ORACLE
NOSQL DATABASE

Oracle NoSQL Database

Fast, Reliable, Predictable

ORACLE WHITE PAPER | JUNE , 2018

Contents

Boosted by the Oracle France WeLoveStartups initiative, we tremendously increased our Big Data capabilities using Oracle REST Data Services to leverage our Oracle NoSQL Database, Enterprise Edition. Only Oracle could have helped us achieve a fully functional Big Data infrastructure in just hours.

WASSEL GUERBAA, CEO

CEO

BUSIT SAS

Introduction

Oracle NoSQL Database enables the creation of innovative applications that engage customers and create business value for constantly changing business requirements. Your organization can quickly respond to new opportunities that require very fast storage and retrieval of data with extremely low latency. Oracle NoSQL Database is a scalable, distributed NoSQL database, designed to provide highly reliable, flexible and available data management across a configurable set of storage nodes. Oracle NoSQL Database has been designed to be flexible in a number of areas:

- Developers can create innovative applications using a number of popular programming languages and are able to model the data in a number of ways.
- Administrators can plan for varying workloads by scaling both vertically and horizontally, using industry standard servers.

Oracle NoSQL Database Overview

NoSQL databases represent a recent evolution in enterprise application architecture, continuing the evolution of the past twenty years. In the 1990's, vertically integrated applications gave way to client-server architectures, and more recently, client-server architectures gave way to three-tier web application architectures. In parallel, the demands of web-scale services added very low latency real-time access as well as offline map-reduce processing into the mix. Data architects started eschewing transactional consistency in exchange for incremental scalability and large-scale distribution. The NoSQL movement emerged out of this second ecosystem.

NoSQL is often characterized by what it's not – depending on whom you ask, it's either not only a SQL-based relational database management system, or it's simply not a SQL-based Relational Database Management System (RDBMS). While those definitions explain what NoSQL is not, they do little to explain what NoSQL is. Consider the fundamentals that have guided data management for the past forty years. RDBMS systems and large-scale data management are characterized by the transactional ACID properties of Atomicity, Consistency, Isolation, and Durability. In contrast, NoSQL is sometimes characterized by the BASE acronym:

Basically **A**vailable: Use replication to reduce the likelihood of data unavailability and use sharding (partitioning the data among many different storage servers) to make any remaining failures partial. With the advent of sharding and replication for availability comes the resulting benefit of share-nothing architecture and horizontal

scaling. The result is a system that is massively scalable and always available, even if subsets of the data become unavailable for short periods of time.

Soft state: While ACID systems assume that data consistency is a hard requirement, NoSQL databases allow data to be inconsistent and relegate designing around such inconsistencies to application developers.

Eventually consistent: Although applications must deal with instantaneous inconsistency, NoSQL systems ensure that at some future point in time the data assumes a consistent state. In contrast to ACID systems that enforce consistency at transaction commit, NoSQL guarantees consistency only at some undefined future time.

NoSQL emerged as companies, such as Amazon, Google, LinkedIn and Yahoo struggled to deal with unprecedented data and operation volumes under tight latency constraints. Responding in real time or near real time to millions of requests while maintaining predictable latencies became the core technical driver for enabling rich user experiences, highly targeted ad placement systems, and the collection of large volumes of machine-generated data for subsequent offline analysis. Traditional relational databases were not up to the task, so enterprises built upon a decade of research on distributed hash tables (DHTs) and either unconventional relational database systems or embedded key/value stores, such as Oracle's Berkeley Database (BDB), to develop highly available, distributed key-value stores.

While many NoSQL databases have evolved over the last several years, a scant few have been able to deliver on the notion of truly flexible transaction models. Typically, a NoSQL database must trade-off the ability to offer the developer a consistent and isolated view of data for either performance, scale, ease of programming, or all the above. In contrast, The Oracle NoSQL database offers a hybrid ACID/BASE model with near zero impact on performance and scale while achieving the ease of programming that delights developers. Application architects and developers can decide when it is appropriate to relax transactional constraints or to tighten them up. With each access to the database, the appropriate level of consistency (for reads) and durability (for writes) can be chosen based on the specific needs of the application.

Organizations have embraced NoSQL technology as one of the technology dimensions in the enterprise application architecture, and Oracle's NoSQL Database provides all the desirable features of NoSQL solutions necessary for seamless integration into any enterprise application architecture. Figure 1 shows a canonical acquire-organize-analyze data cycle, demonstrating how Oracle's NoSQL Database fits into such an ecosystem. Oracle provided modules allow the Oracle NoSQL Database to integrate with Hadoop MapReduce, Hive, Spark, or with the Oracle Database. Utilizing the Oracle Database for accessing Oracle NoSQL Database data further enhances the integration landscape for the business, allowing existing BI tooling, Oracle Enterprise R, or Oracle's Advanced Analytics to access data stored in the Oracle NoSQL Database.

