

New Oracle NoSQL Database APIs that Speed Insertion and Retrieval

ORACLE WHITE PAPER | FEBRUARY 2016



Introduction

Fast insertion and retrieval of data into a NoSQL database is of critical importance to a modern organization. As large amounts of data are created (think Big Data or Internet of Things), a reliable and industry-leading database must be able to respond to the demands of both the users who consume the information as well as the mechanisms that are creating the data.

In the following sections we discuss:

- Oracle NoSQL Database Summary and Motivation for a new API
- BulkPut API with example code
- BulkGet API with example code

Oracle NoSQL Database

Oracle NoSQL Database is a highly scalable, highly available, fault tolerant, “Always On” distributed key-value database that you can deploy on low cost commodity hardware in a scale-out manner. Use cases of Oracle NoSQL Database include Distributed Web-scale Applications, Real Time Event Processing, Mobile Data Management, Time Series and Sensor Data Management, Online Gaming, etc. Oracle NoSQL Database offers all the features that are common to a typical NoSQL product like Elasticity, Eventually Consistent Transactions, Multiple Data Centers, Secondary Indexes, Security and Schema Flexibility. The key differentiators include ACID Transactions, Online Rolling Upgrade, Streaming Large Object Support, availability on Engineered Systems, and Oracle Technology integrated.


Data can be modeled as relational-database-style tables, JSON documents, or key-value pairs. Oracle NoSQL Database is a sharded (shared-nothing) system that distributes the data uniformly across the multiple shards in the cluster, based on the hashed value of the primary key. Within each shard, storage nodes are replicated to ensure high availability, rapid failover in the event of a node failure, and optimal load balancing of queries. NoSQL Database provides Java, C, Python and node.js drivers and a REST API to simplify application development. NoSQL Database is integrated with a wide variety of related Oracle and open source applications in order to simplify and streamline the development and deployment of modern big data applications. NoSQL Database is dual-licensed and available as an open-source community edition as well as a commercially licensed Enterprise Edition.

Motivation

Our customers have often asked us “what’s the fastest and most efficient way to insert and retrieve large number of records in Oracle NoSQL database?”

Recently, a shipping company reached out to us with the specific requirement of using Oracle NoSQL database for their ship management application, which is used to track the movements of their container ships that move the cargo from port to port. The cargo ships are all fitted with GPS and other tracking devices that relay a ship’s location after a few seconds into the application. The application is then queried for:

- 1) The location of all the ships displayed on the map
- 2) A specific ship’s trajectory over a given period of time.



As the volume of the location data started growing, the company started finding it hard to scale the application and is now looking at a back-end system that can ingest this large data-set very efficiently.

Oracle NoSQL Database BulkPut

Historically, we have supported the option to execute a batch of operations for records that share the same shard key, which is what our large airline customer (Airbus) has done. They pre-sort the data by the shard key and then perform a multi-record insert for a batch of records that share the same shard key. Basically, rather than sending and storing one record at a time, they can send a large number of records in a single operation. This certainly saved network trips, but they could only batch insert records that shared the same shard key. With Oracle NoSQL Database release 3.5.2, we have added the ability to do a bulk insert records across different shards in parallel, allowing application developers to work more effectively with very large data-sets.

The BulkPut API is available for table as well as for the key/Value data model. The API provides significant performance gains over single row inserts by reducing the network traffic round trips as well as by doing ordered inserts in batch on internally sorted data across different shards in parallel.

