oracle RDBMS

The history of Cassandra, mongoDB and the Oracle RDBMS are exemplary in showing how the requirements and technology that were available at the time when they were first invented influenced the design. The Oracle RDBMS was designed to be a universal DBMS, while Cassandra and mongoDB are purpose-built DBMS that were created to address certain data management requirements.

The history of the Oracle RDBMS is tightly coupled with the history of relational databases as first introduced by E.F. Codd in his famous paper, titled "A Relational Model of Data for Large Shared Data Banks", published 1970 [Codd 1970]. The Oracle RDBMS was the first commercial implementation of a Relational Database Management System:

"It all began when Larry Ellison saw an opportunity other companies missed: a description of a working prototype for a relational database. No company had committed to commercializing the technology, but Ellison and co-founders Bob Miner and Ed Oates realized the tremendous business potential of the relational database model."[8]

According to Oracle's Interactive Timeline[9], the first version of the "Oracle Database" was released in 1979: "Although RSI [author remark: Relational Software Inc., former company name] has supplied several government agencies with early versions of its RDBMS product, the company releases its relational database product, "Oracle," for the first time commercially in July [author remark: 1979]."

[8] *Oracle's History: Innovation, Leadership, Results*: http://www.oracle.com/us/corporate/history/index.html
[9] *Oracle Interactive Timeline*: http://oracle.com.edgesuite.net/timeline/oracle/

Cassandra and mongoDB have a far shorter history. According to DataStax, Cassandra "was created for solving the problem of inbox search at Facebook. It combines Amazon Dynamo's [DeCandia et. al. 2004] fully distributed design with Google Bigtable's [Chang et. al. 2006] column-oriented data model. Facebook open-sourced Cassandra in 2008 and it became an Apache Incubator project. In early 2010, Cassandra became a top-level Apache project."[10]

As Kristina Chodorow in her "Snail in a Turtleneck"-Blog[11] describes, "mongoDB was created by the founders of DoubleClick. Since leaving DoubleClick, they founded a number of startups and kept running into the same scaling problems over and over. They decided to try to create an application stack that would scale out easily, as companies everywhere seemed to be running into the same issues."

mongoDB was originally developed as the database backend for "an application platform for the cloud, similar to Google App Engine[12]". "The problem was, no one cared about Google App Engine and certainly no one cared about 10gen's [the company that was subsequently founded by the founders of DoubleClick] app engine. ... After a year of work and practically no users, we ripped the database out of the app engine and open sourced them."

The DBMS variety – a result & a choice, but not required

There is no database type that is simply the best one. Various DBMS are available on the market and as history shows, the traditional RDBMS have been designed as universal database solutions and hence can be used with a variety of applications and for various use cases.

Most of the upcoming NoSQL DBMS solutions were originally designed to address a certain data management requirement, mostly driven by a specific application. For applications that are similar to the one for which those solutions were designed, these DBMS represent a good solution. For other applications they will fall short in certain areas, as their optimization for one use case almost always means that compromises were made in other areas.

As the practical review of various database concepts shows, it is theoretically possible to solve most, if not all, data management requirements with one type of DBMS depending on the capabilities of DBMS layer used as part of a solution and hence, a rich variety of niche DBMS types is probably not required to satisfy market requirements.

[10] DATASTAX, *The History of Cassandra*, http://www.datastax.com/docs/0.8/introduction/index

[11] Kristina Chodorow Blog: http://www.kchodorow.com/blog/2010/08/23/history-of-mongoDB/

[12] Google App Engine: https://developers.google.com/appengine/?csw=1

How to Choose the Right DBMS for Your Business?

Many technical details can be reviewed when it comes to choosing the right DBMS for your business. However, one aspect will be the one influencing the decision the most: costs. The price points at which most of the niche DBMS vendors offer a supported version of their solution is understandably attractive at first glance, while the question remains whether they provide the lowest *T*otal *C*ost of *O*wnership (TCO) in the long term. In other words:"*Do you want to pay now or later?"*

Speaking to various customers, one once told Oracle that over the last year, they have deployed over 180 new NoSQL database projects, which means that this customer has deployed an average of 15 new NoSQL projects per month. The same customer, however, admitted later that if they had known that there will be 180 new NoSQL DB installations after a year, they would have re-considered this approach in the first place, as the number of NoSQL DBMS deployments gets unmanageable.