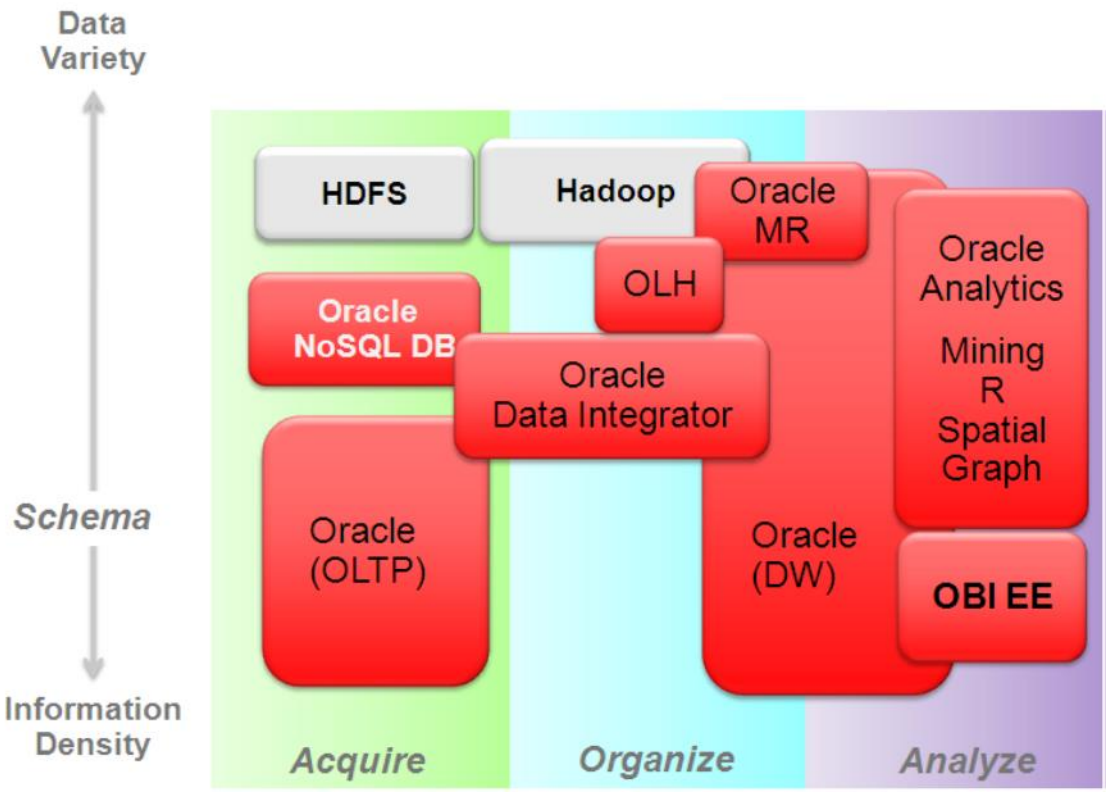

Figure 1 - Acquire-Organize-Analyze

The Oracle NoSQL Database, with its "No Single Point of Failure" architecture is the right solution when data access is "simple" in nature and application demands exceed the volume or latency capability of traditional data management solutions. For example, click-stream data from high volume web sites, high-throughput event processing, and internet-of-things (IoT) sensor data all represent application domains that produce extraordinary volumes of simple keyed data. Monitoring online retail behavior, accessing customer profiles, pulling up appropriate customer ads and storing and forwarding real-time communication are examples of domains requiring the ultimate in low-latency access. Highly distributed applications such as real-time sensor aggregation and scalable authentication also represent domains well-suited to Oracle NoSQL Database.

Technical Overview

Oracle NoSQL Database leverages the Oracle Berkeley Database Java Edition High Availability storage engine to provide distributed, highly-available key/value and table storage for large-volume, latency-sensitive applications or web services. While Oracle Berkeley DB provides the underlying storage management functionality, the upper layers of the Oracle NoSQL Database provide critical features such as online elastic scale out and scale in, support for multiple data models (key/value, table, JSON (JavaScript Object Notation) documents and graph), SQL Query, full text search, authentication, authorization, and multi-datacenter disaster recovery.

Architectural Data Model

At its core, Oracle NoSQL Database implements a key/value map from user-defined keys to opaque data items. It records version numbers for key/data pairs but maintains the single latest version in the store. Applications do not need to be concerned about reconciling incompatible versions because Oracle NoSQL Database uses leader based replication; the Paxos master node always has the most up-to-date value for a given key, while non-master replicas might have slightly older versions. Applications can use version numbers to ensure consistency for read-modify-write operations.

