Capacity planning for Oracle NoSQL Database Cloud Service

WHITE PAPER / FEBRUARY, 2020

ORACLE

Contents

INTRODUCTION

When you use Oracle NoSQL Database Cloud Service, you don't need to provision CPU, memory or storage IOPS, but instead you must specify your requirements for write, read and storage activity when creating the table (referred to as provisioning). Throughput is specified in terms of write and read units and storage is specified in gigabytes (GB). In the provisioning process, Oracle NoSQL Database Cloud Service reserves those resources to meet your application requirements, while ensuring sufficient, high performance storage.

It's important to properly evaluate how many write and read units you need to provision. Provisioning too few units may result in some of your requests getting throttled, and provisioning too many units may result in unused performance. This white paper describes how you should approach throughput estimation in Oracle NoSQL Database Cloud Service, and presents an example with a sample workload for estimating write and read units.

PROVISIONED THROUGHPUT UNITS

First, we need to define each of the units.

Write Units (WU)

One Write Unit (WU) is defined as the throughput of up to 1 KB of data per second for a write operation.

Read Units (RU)

One Read Unit (RU) is defined as the throughput of up to 1 kilobyte (KB) of data per second for an eventual consistent read operation (i.e., where the data returned may not be the most recently written data to the database; if no new updates are made to the data, eventually all accesses to that data will return the latest updated value). To achieve the throughput of up to 1 KB of data per second for an absolute consistent read (i.e., where the data returned is expected to be the most recently written data to the database), the equivalent of 2 Read Units would need to be provisioned.

For both RU and WU, if a record size is less than 1 KB, it is rounded up to 1 KB. For example, if it is 0.5 KB, then use 1 KB in the throughput calculation. If it is more than 1KB, then round up to the next whole KB (i.e. record size of 3.4 KB is rounded up to 4 KB).

Storage Units

A storage unit is a measure of storage capacity used. One “Gigabyte Storage Capacity per Month” is defined as a gigabyte (1073741824 bytes) of computer storage space used by a storage filer of the Oracle Cloud Service during a month of the service. One (1) gigabyte (GB) of computer storage provides space for user data and indexes.

Specify provisioned throughput units

Oracle NoSQL Database Cloud Service provides an API to specify the read and write throughput and storage capacity for a table. A TableLimits instance is used during table creation to specify those capacities/limits. This API takes as input the throughput units and storage unit.

For example, using the Java API to create a table with the limits set to 2000 RUs, 100 WUs and storage of 500 GB, the application would issue the following request:

```
TableRequest tableRequest = new TableRequest()

  .setStatement("create table if not exists user(id integer,

       FirstName string, LastName string, ZipCode integer,

       weight double, primary key(id))")

  .setTableLimits(new TableLimits(2000, 100, 500))

  .setTimeout(1000);

 TableResult res = conn.tableRequest(tableRequest);
```

Modifying Throughput Settings

The same TableLimits instance that is used during table creation is also used to change the limits of an existing table. This provides an easy and consistent interface. The operation is performed without any downtime, and typically takes effect as soon as the API call returns.

Factors that impact the capacity unit

Before you provision the capacity units, it’s important to consider the following variables that impact the read, write and storage capacities.

- Record size: As size increases, the number of capacity units consumed to write or read the data also increases.
- Data consistency: Eventual consistent reads are one half the cost of absolute consistent reads.
- Secondary Indexes: When a record in a table is added, updated, or deleted, updating the secondary indexes consumes write units. The total provisioned throughput cost for a write is the sum of write units consumed by writing to the table and those consumed by updating the local secondary indexes.
- Data modeling choice: With schema-less JSON, each document is self-describing which adds meta-data overhead to the overall size of the record. With fixed schema tables, the overhead for each record is exactly one byte.
- Query patterns: The number of query results, the number of predicates, the size of the source data, and projections all affect the cost of query operations. A query based on a full table scan would consume more throughput than a query that’s based on a key only scan (e.g. this query, with ID as the primary key, “*SELECT **ID** FROM T1 WHERE ID >1*” performs a

"key only" scan and consumes less throughput than this query "*SELECT * FROM T1 WHERE ID >1*" performing an index scan, but must read the entire record after scanning the index).

REQUEST UNIT CONSUMPTION FOR READ AND WRITE OPERATIONS DESIGN

While determining your read and write consumption needs, it is important to understand how different operations impact those read and write consumption needs.

Read Operations (Gets, Selects)

The following describes how Oracle NoSQL Database Cloud Service operations consume read units:

As mentioned previously, record sizes are rounded up to the next whole KB boundary.

Read operations are broken down into two buckets, reading a single record with a GET request and reading records with a SQL select query.