This is an example for the cost challenges associated with using purpose-built DBMS. In case of traditional RDBMS the license and support costs are typically the main cost contributors to the total cost of ownership. With NoSQL or purpose-built database solutions, the TCO is mostly influenced by operational expenditures such as administration costs and secondary solution costs, such as additional administrative tools and support that needs to be purchased to efficiently operate those databases.

The subsequent part of the paper will review the six most relevant areas that you should take into consideration when choosing a standard DBMS in order to determine whether the DBMS solution of choice will be able provide the lowest TCO for your data center in the long term.

Operation (standard features and management)

Cassandra, mongoDB and other niche solutions often do not provide features that are generally considered "standard features" for traditional RDBMS, such as joins or aggregation. Providing such functionality is not a strict requirement, but has proven to be useful, as the lack of such features will lead to cost intensive workarounds. For example:

1. Storing similar data more than once.
2. Using additional software external to the database to provide missing functionality.
3. Running multiple queries to get a sub-set of data and letting the application perform the joins.

Similar holds true for administration. For production environments the installation, maintenance and patching needs to be orchestrated and ideally automated. While automation is not a strict requirement, it has proven to be useful, as the lack of such features will lead to cost intensive workarounds such as:

1. DBAs write installation and patching scripts.
2. Orchestration software is purchased to monitor and maintain the DB deployments.

Part of the value proposition of most of the purpose-built DBMS solutions is that deployments are rpm-based, making it simple and fast to deploy an initial database. Replication and replication groups can be enabled and managed by the user to allow for maximum flexibility.

For a production environment this type of flexibility can be expensive, while the question arises of whether or not the initial deployment time is relevant, as it should be an infrequent task. Configuring and maintaining a cluster for scalability can be a rather challenging and ongoing task. It is therefore desirable that the DBMS of choice provides an easy way of managing the system as a whole.

In a simple Cassandra cluster with six servers[13] (clustered servers are referred to as "nodes") across two data centers for example, the setup not only foresees potential restarts and data clearance, it also requires maintaining a configuration file (the Cassandra.yaml configuration file) in order to define the nodes in the cluster as well as their roles (e.g. seed nodes). In a 6-node cluster, the Cassandra yaml configuration file contains six of these node entries. In an *n*-node cluster, *n* entries need to be manually maintained and distributed across *n* nodes. In addition, one topology file needs to be maintained.

Using Cassandra or mongoDB, this type of day-to-day management, including patching and upgrades, has to be performed without the help of management tools, unless secondary solutions are purchased. Using a standardized solution such as Oracle Real Application Clusters (RAC), standard management tools are not only provided free of charge, they are fully cluster-aware and thereby simplify day-to-day management tasks in clusters of any supported size.

Furthermore, certain management operations are not required in a shared disk-based RDBMS, such as Oracle RAC, as they are inherent to using sharding as a scalability approach. For example, in case of a Cassandra database, reorganization of the cluster or the structure of the shards require manual edits as well as redistribution of configuration files along with a potentially manual (re-)distribution of binaries. Depending on the use case for the database, those operations may become a regular task as explained in the subsequent chapter of this paper. Using Oracle RAC, data is stored in a central fashion and will almost never require physical reorganization or redistribution, lowering the TCO in the long term.

Data Management (access and modification)

From a development point of view, NoSQL databases are attractive, as they allow for a very flexible and dynamic development style. Unlike relational databases, NoSQL databases do not require a restrictive relational data model to be established prior to developing the application and populating data. Key-value pairs are often "good enough" for a lot of specific use cases.

[13] DataStax, *Initializing a multiple node cluster (multiple data centers)*, http://www.datastax.com/documentation/cassandra/2.0/#cassandra/initialize/initializeMultipleDS.html

However, “flexible” does not mean “simple”. When it comes to choosing a standard DBMS specifics matter, since using a variety of different solutions that do not follow any standards otherwise increases training requirements and costs. Data modeling and optimization as well as data storage requirements should therefore be reviewed for each DBMS that is planned to be used in the data center.

As a side effect of using a key-value pair approach to storing data, one may have to de-normalize and duplicate data for read performance, although one should not de-normalize, if not needed. “It’s all about finding the right balance” as A Jay Patel in his eBay tech blog titled “Cassandra Data Modeling Best Practices, Part 1” illustrates[14]. Finding this balance is a planning intensive task and requires ongoing optimization, leading to higher operational costs.