Data Modeling Options

As a true multi-model data store, the Oracle NoSQL Database provides several different options for data modeling:

- Key/value pairs – Using this modeling option, developers specify string keys and opaque byte arrays as values. The Oracle NoSQL Database does not interpret the byte arrays. These are serialized and de-serialized by the application.
- Tables – Using this modeling option, developers specify table definitions using a SQL data definition language, very similar to the relational database. Command line SQL for querying the Oracle NoSQL Database or the Oracle NoSQL Database APIs can be used for table data access.
- Graph – Using this modeling option, developers define any number of name/value pairs for the edges or vertices of the graph. Then, edges are created that draw "labeled" relationships between the graph vertices.
- JSON documents – Using this modeling option, developers can store JSON objects in the Oracle NoSQL Database and use SQL to access these objects. The Oracle NoSQL Database provides a rich SQL dialect with specialized operators designed specifically for slicing through JSON objects. Secondary indexes specified by JSON path expressions are also supported via this SQL dialect.

Developers may choose to use either raw key/value pairs or tables to model the data for their applications. Several factors should be considered when choosing to model data one way or another:

- Secondary or full-text indices needed? If your application may benefit from secondary indices or full-text indices, then you must use tables instead of raw key/value pairs to model your data. Oracle NoSQL Database uses the table definition to enable secondary index creation.
- Fine-grained authorization needed? If you need to secure your data by using fine-grained authorization, then you must use tables instead of raw key/value pairs to model your data.
- SQL query – Will you plan to access your data using SQL or API calls? If you plan to use SQL, then you must use tables instead of raw key/value pairs to model your data.
- JSON documents – Will your application be designed to use JSON as the primary data format? If this JSON needs to be stored persistently and queried, then you must use the table model and the associated JSON datatype.

Sharding Model

Oracle NoSQL Database hashes keys to shards to provide distribution over a collection of storage nodes that provide storage for the database. However, applications can take advantage of the subkey (also known as a child table) capability to achieve data locality. A primary key is the potential concatenation of a shard key and a non-shard key, both of which are specified by the application. All records sharing a shard key are co-located to achieve data-locality. Within a co-located collection of shard keys, the full primary key, comprised of both the shard and non-shard portions, provides fast, indexed lookup. For example, an application storing user emails might use the user ID as a shard key and then have several child tables for different email folders owned by the user such as inbox, deleted, drafts, etc. When rendering the user interface upon login, the first page of a user's inbox messages (or any other folder for that user) can be read using a single API call from the application. In other words, a single network call can be made to Oracle NoSQL Database to retrieve all of the messages for a particular user.

Consistency and Durability Model

While many NoSQL databases provide eventual consistency, Oracle NoSQL Database provides several different consistency policies. At one end of the spectrum, applications can specify absolute consistency, which guarantees

that all reads return the most recently written value for a designated key. At the other end of the spectrum, applications capable of tolerating inconsistent data can specify weak consistency, allowing the database to return a value efficiently even if it is not entirely up to date. In between these two extremes, applications can specify time-based consistency to constrain how old a record might be with respect to the latest version at the leader, or version-based consistency to support both atomicity for read-modify-write operations and reads that are at least as recent as the specified version. Figure 2 shows how the range of flexible consistency policies enables developers to easily create business solutions providing data guarantees while meeting application latency and scalability requirements.

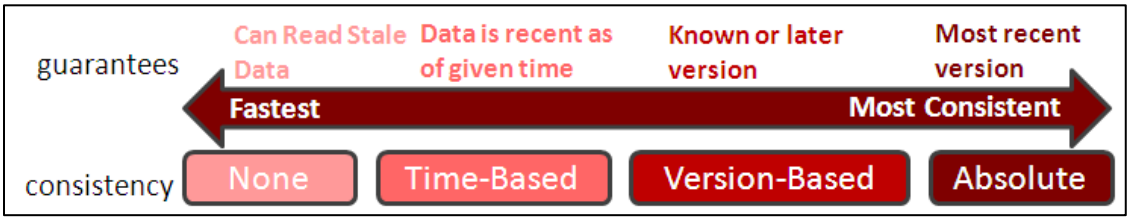

Figure 2 - CAP Model

Oracle NoSQL Database also provides a range of durability policies that specify what guarantees the system makes after a crash. At one extreme, applications can request that write requests block until the record has been written to stable storage on all copies. This has obvious performance and availability implications but ensures that if the application successfully writes data, that data will persist and can be recovered even if all the copies become temporarily unavailable due to multiple simultaneous failures. At the other extreme, applications can request that write operations return as soon as the system has recorded the existence of the write, even if the data is not persistent anywhere. Such a policy provides the best write performance but provides no durability guarantees. By specifying when the database writes records to disk and what fraction of the copies of the record must be persistent (none, all, or a simple majority), applications can enforce a wide range of durability policies and make the crucial trade-offs between latency and durability.