BulkPut API

KV interface: Loads Key/Value pairs supplied by special purpose streams into the store.

```
public void put(List<EntryStream<KeyValue>> streams, BulkWriteOptions  
bulkWriteOptions)
```

Table interface: Loads rows supplied by special purpose streams into the store.

```
public void put(List<EntryStream<Row>> streams, BulkWriteOptions bulkWriteOptions)
```

streams the streams that supply the rows to be inserted.

bulkWriteOptions non-default arguments controlling the behavior the bulk write operations

Stream Interface:

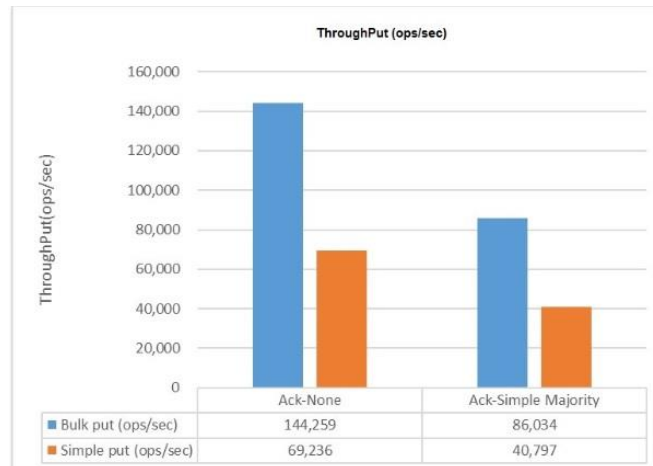
```
public interface EntryStream<E> {  
    String name();  
    E getNext();  
    void completed();  
    void keyExists(E entry);  
    void catch Exception (RuntimeException exception, E entry);  
}
```

BulkPut Performance

We ran [Yahoo Cloud Server Benchmark](#) (YCSB) benchmark internally to measure the performance of the new BulkPut API. For the performance test, we set up a 3x3 NoSQL cluster, - three shards each having three copies of data for scalability and high availability reasons. The cluster was set up to run on bare metal servers with uniform configuration – a total of nine servers (One NoSQL Storage Node per server) each configured with 250 GB RAM, 1TB HDD running Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz. The client machine also had similar hardware configuration.

For the workload distribution we used Zipfian distribution of keys, which is intended to model “real” loads. We ingested a total of 150 M records (50 M records per shard) across the datastore, using a total of nine parallel

threads (3 per shard) and a total of 54 input streams across nine storage nodes (6 per storage node). Parallel threads can be configured using `perShardParallelism` of `BulkWriteOptions` and `InputStream` can be configured using `streamParallelism` of `BulkWriteOptions`. The results for the benchmark run are shown in the graph below:



Oracle NoSQL database allows a configurable acknowledgment-based durability policy that describes whether the master node will wait for these acknowledgments before considering the write operation to have completed successfully. You can require the master node to wait for no acknowledgments, acknowledgments from a simple majority of replica nodes in primary zones, or acknowledgments from all replica nodes in primary zones. The more acknowledgments the master requires, the slower its write performance will be.

We ran the performance test with Durability settings for None (Ack-None) and Simple Majority (Ack-Simple Majority). The above graph compares the throughput (ops/sec) of BulkPut API and Simple Put API with NoSQL store having 1000 partitions for different durability settings. As seen from the above charts, there is over a 100% increase in throughput with either of the durability settings.

Sample Example on GitHub

An example of the BulkPut API can be found at: <https://github.com/swatianand/NoSQL>

The sample demonstrates how to use the BulkPut API in your application code. There's also a ReadMe file in the same repository. Refer to the readme file for details related to the program execution.

Oracle NoSQL Database BulkGet API

Customers continue to ask for fast and easy-to-use methods for retrieving data from a NoSQL database. An example of such a request would be on an eCommerce website, where potential customers want to retrieve all the phones in the price range of \$ 200 to \$ 500 from Apple, Samsung, Nokia, and Motorola (for example) and a host of other manufacturers to return all the details including the images of the products.