As Jay Patel in his blog continues to explain, choosing an appropriate data model optimized for the (anticipated) access pattern of the application that will be used with the database has a strong influence on the efficiency and the performance that one can expect from the database. This especially applies to Cassandra databases, which use a key-value pair approach to store data, on top of which a SQL-like layer (CQL = Cassandra Query Language) can be used to access the data.

Cassandra supports indexes to look up data, but does not support joins. Ensuring good performance for frequent queries based on data from related data sets may require keeping the data for each set of data as well as the joined data. As a side effect, dynamic queries may often still not provide the best performance, as the data may not have been optimized for them.

Migration (inter-solution data access and migration)

Another challenge in addition to the management costs associated with handling distributed NoSQL databases will be the migration of projects that outgrow the capabilities of a given NoSQL database over time. Developers often use NoSQL databases because they very efficiently solve a problem at hand. Over time, however, these applications may become more complex or more important so that those databases need to be integrated into an existing infrastructure that ensures certain Service Level Agreements (SLAs). At this time, a migration to a database tailored to hosting business critical applications may become required.

Accessing or moving data between different NoSQL databases can be challenging, if the data model needs to be optimized depending on the specifics of the DBMS. Standardization has traditionally been an attempt to simplify moving between databases. In the absence of a NoSQL standard to storing data, JSON seems to be one of the upcoming document-based models that are frequently used to store data in a format that will satisfy a lot of the “modern development requirements.”

[14] Patel, Jay, *Cassandra Data Modeling Best Practices, Part 1*, http://www.ebaytechblog.com/2012/07/16/cassandra-data-modeling-best-practices-part-1/#.Uwa4NRCmVac

Most of these modern development requirements are based on the idea to change the application data model on demand, which is contrary to the idea of using schemas. Schema-less data management with the ability to persist application objects in a document-centric approach becomes more and more popular; so are the databases supporting this model.

mongoDB supports JSON, as well as CouchDB and Oracle's NOSQL database. Starting with Oracle Database 12c Patch Set One (12.1.0.2), JSON will also be supported in the Oracle Database, which will provide access to the JSON data using various APIs, including SQL, as it was announced during Oracle Open World 2013, session [CON9348] - "*Storing and Querying JSON Data in Oracle Database 12c*".

Using JSON as a document-centric standard for storing data has multiple advantages. It is a defined, yet flexible data model that allows schema-less development. Creating indexes on documents has been a widely supported feature for a variety of DBMS, although optimized indexes are still required. Last but not least, a standardized way of storing data will simplify the migration.

Choosing a NoSQL database that supports JSON for storing data will therefore be useful when the need arises to move data into the Oracle Database for example. There is now a choice to either retain the JSON documents or convert the data model into a relational model as needed. Either case, one will immediately benefit from the Oracle Database as an infrastructure.

One of the benefits that Oracle as a company offers in this context is a migration path between different database solutions. As Andy Mendelsohn, Executive Vice President Oracle Database Server Technologies, explains in his keynote during the NoSQL Now! 2013 Conference & Expo[15], Oracle offers a variety of standard database management solutions, tailored to various use cases as illustrated in figure 2 below.

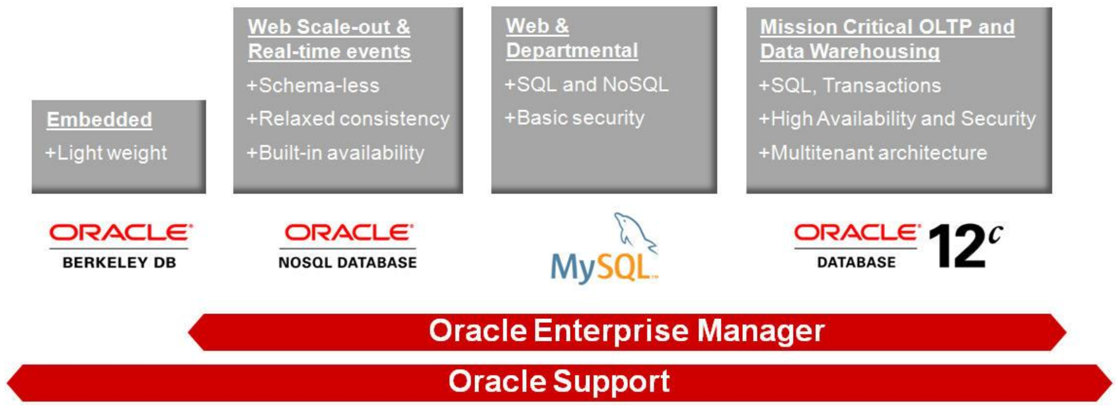

Figure 2: Oracle Databases - Providing the right database technology for the job

[15] Mendelsohn, A., *Oracle NoSQL Database*, http://www.slideshare.net/Dataversity/keynote-presentation-oracle-nosql-database