Programming Models

As previously stated, developers have a rich choice of modeling constructs in Oracle NoSQL Database, and these data modeling constructs lead directly to the programming model. Oracle NoSQL Database enables developers to both model their data and use a natural API together. For example, if you need to use JSON as your data model, the Oracle NoSQL Database JSON API has been developed to simplify the development of your application. Likewise, using tables to model your application will lead to the natural use of Oracle NoSQL's Database Table API. In this paper, we will give examples of these two specific models.

The Table Model

One of the data models exposed by the Oracle NoSQL Database is the table model. While this model may be more familiar to developers experienced with relational databases, the Oracle NoSQL Database table model offers unique datatypes not found in the relational world such as arrays, maps, and embedded records. Creating tables in Oracle NoSQL Database is accomplished via SQL Data Definition Language (DDL), very similar to relational SQL. Below is an example DDL statement that will create a table called User with the following columns:

- Id – This will be the primary as well as shard key (unless specified, the shard key defaults to the primary key)
- firstName and lastName – Strings to store the first and last name for the user.
- An array of address records – We will store multiple addresses for this user. Each address record will contain the following attributes:
    - Street, City, State – The street, city, and state for this address

- zip – The integer zip code for this address
- addrType – An enumeration that can contain either “work” or “home”

In Oracle NoSQL Database, the DDL for creating this table can be specified as:

```
Statement result res = KVStore.executeSync("CREATE TABLE user(id INTEGER,
        firstName STRING, lastName String,
        addresses ARRAY(RECORD (street STRING, city String, state STRING,
                                zip INTEGER, addrType ENUM(home, work),
                                primary key(id))))");
```

Incorporating Oracle NoSQL Database into applications is straightforward. APIs for basic Create, Read, Update and Delete (CRUD) operations, as well as SQL access, are packaged in a single jar file. Applications can use the APIs or SQL queries from one or more client processes that access a stand-alone Oracle NoSQL Database server process, alleviating the need to set up multi-system configurations for initial development and testing.

Accessing Table Data via SQL

For queries that don’t fit the pattern of simple key/value access, or simple range scans by a secondary key, the Oracle NoSQL Database offers native SQL access to the data. While the Oracle NoSQL Database provides a fairly rich SQL dialect, this SQL can be seen as a simpler subset of the more powerful and comprehensive SQL provided by the Oracle RDBMS. For those applications that require this level of SQL access, the Oracle NoSQL Database is integrated with the Oracle RDBMS, allowing the RDBMS to query data directly from Oracle NoSQL Database tables.

One notable area of difference between the Oracle RDBMS and Oracle NoSQL Database dialects of SQL is the notion of SQL query-ability over non-relational data such as arrays and maps. The SQL dialect in the Oracle NoSQL Database provides powerful constructs for slicing through arrays and maps as well as providing filtering expressions over these structures. The examples below illustrate a small subset of this capability.

1. Return the first and last names of all users in New York city:

```
select
    firstName,
    lastName
from
     user
where
     addresses.phones[$element.city = "New York"]
```

2. Return the first and last names of all users who have a home address in the 94107 zip code and order the results by the lastName attribute:

```
select
    firstName,
    lastName
from
     user
where
     addresses.phones[$element.zip= 94107] and
     addresses.phones[$element.attrType = "home"]
order by lastName
```

3. Display the first page (25 results per page) of all users in New Haven, Newark, and New York City:

```
select
    *
from
    user
where
    addresses.phones[$element.city = "New York"] or
```

```
            addresses.phones[$element.city = "New Haven"] or
            addresses.phones[$element.city = "Newark"]

        order by lastName limit 25 offset 1
```

Working with JSON Documents

JSON has grown dramatically over the last several years as a convenient data format for web applications, returning results from service API calls, and data interchange between applications. As a self-describing data type, JSON formatted data offers the ultimate in flexibility for those applications wishing to store ad-hoc content without having to specify the schema up front. Fixed schemas are stored once, while every JSON document contains a copy of the schema.

Oracle NoSQL Database offers a unique combination of fixed schema tables as well as ad hoc schema-less support for JSON documents. Developers can choose which parts of an application should leverage the flexibility of schema-less JSON and which parts of an application should optimize the storage and compute in exchange for a fixed schema.