For every SQL select operation there can be a 2 KB fixed consumption cost for preparing the statement. If an explicit prepare statement is used, the cost is on that statement. This is the optimal approach. If the prepare statement does not exist, it is done internally and charged against the select.

For every SQL select operation there is at least 1 read consumed on an index read.

Through the use of the QueryRequest() API, two types of limits can be set, namely, one limiting the number of rows returned and another limiting the total data read. If a query returns more rows than the limit, each group returned is called a batch. If limits are set and triggered, there is a 1KB cost for each batch.

We will use the following table definition T1 to illustrate the concepts:

`T1 (id integer, primary key (id), c1 string, c2 string)` record size is 1.5KB

1. Get request example with reads:

```
MapValue key = new MapValue().put("id", 10);

GetRequest getRequest = new GetRequest()

            .setKey(key)

            .setTableName("T1");

GetResult getRes = handle.get(getRequest);
```

This will consume the following read units:

```
record size rounded up = 2 KB (record)
```

If using absolute consistent reads, the consumption would be:

```
Record size * 2 = 2 KB (record) * 2 = 4 KB
```

Using a GET is the most inexpensive read operation. The cost is determined by the size of the record rounded up, and the read consistency.

2. Query examples with reads

The consumption from a query depends on whether we are querying by primary key or by secondary key. In these formulas, we are assuming the optimal approach and have issued a prepare statement in advance. This 2KB cost is not figured into the select statement calculations below.

- Primary key case using the following query

```
SELECT * from t1 where id = 1
```

This consumes the following read units:

```
Record size  + 1 KB (for index read) =

2 KB (record) + 1 KB (index) = 3 KB
```

If using absolute consistent reads, the consumption would be `2 * 3 KB = 6 KB`

If we extend this example and instead of selecting 1 record, we go ahead and want all the records in the table then we would multiply the record and index parts by the number of records (assume 100 records).

```
2 KB (record) * 100 + 1 KB (index) * 100 = 300 KB
```

Extending this example further, if you do a full table scan and no records matched the predicates (returning 0 rows) there still would be 300 KB consumed because all the records needed to be looked at.

- Secondary key case

When querying with a secondary index many records may match a given secondary index entry. Much like we did for the full table scan, we need to multiply by the number of records that matched on the record and index.

Let's add a secondary index to our table:

```
Create index c_idx on table t1 (c1);
```

Let's use this query as our example and assume it returns 10 rows:

```
select * from t1 where c1 = "nosql";
```

This will consume the following read units:

```
Record size * (number of records) + 1 KB index * (number of
records)  =

2 KB (record) * 10 + 1 KB (index) * 10 = 30 KB
```

If there wasn't a secondary index on the table, this query would have resulted in a full table scan and would have consumed 300 KB (see above). Hence, you can lower your provisioned read unit costs by carefully examining your queries for the possibility of adding secondary indexes.

Write Operations (Puts, Deletes, Updates)

The following describes how Oracle NoSQL Database Cloud Service operations consume write units. In nearly all the formulas, absolute consistent reads are needed to guarantee we are reading the most recent record.

1. Put

   In a put operation, you account for the insert of the record as well as the index. The put operation can be performed conditionally or unconditionally. If done conditionally, then there are read units consumed in addition to write units. Those reads are done with absolute consistency so there is a multiplier of 2 in the formula. It is possible that a single put operation incurs several index operations and this needs to be taken into account.

   A. Unconditional – the formula is shown below

      Write KB `= record size + 1 KB (index) * Number of secondary index writes`

   B. Conditional (use of IfAbsent or IfPresent) - the formula is shown below. In the IfAbsent case, the "original record size" is zero. This is effectively an update statement. With 1 secondary index the "number of secondary index values changed" is 2 if using IfPresent and 1 if using IfAbsent.

      Write KB `= original record size + new record size + 1 KB (index) * Number of secondary index values changed`
      Read KB `= 1 KB (index) * 2`

2. Delete

   The default behavior for a delete is to delete the row if it exists. A delete incurs both read and write unit costs. Since we have to guarantee we are looking at the most current version of the row, absolute consistent reads are used, therefor the 2 mulitplier in the read unit formula. Several index writes are possible. The formulas are:

   Read KB = `1 KB (index) * 2`
   Write KB = `Record size + 1 KB (index) * Number of secondary index writes`

3. Update

   When rows are updated using a query (SQL statement), then both read and write units are consumed. Depending on the update it may need to read the primary key, secondary key, or even the record itself. Absolute consistent reads are needed.