Additional value that Oracle as a DBMS (not just RDBMS) vendor provides is the integration of the Oracle solutions with Oracle Enterprise Manager (EM) as the standard management tool as well as a support organization that acts as the primary point of contact for all DBMS related questions. Oracle Enterprise Manager enables monitoring and automation for any Oracle Database solutions free of charge, while Oracle Support as the primary point of contact eliminates effortful multi-vendor calls in case support for any DBMS is required. Consequently, Oracle as a solution provider should always be considered when it comes to choosing any type of DBMS for your data center.

Scalability (vertically and horizontally)

Horizontal scalability is one of the main selling points for both Cassandra and mongoDB and both solutions use a sharding approach for scalability. For read-mostly applications, sharding will work; even for data warehouse applications with ETL-based data loads, sharding works as designed. For OLTP applications alternative approaches provide a better performance.

When using sharding to scale horizontally, which means using more than one server in the data center concurrently to run an operation, the question of vertical scalability inevitably arises, so do questions regarding partially (un)available data.

Vertical and horizontal scalability

In sharded environments, the size of a shard is typically determined by a hash function or the data is partitioned using a given partition key. The distribution of work across shards thereby depends either on the system and the hash function or the structure of the data. Either case, the non-trivial question of how many servers need to serve a shard will need to be answered.

If using more than one NoSQL database as part of the deployment in the data center, using servers as shared resources and allocating databases from different vendors on the same server is one deployment model that will soon become a necessity, raising the question on how to isolate the various databases on the same server. Virtual machines are typically the logical answer. Virtual machines work fine for this matter, but will increase the management overhead and resource utilization.

Sharding is not the simplest approach when maintaining databases; neither will it necessarily help to reduce the data stored. Sharding assumes that data is partitioned over various data stores. This means that the total amount of data that is stored as part of the databases is merely partitioned, not reduced. If it is further assumed that each shard is replicated to prevent data loss in case of a (partial) failure, the amount of data stored as part of the database doubles in the same way as if a shared disk approach with mirroring was used.

Consequently, sharding does not solve all the problems. It certainly avoids some problems, but leaves others unsolved or to the discretion of the user. The latter approach can lead to a significant amount of planning as well as maintenance, thereby increasing management costs.

Dealing with partially (un)available data

One of the biggest disadvantages of sharding (as mostly used in combination with a BASE consistency model) is that the DBMS needs to provide a solution on how to deal with partially (un)available data.

Partially (un)available data generally appears in two ways: 1) a shard is not available (despite replication) at all at the moment a query is run or 2) the data from a shard cannot be returned in a reasonable amount of time. A subset of the partially available data problem is the "inconsistent data problem", which can occur in databases that have chosen to implement eventual consistency.

Eventual consistency (BASE) assumes that a database "converges" at some point in time. In the meantime, however, it is not assured that all replicas will have the latest update of the modified data. Querying those replicas during this time can lead to wrong results and therefore requires the application to be sharding-aware, unless the sharded DBMS prevents reading from those replicas, which typically leads to a decrease in performance.

Considering various sharding approaches and the configuration choices that need to be made to optimize their performance (often referred to as "making the application sharding-aware"), it is fair to say that if similar optimization is performed on a per-application basis in a shared disk environment, the application would perform equally well, if not better.

Using shared disk, however, the application developer or administrator would typically not have to think about "making the application sharding-aware", "partially (un)available data", "eventual consistency" and the access pattern that the application will expose in order to optimize the data, leading to reduced overall planning and managing costs.

High Availability (locally, remotely and long term)

Cassandra and mongoDB provide a defined level of high availability. The fundamental assumption, however, is that high availability is based on partition tolerance. Both solutions also use clustering as well as sharding to achieve availability. Clustering is used to coordinate roles in the cluster and to determine availability of components. Both solutions thereby combine what is typically referred to as protection against component failures and protection against data loss.

Protection against component failures

Cassandra explicitly mentions its failure tolerance and continuous availability in addition to scalability, which in case of the Cassandra database is based on a token-ring cluster architecture. Cluster configuration information is exchanged via special servers in the cluster called "seed nodes". Cluster membership information and updates are exchanged via the public network (by default; a dedicated network is recommended) using the "gossip" protocol.