The fixed schema version from our previous example can be represented as schema-less JSON below:

```
Statement result res = KVStore.executeSync("CREATE TABLE user(id INTEGER,
                                        userData JSON primary key(id)))");
```

and JSON data can be inserted using:

```
Row r = Table.createRow().put("id", 123456).put("userData",
                                              FieldValueFactory.createValueFromJson(
       "{
          "firstName" : "John",
          "lastName" : "Doe"
           "addresses": [
           {"street" : "127 Spring St", "city" : "New York", "state" : "NY",
            "zip" : 10012, "addrType" : "work"},
           {"street" : "625 Ridgewood Rd", "city" : "Paramus", "state" : "NJ",
            "zip" : 07675, "addrType" : "home"}
         ]
       }");
       ));
```

JSON data can also be queried using the same SQL dialect that operates on Tables. Additionally, for ad-hoc JSON documents, several new operators and a "case" expression are introduced to aid in the processing of documents that may have disjoint or unknown formats.

1. Find all users in New York city. In this case, if the document has an array of addresses then apply the predicate to the array, otherwise apply the predicate directly to an attribute called "city" in the main document.

```
select
    firstName,
    lastName
from
     User u
where
     (case when u.addresses instanceof ARRAY then
      u.addresses[$element.city = "New York"] else
      u.city = "New York" end)
```

2. Find all users in in New York City when the JSON document contains a city attribute, in the document for the query predicate

```
select *
from User u
where
      (case when exists(u.address.city) then u.address.city = "New York"
```

Incorporating Oracle NoSQL Database into applications is straightforward. APIs for basic Create, Read, Update and Delete (CRUD) operations, as well as SQL access, are packaged in a single jar file. Applications can use the APIs or SQL queries from one or more client processes that access a stand-alone Oracle NoSQL Database server process, alleviating the need to set up multi-system configurations for initial development and testing.

Failure Resiliency

Oracle NoSQL Database exposes the concept of availability zones as fault isolation containers. An availability zone can take the form of a rack of servers, a top of rack switch, the floor of a building, or an entire data center. Zones are also characterized as primary or secondary, having primary zones available for master elections as well as quorum acknowledgment. Oracle NoSQL Database further permits the configuration of any number of primary and secondary zones. Once configured, the replicas for each shard in the data store are laid out across these zones such that failure of any one zone will limit (or contain) this fault so that quorum is always maintained by every shard in the datastore.

The Oracle NoSQL Database deploys a separate Administration Service which is highly-available. Consistent with the Oracle NoSQL Database "No Single Point of Failure" philosophy, the ongoing operation of an installation is not dependent upon the availability of the Administration Service. Thus, both the database and the Administration Service remain available during configuration changes.

Architecture

We present the Oracle NoSQL Database architecture by following the execution of a write operation through the logical components of the system and then discussing how those components map to actual hardware and software operation. We will create a record with key "Katana" and a JSON object as the value "{"a": "foo"}". Figure 3 depicts the method invocation putIfAbsent("Katana", "{"a": "foo"}.

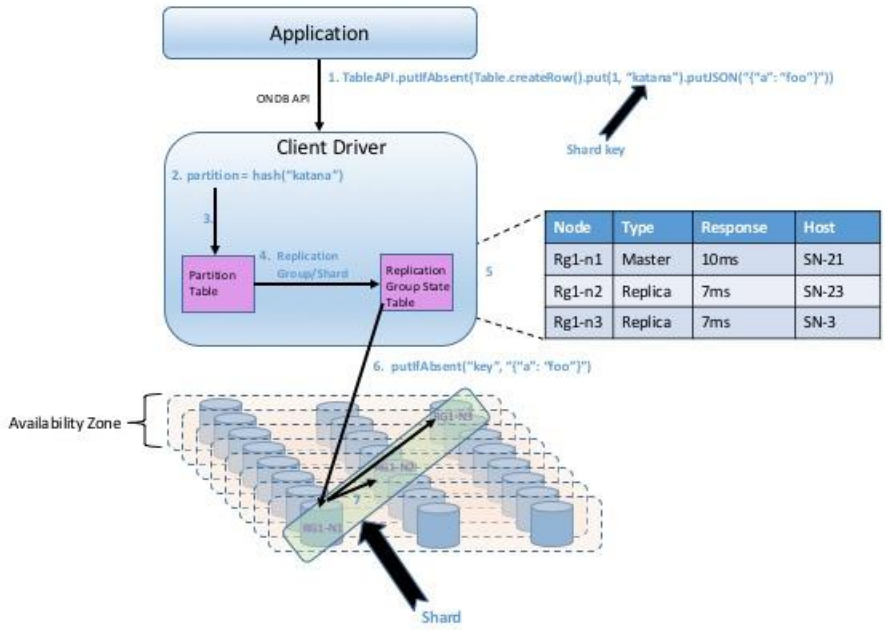

Figure 3 – Architecture - Oracle NoSQL Database – Write Request Flow

The application issues the putIfAbsent method to the Client Driver (step 1). The client driver hashes the key "Katana" to select one of a number of partitions (step 2). Each partition is assigned to a particular replication group (shard). The driver consults the partition table (step 3) to map the partition number to a shard.