With release 3.4.7, we have introduced a high performance Bulk Get API, to retrieve records matching multiple primary keys in a single operation. For those of you who are familiar with SQL syntax, you can think of this to be similar to the IN clause that you supply to the SQL query. The API takes a list of keys, can be a partial or complete shard key, does a parallel scan across the shards for all keys in the list, and provides an iterator API over the

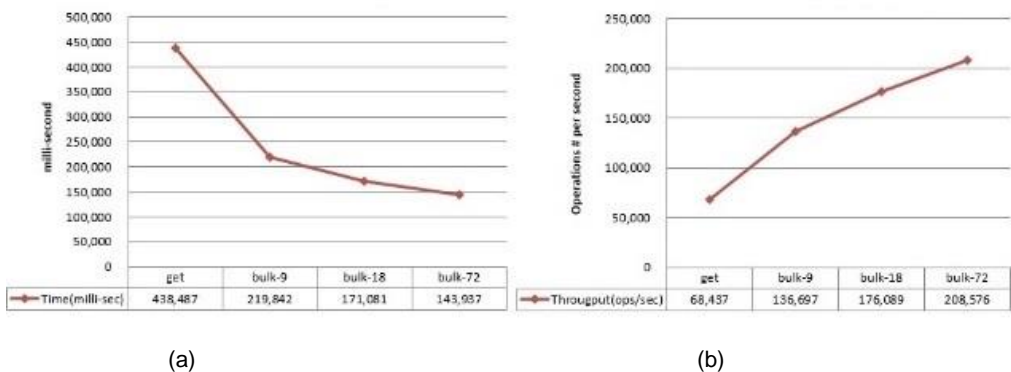
matching rows including those matched by the ancestor or descendant tables. The result is not transactional and the operation effectively provides read-committed isolation. The implementation batches the fetching of rows in the iterator, to minimize the number of network round trips, while not monopolizing the available bandwidth. Batches are fetched in parallel across multiple Replication Nodes, and the degree of parallelism is controlled by the *TableIteratorOptions* argument.

There are a few constraints the user should be aware of: 1) A primary key must contain all the fields defined for the table's shard key 2) The primary key should belong to the same table.

For more information, refer to the Java documentation for the API. We support both the key/value and table interfaces for the API.

Performance


In our internal [Yahoo Server Scalability Benchmark](#) (YCSB) runs we found that we could retrieve 30 M rows in 149 sec with 72 executor threads, running on 3x3 NoSQL cluster (3 shards each having 3 copies of data), with 90 reader threads (client-side threads) and each record size is 100 bytes. The number of executor threads can be configured by the *maxConcurrentRequests* parameter of the *TableIteratorOptions*. The hardware configuration of the machine is similar to those described in the BulkPut performance section described above. Refer to the chart below for details of the benchmark runs:



As seen from the timing graph (a) above, getting 30 M rows using simple get api would have taken us 420 seconds, which reduces to 149ms with 72 executor threads (Plotted on X-Axis) using the bulk get API. This is almost a 3X improvement! And as seen on the graph (b) the throughput went to 200K ops/s with 72 executor threads from 68k ops/sec using a simple get operation. That is again a **3X improvement!**

In the above charts, the bulk-X is the maximum number of concurrent request that specifies the maximum degree of parallelism (in effect the maximum number of NoSQL Client side threads) to be used when running an iteration. The optimal value for a parameter varies based on the nature of the application requirements -- some may want to be unobtrusive and use minimal resources (but efficiently) with elapsed time being a lower priority, e.g. running analytic on secondary zones, whereas some may want a strong real-time latency running multi-get on Primary Zones.

Sample Example on GitHub



An example of the BulkPut API can be found at: <https://github.com/swatianand/OracleNoSQLBulkGet>. The sample example demonstrates how to use the new BulkGet API in the phone example that's described above. The repository also contains a ReadMe file that describes the table and also the steps to run the example. The example returns an iterator over the keys matching the manufacturers within the price range [200-500] supplied by the iterator. If along with other details it also desired to retrieve the images of all the phones, then the images can be modeled as a child table (for efficiency reasons) and the same can be retrieved in the single API call.

Summary





Oracle NoSQL BulkGet API and BulkPut API provide the most effective and performant way to store and fetch the data in bulk from Oracle NoSQL database. As demonstrated by the YCSB runs, using this API you can expect between a 2 and 3 times performance improvement for retrieving data in bulk.



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

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group. 0116

February, 2016
Authors: Anand Chandak, Jin Zhao, Michael Schulman



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