This architecture is suitable for managing server membership in the Cassandra cluster, but limited when it comes to managing consistency, node fencing or any other application. In addition, using a database-specific clustering technology has various disadvantages and Oracle is the first to admit that having database-specific clustering solutions are hard to justify, as this was the approach taken when Oracle Real Application Clusters (RAC) 9*i* was first released more than ten years ago.

For a production environment the need to use various clusters on the same infrastructure will increase the chattiness on the (public) network and increase the management and training costs as well as the complexity of the system, since multiple specific cluster solutions need to be maintained individually.

Using different cluster models for availability and scalability also makes the migration between different DBMS difficult, as one would have to consider the dependencies on the cluster for each database management system. This complicates the integration into an existing infrastructure, as applications using a DBMS might require a cluster that (partially) overlaps with the DBMS cluster for example. Using a general purpose cluster for both layers simplifies this integration and thereby allows for reducing the number of items that need to be managed to in the data center.

Oracle stopped using Oracle RAC-specific clustering solutions on Linux and Windows when Oracle Clusterware 10g was release. Oracle Clusterware is a generic clustering solution that is available on all Operating Systems supported for Oracle RAC. Oracle Clusterware serves as the integrated foundation for running Oracle RAC[16] databases, while it provides a cluster-based resource management solution for any application. As part of the Oracle Grid Infrastructure, Oracle Clusterware can also be used to manage MySQL databases in the cluster[17].

Protection against data loss

Unlike Cassandra and mongoDB, traditional RDBMS typically use a separation between clustering and protection against storage failure, following the distinction between local and remote high availability.

Local high availability is typically provided by a cluster in this case, while remote high availability is achieved by a replication mechanism that typically does not require tight coupling between nodes either on server- or on the storage-level. The replication mechanism that Cassandra uses follows a similar paradigm by allowing hosting a replica in a different data center, but in order to make efficient use of this replica, it needs to be in the same database cluster.

In case of an Oracle Database, the local database is typically protected by Oracle RAC and a remote protection is provided by Oracle Active Data Guard. The assumption is that the shared disk-based

[16] For more information on Oracle Real Application Clusters (RAC), see http://www.oracle.com/goto/rac

[17] For more information see the “Oracle Grid Infrastructure (Bundled) Agents” Reference Guide: http://www.oracle.com/technetwork/database/database-technologies/clusterware/downloads/ogiba-2189738.pdf

Oracle RAC cluster uses a first level storage failure protection such as a local RAID implementation, while the site synchronized via Oracle Active Data Guard would be used as a secondary storage failure protection site, should the complete storage in the first site fail. In addition, Oracle Active Data Guard can be used to outsource read-only queries and acts as a secondary protection, should all the servers in the primary cluster fail at the same time.

The downside of separating the protection provided by the cluster and by the replication as in the Oracle RAC / Oracle Active Data Guard example is that the data potentially needs to be maintained three times (active set of shared disks, their mirror, and the Data Guard copy). Using parity-based RAID implementations can help to reduce the amount of data that has to be stored.

Sharding, however, does not necessarily help to reduce the amount of data that needs to be stored on disk. If the assumption is that high availability of the shard is maintained while re-organization takes place, likeliness is that multiple copies of the sharded data need to be maintained. If full redundancy is maintained for every shard that constitutes the database, no data reduction takes place at all, while there is still additional overhead in maintaining this setup.

Security (for data at rest and in transit)

Most NoSQL databases are used for applications that work on "big data", which means that there is a "huge amount of" security needed to protect this data from un-authorized access. Unauthorized data access can occur when the data is at rest (when being stored on disk) or while in transition (being read, updated, written or while generally being transferred via the network).

Given that in sharded environments data is not only read on a frequent basis, but potentially also transferred via the network due to re-organization or relocation needs, the DBMS should ideally provide independent security features such as encryption for data at rest as well as for data in transit (network encryption). For the latter, a standard solution can be used.

If local disks are used for storing data in shards, removing the data from those disks should also consider providing an "erase" functionality to prevent un-authorized access to data after a subsequent de-commission of those disks, especially if encryption functionality is not offered or used. While it could be argued that this type of functionality is not necessarily provided by other, shared disk-based solutions, either, it needs to be considered that these solutions by definition either assume enterprise-level storage (as opposed to local storage), for which respective operational flows have already been established in most data centers, or they provide a default encryption solution.