A shard consists of some (configurable) number of replication nodes, and each replication node is resident in a separate availability zone (for fault containment). The number of replication nodes in a shard dictates the number of failures from which the system is resilient; a system with three nodes per shard can withstand two failures while continuing to service read requests. Its ability to withstand failures on writes is based on the requested durability policy in the write API call. If the application does not require a majority of participants to acknowledge a write, then the system can also withstand up to two failures for writes. A five-node group can withstand up to four failures for reads and up to two failures for writes, even if the application demands a durability policy requiring a majority of sites to acknowledge a write operation.

Given a shard, the Client Driver next consults the Replication Group State Table (RGST) (step 4). For each shard, the RGST contains information about each replication node comprising the group of replication nodes in the shard(step 5). Based upon the information in the RGST, such as the identity of the master and the load on the various nodes in a replication group, the Client Driver selects the node to which to send the request and forwards the request to the appropriate node (step 6). In this case, since we are issuing a write operation, the request must go to the master node.

The replication node then applies the operation. In the case of a putIfAbsent, if the key exists, the operation has no effect and returns an error, indicating that the specified entry is already present in the store. If the key does not exist, the replication node adds the key/value pair to the store and then propagates the new key/value pair to the other nodes in the replication group (step 7).

Implementation

An Oracle NoSQL Database installation consists of two major pieces: a Client Driver and a collection of Storage Nodes. As shown in Figure 3, the client driver implements the partition map and the RGST, while storage nodes implement the replication nodes comprising shards. In this section, we'll take a closer look at each of these components.

Storage Nodes

A storage node (SN) is typically a physical machine with local persistent storage, either disk or solid state, a CPU with one or more cores, memory, and an IP address. A system with more storage nodes will provide a greater aggregate throughput or storage capacity than one with fewer nodes, and systems with a greater degree of replication in replication groups can provide decreased request latency over installations with smaller degrees of replication. In addition, more SNs results in higher availability across the entire system.

A Storage Node Agent (SNA) runs on each storage node, monitoring that node's behavior. The SNA both receives configuration from and reports monitoring information to the Administration Service. The SNA collects operational data from the storage node on an ongoing basis and then delivers it to the Administration Service when asked for it.

A storage node serves one or more replication nodes. Each replication node belongs to a single replication group. The nodes in a single replication group all serve the same data. Each group has a dynamically elected master node that handles all data modification operations (create, update, and delete). The other nodes are read-only replicas but may assume the role of master should the master node fail. A typical installation uses a replication factor of three in the replication groups, to ensure that the system can survive at least two simultaneous faults and continue to service read operations. Applications requiring greater or lesser reliability can adjust this parameter accordingly.

Figure 4 shows an installation with 10 replication groups (0-9) (also known as a shard). Each replication group has a replication factor of 3 (one master and two replicas) spread across two data centers. Note that we can place two of the replication nodes in the larger of the two data centers and the last replication node in the smaller one. This sort of arrangement might be appropriate for an application that uses the larger data center for its primary data access, maintaining the smaller data center in case of catastrophic failure of the primary data center.

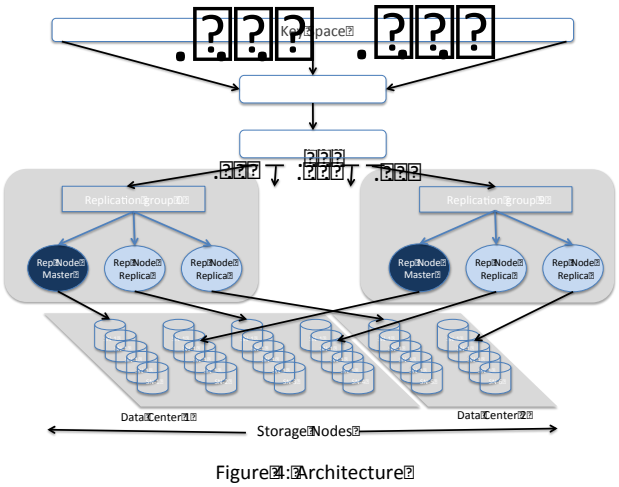

Figure 4: Architecture

Figure 4 – Architecture using 10 Shards and RF=3

The 10 replication groups are stored on 30 storage nodes, spread across the two data centers.

Replication nodes support the Oracle NoSQL Database API via RMI calls from the client and obtain data directly from or write data directly to the log-structured storage system, which provides high-performance writes, while maintaining index structures that provide low-latency read performance.

Oracle NoSQL Database uses replication to ensure data availability in the case of failure. Its single-master architecture requires that writes are applied at the master node and then propagated to the replicas. In the case of failure of the master node, the nodes in a replication group automatically hold a reliable election (using the Paxos protocol), electing one of the remaining nodes to be the master. The new master then assumes write responsibility.