   Read KB = `Record size * 2 + [1 KB (index)* number of secondary index reads] * 2`
   Write KB = `original record size + new record size + 1 KB (index) * Number of secondary index writes`

Example update:

Table Users (id integer, name string, age integer, primary key(id))
The table has index on name and age. Record size: 1 KB

Case 1: Modify users age

UPDATE Users u SET u.age = 31 WHERE u.id = 1;

This operation consumes the following:

Read KB = 1 KB * 2 + 1 KB (age index) * 2 = 4 KB
Write KB = 1 KB (old record) + 1 KB (new record) + 1 KB * 1(update to age index) = 3 KB

Case 2: Modify users name and age

UPDATE Users u SET u.age = 31 SET u.name = john.doe WHERE u.id = 1;

This operation consumes the following:

Read KB = 1 KB * 2 + 2 KB (age index and name index) * 2 = 6 KB
Write KB = 1 KB (old record) + 1 KB (new record) + 1 KB * 2(update to age index and name index) = 4 KB

EXAMPLE OF READ AND WRITE UNITS ESTIMATION

Let's take a real world example of how to estimate read and write capacities. In this example, we are using Oracle NoSQL Database Cloud Service to store a product catalog for an e-commerce application. The process of estimation is broken into the following steps:

Step 1: Identify the data model (JSON or fixed-table), record and key size for the application.

We are using a JSON model and have created a simple table with 2 columns: a record identifier (primary key) and JSON document for the product details. There is overhead in this case because each document is self-describing.

The JSON document, which is just under 1 KB in size when stored in the database because of compression, is shown in Appendix A.

Let’s assume this application has 100,000 records. The primary key is a unique identifier for the record and is 20 bytes in size. Prepare statements have been used, therefore the select statement will not have the 2 KB fixed cost.

Also, for this example some queries read records via a secondary index (i.e. *find all the records that have screen size of 13 inche*s). Thus, we need an index created on the screen-size field (one secondary index).

This information is summarized below:

Tables	Rows per table	Columns per table	Key Size bytes	Value Size bytes (sum all columns)	Indexes	Index Key Size bytes
1	100000	2	20	2 KB	1	20

Step 2: Identify the list of operations (typically CRUD operations and Index reads) on the table and at what rate (per second) these are expected:

Operation	Number of operations per sec
Create Records	3
Read the records via Primary Key *(i.e. read the product document via product id)*	300
Read the records via Secondary Index *(i.e. get all documents that have screen size of 13 inches)*	10
Update records (i.e. update the product description or add a new attribute)	5
Delete records	1

Step 3 Identify the read and write consumption in KB.

Operation	Read Consumption in KB	Write Consumption in KB
Create records	0 (not checking if the key exists)	Record size + 1 KB (index) *(number of indexes) = 1 KB + 1 KB (1) = 2 KB Let's assume the records are created without perform any conditional check.

GET by primary key	Record size = 1 KB	0
Query records by secondary index	[1 KB (index) + record size] * Number of records matched Let's assume 100 records are returned 100 * (1 KB + 1 KB) = 200 KB There's also some variable overhead depending on how many batches are returned and the size limit that is set for query Let's say this variable consumption is 10 KB So, the total KB consumption is: 200 KB + 10 KB = 210 KB	0
Update	1 KB (index) * 2 + Record size*2 = 2 KB + 2 KB = 4 KB	original record size + new record size + 1KB (index) (number of writes) 1 KB + 1 KB + 1 KB = 3 KB Let's assume the updated record is same size as old (1 KB)
Delete	1 KB (index) * 2 = 1 KB x 2 = 2 KB	Record size + 1 KB (Index)* (number of indexes) = 1 KB + 1 KB (1 index) = 2 KB

Then, using the above KB consumption, we determine read and write units for our workload that we identified in step #2 above

Operations	**#**	**Read Units/sec**	**Write Units/sec**
Create	3	0	6

Get by primary key	300	300	0
Query by Index	10	2100	0
Update	5	20	15
Delete	1	2	2
Total		2422	23

So, the table needs to be created with a provisioned throughput of **2422** Read Units and **23** Write Units.

Note: The preceding calculations assume eventual consistent read requests. For an absolute consistent read request, the read capacity units would be 4844 Read Units.

ESTIMATING THE MONTHLY COST

After calculating the capacity units, the next step is determine the cost. Oracle provides you with a cost estimator to figure out your monthly usage and costs before you commit to a subscription model or an amount.

The Cost Estimator automatically calculates your monthly cost based on your input of read units, write units, and storage.

Access the Cost Estimator on the Oracle Cloud website. Select the Data Management check box. Scroll through to locate Oracle NoSQL Database Cloud, and click Add to add an entry for Oracle NoSQL Database Cloud under the Configuration Options. Expand NoSQL Database to find the different Utilization and configuration options. Input values for the Utilization and Configuration parameters to estimate the cost for Oracle NoSQL Database Cloud Service usage from your Oracle Cloud Pay-As-You-Go and Monthly Flex subscriptions.