Similar considerations apply to other areas, which are often overlooked as they are taken for granted using traditional multi-purpose RDBMS. Such areas cover, but are not limited to, backup and recovery for example. Using the Oracle RDBMS as an example, backup and recovery has been a strong design point right from the start. RMAN (the Oracle Recovery Manager) is the free-of-charge standard tool to back up an Oracle Database in various states (online or offline). It is well integrated with third party backup solutions, supporting useful features such as incremental backups as well as snapshot based backups, of which all can be combined with security features inherent to the Oracle Database.

Conclusion

The IT business evolves in circles with a tendency to form a spiral. This discussion shows that the recent increase in the number of NoSQL databases, mostly considered Niche Players in Gartner's recent Magic Quadrant (MQ) for Operational Database Management Systems, formerly OLTP [Gartner 2013], is no surprise. Most of these solutions are purpose-built DBMS that were developed to address very specific use cases.

The popularity of some of those solutions and the support that they have received is a result of the cost advantages they provide at first glance. For some, rather specific use cases, using those purpose-built databases makes perfect sense.

However, NoSQL databases have not solved any problem that cannot be solved using traditional RDBMS. This means, it is more likely that the currently leading RDBMS vendors will close the perceived gaps in functionality leading to a broad but rather insignificant adjustment of the market.

The purpose-built nature of most NoSQL DBMS allows them to benefit from advances in other technology areas, such as networking, faster. Traditional relational database management systems have a longer history and need to maintain backward compatibility, hence, may need some time to adopt new technologies, while they eventually will adapt accordingly.

Most of the leading RDBMS vendors listed in Gartner's recent Magic Quadrant (MQ) have already solved the problems that nearly all NoSQL databases will have to solve when hitting certain thresholds as a result of growth. Growth and requirements of business critical applications will force almost all NoSQL DBMS vendors to provide comprehensive solutions for data consistency, security and overall management, cutting into the cost benefits they seem to provide at first glance.

Concluding, the leading relational DBMS providers will not rest to provide the same benefits that NoSQL databases provide. They will add the additional benefits they already contain and offer the integrated solution at a competitive price point. First in line: Oracle with its rich DBMS solution portfolio, including its flagship solution, the Oracle Database 12*c*. Welcome to the future.

Appendix A – References

[Brewer 2000] Brewer, Eric (2000), *Towards Robust Distributed Systems*: http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

[Brewer 2012] Brewer, Eric (2012), *CAP Twelve Years Later: How the "Rules" Have Changed:* http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed

[Chang et. al. 2006] Fay Chang, Jeffrey Dean, Sanjay Ghemawat,Wilson C.Hsieh, Deborah A.Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E.Gruber (2006), *Bigtable: A Distributed Storage System for Structured Data*: http://static.googleusercontent.com/media/research.google.com/en/us/archive/bigtable-osdi06.pdf

[Codd 1970] Codd, E.F. (1970), *A Relational Model of Data for Large Shared Data Banks*, http://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf

[DeCandia et. al. 2007] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swami Sivasubramanian, Peter Vosshall and Werner Vogels, "*Dynamo: Amazon's Highly Available Key-Value Store*", in the *Proceedings of the 21st ACM Symposium on Operating Systems Principles*, Stevenson, WA, (2007): http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf

[Gartner 2013] Gartner (October 2013), *Magic Quadrant for Operational Database Management Systems*: http://www.gartner.com/technology/reprints.do?id=1-1NAMY5B&ct=131121&st=sb

[Michalewicz 2003] Michalewicz, M. (2003): *Eigenschaften von Hochleistungsdatenbanken (im Cluster) vor dem Hintergrund der Verfügbarkeit am Beispiel der Oracle9i Datenbank mit Oracle Real Application Clusters,* Datenbank-Spektrum 7: pages 28-37

[Petersen et. al. 1997] Petersen, K.; Spreitzer, M. J.; Terry, D. B.; Theimer, M. M.; Demers, A. J. (1997), "*Flexible update propagation for weakly consistent replication*". *ACM SIGOPS Operating Systems Review* 31 (5): 288. doi:10.1145/269005.266711

[Stonebraker 1986] Stonebraker, M. (1986), *The Case for Shared Nothing*: http://db.cs.berkeley.edu/papers/hpts85-nothing.pdf

ORACLE®

Back to the Future with Oracle Database 12c
April 2014
Author: Markus Michalewicz
Contributing Authors: Bob Thome

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com

Oracle is committed to developing practices and products that help protect the environment

Hardware and Software, Engineered to Work Together