Client Driver

The client driver is a Java jar file that exports the API to applications. The client driver is topology aware using the Replication Group State Table (RGST). The topology efficiently maps keys to partitions and from partitions to replication groups. For each replication group, it includes the hostname of the storage node hosting each replication node in the group, the service name associated with the replication nodes, and the data center in which each storage node resides. The client then uses the RGST for two primary purposes: identifying the master node of a replication group, so that it can send write requests to the master, and load balancing across all the nodes in a replication group for reads. Since the RGST is a critical shared data structure, each client and replication node maintains its copy, thus avoiding any single point of failure. Both clients and replication nodes run a RequestDispatcher that use the RGST to (re)direct write requests to the master and read requests to the appropriate member of a replication group.

The topology is loaded during client or replication node initialization and can subsequently be updated by the administrator if there are topology changes. The RGST is dynamic, requiring ongoing maintenance. Each replication node runs a thread, called the Replication Node State Update thread, that is responsible for ongoing maintenance of the RGST. The update thread, as well as the RequestDispatcher, opportunistically collect information on remote replication nodes including the current state of the node in its replication group, an indication of how up-to-date the node is, the time of the last successful interaction with the node, the node's trailing average response time, and the current length of its outstanding request queue. In addition, the update thread maintains network connections and re-establishes broken ones. This maintenance is done outside the RequestDispatcher's request/response cycle to minimize the impact of broken connections on latency.

Security

Oracle NoSQL Database can be configured securely. In a secure configuration, network communications between NoSQL clients, utilities, and NoSQL server components are encrypted using SSL/TLS, and all processes must authenticate themselves to the components to which they connect.

There are two levels of security to be aware of. These are network security, which provides an outer layer of protection at the network level, and user authentication/authorization. Network security is configured at the file system level typically during the installation process, while user authentication/authorization is managed through NoSQL utilities.

Oracle NoSQL Database offers following capabilities to secure the store and set password complexities:

- Security Configuration Utility. Allows for configuration to add security to a new or to an existing Oracle NoSQL Database installation.
- Authentication methods. Oracle NoSQL Database provides password authentication for users and systems. The Enterprise Edition version of Oracle NoSQL Database also supports Kerberos authentication.
- Encryption. Data is encrypted on the network to prevent unauthorized access to that data.

- External Password Storage. Oracle NoSQL Database provides two types of external password storage methods that can manipulate (one type for CE deployments). The types are clear text and using Oracle Wallet (available with the Oracle NoSQL Database Enterprise Edition).
- Security Policies. Oracle NoSQL Database allows to set up behaviors in order to ensure a secure environment.
- Role-based authorization. Oracle NoSQL Database provides predefined system roles, privileges, and user-defined roles to users. You can set desired privileges to users by role-granting.

Integration

Oracle NoSQL Database (Enterprise Edition version), as delivered, is ready to be integrated with various Oracle technologies such as the Oracle Database, Oracle Enterprise Manager, Oracle Coherence, Oracle Big Data Spatial and Graph, Oracle Rest Data Services, and open source technologies such as Hadoop, Hive and Spark.

Oracle Database Integration

To read data from Oracle NoSQL Database into an Oracle Database External Table, define an External Table with one or more location files; the location files will not actually contain any data, but instead contain configuration information related to connecting to the Oracle Database and Oracle NoSQL Database, query restrictions, and formatting.

Specifying multiple location files in the External Table definition allows for a degree of parallelism during data reading. Once the External Table is defined, run the Publish utility to "publish" information about the External Table and how to access the data in Oracle NoSQL Database. This information is written as an XML document into each of the location file(s).

When the publish is completed, a SQL SELECT may be executed against the External Table. This will cause the <KVHOME>/exttab/bin/nosql_stream script to be invoked, which in turn causes the Preproc class to be invoked. The preprocessor

- Reads the configuration and NoSQL Database access information from the location file,
- Reads the data from the NoSQL Database,
- Formats it using either a user defined formatter or a default format, and
- Writes it to stdout.

Hadoop, Hive and Big Data SQL Integration

Hadoop to Oracle NoSQL Database is supported by running Hadoop MapReduce jobs against the data that is stored in the Oracle NoSQL Database table. A custom InputFormat class defines the method of reading and writing the MapReduce input data. There is also a custom implementation of InputSplit that defines how to get the list of logical “splits” for the job and a custom implementation of the RecordRead class that does the work of locating and returning the Key/Value pairs used by MapReduce and SerDe.