OPTIMIZE ORACLE NOSQL DATABASE CLOUD SERVICE PROVISIONED UNIT USAGE

Here is the recommended workflow to optimize Oracle NoSQL Database Cloud Service provisioned unit usage:

- Perform an initial, rough evaluation using the capacity calculator.
- Use the Oracle NoSQL Database Cloud Service API to benefit from automatic retries when requests get throttled and make sure that your application code gracefully supports the case when all retries fail. You can implement *oracle.nosql.driver.RetryHandler* to define the retry strategy. Then set the *RetryHandler* through *NoSQLHandleConfig.setRetryHandler.* When the request gets throttled, you will see *oracle.nosql.driver.ThrottlingException* which could be

further classified into three subtypes: *ReadThrottlingException, WriteThrottlingException, OperationThrottlingException.*

- Monitor throttling errors from the API. Start with conservative limits such as 10 throttled requests over the last 15 minutes and switch to more permissive rules once you figure out your actual consumption (remember that occasional throttles are fine, they show that you're close to the limits you've set and that's what you want to do). For monitoring you can use the class *TableUsageResult* API. In the *TableUsageRequest* you can specify *startTime()* and endTime() to return the usage for the specified time range. *TableUsageResult* contains a *TableUsage* array. Each entry in the *TableUsage* array represents the table usage stats during the *TableUsage.secondsInPeriod* from *TableUsage.startTimeMillis*. You can add up *TableUsage.readThrottleCount* to get a count of all of the throttling requests that you received in the last 15 minutes.

- Use the monitoring API to understand the traffic pattern, so you can consider the need to dynamically adjust your throughput provisioning over time, perhaps over a day or week. As stated earlier, there is an API available to monitor table usage stats through *TableUsageResult*. You could sum *readThrottleCount*, *writeThrottleCount* or *storageThrottleCount* over day/week to understand the traffic pattern.

- Regularly monitor the provisioned vs. consumed capacity unit ratio to make sure you have not over-provisioned your table.

- Scale your throughput elastically and on-demand. Monitoring the consumption of your request units and the ratio of throttled requests could reveal that you don't need to keep a constant performance level throughout the day or the week; many web applications receive less traffic at night or during the weekend.

SUMMARY

Oracle NoSQL Database Cloud Service is a fully managed database service that provides high performance with seamless scalability. It frees the IT department from the headaches of provisioning hardware and systems software, setting up and configuring a distributed database cluster, and managing ongoing cluster operations. There are no hardware administration costs since there is no hardware to maintain.

By identifying your workload, and identifying the typical operations that you need to perform, it is possible to estimate the read and write units that would needed for your workload.

APPENDIX A

Sample JSON record


```
{
   "additionalFeatures": "Front Facing 1.3MP Camera",
    "os": "Macintosh OS X 10.7",
   "battery": {
       "type": "Lithium Ion (Li-Ion) (7000 mAH)",
     "standbytime" : "24 hours"
   },
   "camera": {
       "features": [
           "Flash",
           "Video"
       ],
       "primary": "5.0 megapixels"
   },
   "connectivity": {
       "bluetooth": "Bluetooth 2.1",
       "cell": "T-mobile HSPA+ @ 2100/1900/AWS/850 MHz",
       "gps": true,
       "infrared": false,
       "wifi": "802.11 b/g"
   },
   "description": "Apple iBook is the best in class computer for your professional and personal work.",
   "display": {
       "screenResolution": "WVGA (1280 x 968)",
       "screenSize": "13.0 inches"
   },
   "hardware": {
       "accelerometer": true,
       "audioJack": "3.5mm",
       "cpu": "Intel i7 2.5 GHz",
       "fmRadio": false,
       "physicalKeyboard": false,
       "usb": "USB 3.0"
   },
   "id": "appleproduct_1",
   "images": [
       "img/apple-laptop.jpg"
   ],
   "name": "Myshop.com : Apple iBook",
```

```
    "sizeAndWeight": {
        "dimensions": [
            "300 mm (w)",
            "300 mm (h)",
            "12.4 mm (d)"
        ],
        "weight": "1250.0 grams"
    },
    "storage": {
        "hdd": "750GB",
        "ram": "8GB"
    }
}
```

ORACLE CORPORATION

Worldwide Headquarters
500 Oracle Parkway, Redwood Shores, CA 94065 USA

Worldwide Inquiries
TELE + 1.650.506.7000 + 1.800.ORACLE1
FAX + 1.650.506.7200
oracle.com

CONNECT WITH US

Call +1.800.ORACLE1 or visit oracle.com. Outside North America, find your local office at oracle.com/contact.

 blogs.oracle.com/nosql  facebook.com/oracle  twitter.com/oraclenosql

Integrated Cloud Applications & Platform Services

Oracle NoSQL Database Cloud V14
February 2020
Author: Anand Chandak, Michael Brey,

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