Scalability Benchmarking: MongoDB and NoSQL Systems
Overview

In recent years alternatives to relational databases have emerged that provide advantages in terms of performance, scalability, and suitability for cloud environments. While they vary significantly in terms of capabilities, many in the industry have adopted the term “NoSQL” to describe these products. In this paper we evaluate the scalability of the three leading products – Cassandra, Couchbase, and MongoDB – using an industry standard benchmark created by Yahoo! called YCSB.

Selecting the appropriate database technology for a project involves careful consideration of many different criteria. While performance is important, it must be considered along with functionality, operational tooling, ease of use, availability of skills, technology ecosystem, security controls, reliability, and other factors. Our goal in this paper is to take a closer look at one of these criteria – scalability – and we encourage readers to use our findings as one of many inputs to their own evaluations.

Methodology

The three technologies we evaluate in this report provide very different feature sets. Comparisons are difficult, but each product provides a simple, core set of capabilities that can be used as the basis of a scalability evaluation.

In 2010, Yahoo! published a benchmark called the Yahoo! Cloud Serving Benchmark (YCSB). This benchmark tests CRUD operations. Many organizations have used YCSB to evaluate databases and different hardware environments. It has become popular and well understood.

We incorporated enhancements provided by each vendor for their client into our tests to prevent the fork of any single vendor from disadvantaging the other products. We also used the latest version of each database and drivers to ensure that the best possible performance for each product was observed.

YCSB is a framework and common set of workloads for evaluating the performance of different databases. YCSB consists of:

- The Client – an extensible workload generator
- The Core Workloads – a set of workload scenarios to be executed by the generator

The following diagram illustrates how YCSB works with a database:
We used YCSB as the basis for all our tests. We used the same number of records, record size, number of fields, and number of operations in all of our tests to provide a common baseline. For each of the tests, we performed multiple independent runs, increasing the number of client threads until the maximum throughput was reached without sacrificing latency. For all three products the ideal number of threads was around 150 for workload A, and around 350 for workload B. For measuring results, we recorded throughput and latencies for reads and updates (average, 95th percentile, and 99th percentile).

Our setup consisted of a database cluster of three servers and one dedicated YCSB server to ensure the YCSB client was not competing with the database for resources. All servers were identical.

We performed separate runs for each of the three configurations we tested, each of the YCSB workloads (Workload A, Workload B), and each of the thread counts we tested.

For each database we performed the following tasks to load the data:
- Start with an empty database instance (data files wiped, server restarted)
- Load 400M records using the “load” phase of YCSB
- Allow the database to stabilize after the load, including compaction

We then ran each workload and recorded the results. Before starting the next workload we allowed the database to stabilize.
Deployment Architecture

All three databases implement distributed architectures. While their designs follow very different approaches, all three databases provide automatic partitioning of data to support scale out, and all three databases rely on asynchronous replication to maintain multiple copies of the data for high availability. Furthermore, all three databases recommend multi-server environments for production deployments.

In these tests we deployed the databases across three bare metal servers. Data was partitioned across all three servers, and each database was configured to maintain two copies of the data.

Test Results

We evaluated two workloads using YCSB: Workload A, which consists of equal numbers of reads and updates, and Workload B, which consists of 95% reads and 5% updates. All tests were performed with 400M records. Because each system is also maintaining two copies of the data, each server is managing ~267M records, which represents a data set larger than RAM. Each test performs 100M operations and records throughput and latencies at the 95th and 99th percentiles for reads and updates separately.
Workload A (50% reads, 50% updates)

When testing a data set larger than RAM, the 50/50 workload in these tests demonstrates that MongoDB provides over 1.8x greater throughput than Cassandra, and nearly 13x greater throughput than Couchbase.

MongoDB provides single-digit millisecond latency at the 99th percentile for reads and updates. Cassandra’s latency is slightly better for updates, but worse for reads, which is to be expected as it is a write-optimized database. We observed surprisingly high latency for Couchbase, which was several times higher than either Cassandra or MongoDB:

<table>
<thead>
<tr>
<th>YCSB (Latencies) – Workload A</th>
</tr>
</thead>
<tbody>
<tr>
<td>99th (Read)</td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td><strong>Cassandra</strong></td>
</tr>
<tr>
<td><strong>Couchbase</strong></td>
</tr>
<tr>
<td><strong>MongoDB</strong></td>
</tr>
</tbody>
</table>
Workload B (95% reads, 5% updates)

When testing a data set larger than RAM, the read-heavy workload (95% reads) shows MongoDB again provides over 1.75x the throughput of Cassandra, and over 6x the throughput of Couchbase.

In this workload the latencies for Cassandra and MongoDB are higher than the 50/50 workload, and Couchbase exhibits the highest latencies overall:

<table>
<thead>
<tr>
<th>YCSB (Latencies) – Workload B</th>
<th>99th (Read)</th>
<th>99th (Update)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cassandra</strong></td>
<td>16ms</td>
<td>8ms</td>
</tr>
<tr>
<td><strong>Couchbase</strong></td>
<td>25ms</td>
<td>53ms</td>
</tr>
<tr>
<td><strong>MongoDB</strong></td>
<td>12ms</td>
<td>12ms</td>
</tr>
</tbody>
</table>
Conclusions

Based on these tests, we have reached the following conclusions:

- MongoDB provides greater performance than Cassandra and Couchbase in all the tests we ran, by as much as 13x.
- Cassandra and MongoDB exhibit very low latency at the 99th percentile for reads and updates in both workloads. Couchbase exhibits significantly higher latency.
- The tests show that in deployments where data exceeds the size of RAM, where data is partitioned across multiple servers, and where data is replicated for high availability, MongoDB provides better performance than Cassandra and Couchbase.
- Insofar as these tests only evaluate the most basic operations of these databases, users should also consider the other features that are important to their application, and they should develop tests that evaluate those capabilities as well.

Environment

These tests were conducted on bare metal servers. Each server had the following specification:
- CPU: 2 x 3.0GHz Intel Xeon(R) CPU E5-2690 v2-DecaCore IvyBridge
- RAM: 96GB: 6x16GB Kingston 16GB DDR3 2Rx4
- Disk: (2) 960GB SanDisk CloudSpeed 1000 SSD
  - drive controller: Adaptec 71605
- OS: Ubuntu 14.10
- Network: 10gigE

The following configurations were made to optimize the environment for each database:
- readahead was reduced to 64 blocks
- transparent huge pages, defrag, numa were disabled

The following YCSB code was used for these tests:
- [https://github.com/usaindev/YCSB](https://github.com/usaindev/YCSB)

Versions of each database:
- Cassandra 2.1.4
- Couchbase 3.0.2
- MongoDB 3.0.3

Driver versions:
• Cassandra: CQL cassandra-driver-core 2.1.5  
• Couchbase: 2.1.2  
• MongoDB: mongo-java-driver 3.0.0

All settings were default for all three databases. The following configurations were made:
• For Cassandra, the commitlog was placed on a separate device  
• For MongoDB the journal was placed on a separate device  
  ○ --storageEngine=wiredTiger  
  ○ zlib compression was enabled for usertable collection.

YCSB test configuration
• 400 million documents  
• 2 copies of the data  
• 100 Million operations  
• Zipfian request distribution  
• In some configurations where running 100 million operations was prohibitively slow,  
  the workload was run with additional parameter “maxexecutiontime=3600” which  
  limited the run to 60 minutes.