Oracle Big Data SQL allows users to employ the SQL language to manage and manipulate data stored in a number of different locations in different databases. Oracle Big Data SQL achieves this by presenting Oracle NoSQL Database data as enhanced external tables to the Oracle Database. This is accomplished by mapping the external semantics for accessing data from these sources to the Oracle Database internal structures.

Oracle Big Data SQL not only enables easy integration of data from Hadoop and NoSQL sources, Big Data SQL also leverages the underlying storage mechanisms to provide the best possible performance. Big Data SQL’s predicate push down technology allows predicates in queries issued in Oracle Database to be executed by remote systems, and to be pushed into certain file formats.

Integration with Hive is achieved by running Hive queries using an SQL-like language called HiveQL (HQL) over the data that is stored in the Oracle NOSQL Database. A custom Hive TableStorageHandler defines the mechanism that is required by both Hive and Big Data SQL, and is used when executing a query against a table in the Oracle NoSQL Database store. In addition there is the custom ObjectInspector class that is used to translate the data types to the corresponding Oracle NoSQL Database data format.

To improve query performance, both Hive and Big Data SQL support a form of predicate pushdown where the client side frontend decomposes the WHERE clause of a query into column information and corresponding comparison operations. This results in the pushing of the resulting components to the database server side for processing. Big Data SQL uses the Hive interface to support predicate pushdown when executing a Big Data SQL query against KVStore table data. To support the predicate pushdown feature, there is a custom implementation of HiveStoragePredicateHandler that supports and defines the criteria for decomposition of predicates being pushed down into table scans and index scans.

Performance

We have experimented with various Oracle NoSQL Database configurations and presented a few performance results of the Yahoo! Cloud Serving Benchmark (YCSB), demonstrating how the system scales with the number of nodes in the system. As with all performance measurements, the results may vary due to a number of factors.

We applied a constant YCSB load per storage node to configurations of varying sizes. Each storage node was comprised of an Intel Xeon Processor X5670 dual socket machine with 6 cores/socket and 24 GB of memory. Each machine had a single local disk and ran RedHat 2.6.18-164.11.1.el5.crt1. At 300 GB, the disk size is the scale-limiting resource on each node, dictating the overall configuration, so we configured each node to hold 100M records, with an average key size of 13 bytes and data size of 1108 bytes.

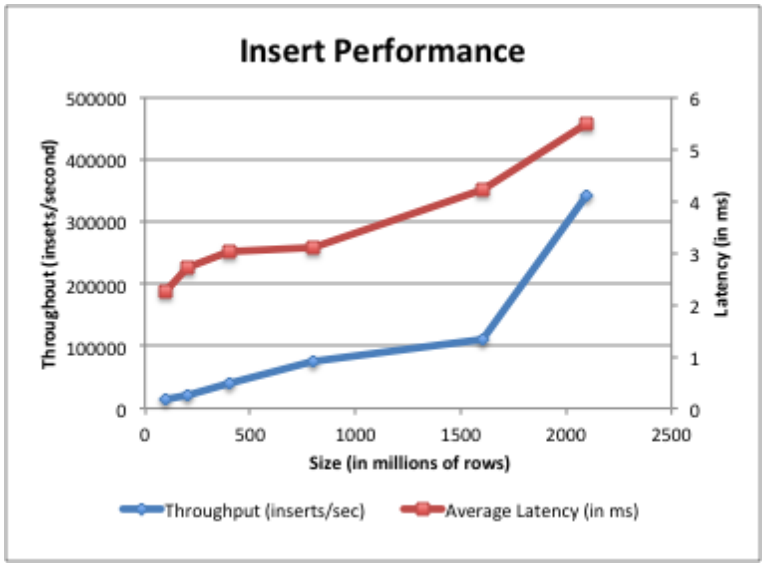

Figure 5 – YCSB Results – Performance as size of the database grows

Figure 5 shows the raw insert performance of Oracle NoSQL Database for configurations ranging from a single replication group system with three nodes storing 100 million records to a system with 32 replication groups on 96 nodes storing 2.1 billion records (the YCSB benchmark is limited to a maximum of 2.1 billion records). The graph shows both the throughput in operations per second (blue line and left axis) and the response time in milliseconds

(red line and right axis). Throughput of the system scales almost linearly as the database size and number of replication groups grows, with only a modest increase in response time.

Conclusion

Oracle's NoSQL Database brings enterprise quality storage and performance to the highly-available, widely distributed NoSQL environment. It is commercially proven, write-optimized storage system delivers outstanding performance as well as robustness and reliability, and its "No Single Point of Failure" design ensures that the system continues to run and that data remains available after a range of failure possibilities.

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

blogs.oracle.com/oracle

facebook.com/oracle

twitter.com/oracle

oracle.com

Integrated Cloud Applications & Platform Services

Oracle NoSQL Database
June , 2018 V7
Author: MS, MB, DR, TG, AC