Comparing NoSQL Solutions In a Real-World Scenario: Aerospike, Cassandra Open Source, Cassandra DataStax, Couchbase and Redis

Composed by Avalon Consulting, LLC

June 2015

Introduction

Specializing in cloud architecture, Emind Cloud Experts is an AWS Advanced Consulting Partner and a Google Cloud Platform Premier Partner that assists enterprises and startups establish secure and scalable IT operations. The following benchmark employed a real-world use case from an Emind customer. The Emind team was tasked with the following high-level requirements:

- Support a real-time voting process during massive live events (e.g. televised election surveys or “America Votes” type game shows)
- Keep voters’ data anonymous but unique
- Ensure scalability to support surges in requests

Why Cloud-Based?

During a voting event, concurrent voting app users could spike to the 10’s of millions. On the other hand, when there are no voting events, usage is practically non-existent. Therefore, it is important to set up a cloud infrastructure that can be quickly built up to handle usage spikes, and quickly torn down once a voting event ends. Efficiently scaling the app's cloud infrastructure up and down to match an event's time schedule can dramatically reduce cost. Thus a cloud-based infrastructure is highly suitable for this task.

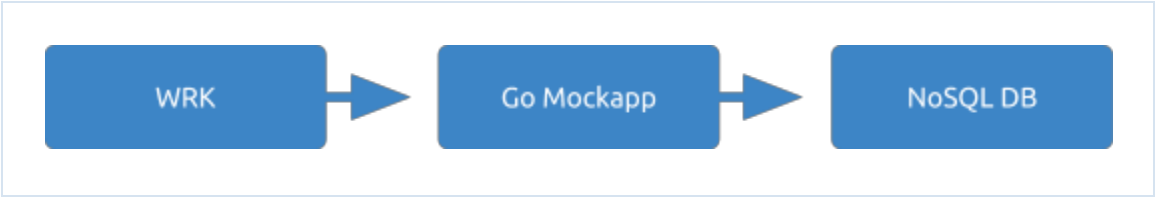
Why NoSQL?

Since scalability and cost effectiveness were considered of utmost importance in this project, Emind decided to use a NoSQL-type database to meet application requirements rather than a relational database. All the NoSQL offerings considered for this application provide efficient horizontal scaling that enables operators to quickly add or remove resources in response to the sudden spikes inherent in a voting application.

Application Flow

High performance is vital to optimize the apps' user experience. The application flow starts on a client device, which makes a call to a stateless application server and sends the user's vote. The application server sends the vote to the database for recording. During this flow, it is important that throughput remain high enough to handle usage spikes, while ensuring low latency.

The following figure represents the benchmark application flow:



Each test load was configured to run for 60 seconds, creating 35 threads and 5000 connections.

The following is the `wrk` command used to execute each test:

```
wrk -c 5000 -t 35 -d 60 "http://172.31.62.126/vote?u=1&v=y”
```

The NoSQL Candidates

The following NoSQL databases were tested:

- Aerospike
- Cassandra Open Source
- Cassandra DataStax Enterprise In Memory
- Couchbase
- Redis Enterprise Cluster

A Go mockup application, "mockapp" was created to simulate the voting app (https://github.com/emind-systems/real_time_vote_benchmark).
The mockapp was responsible for providing stateless application server endpoints, and for sending voting requests to each NoSQL database being benchmarked. The wrk (https://github.com/wg/wrk) HTTP benchmarking tool was used to generate load on each application setup. It captured throughput and latency metrics to determine which database was best suited for the scenario’s requirements.

Finally, Avalon Consulting, LLC was hired to work with each vendor (refer to Appendix A for the introductory email sent to vendors) in order to determine the optimal configuration for each NoSQL database solution and to validate the results by re-running all tests. All vendors responded, which ensured that each system was configured correctly.

The following precautions were taken to fully optimize each product:

Aerospike

- As recommended by the vendor, Avalon referred to its documentation for optimal set-up and configuration of the system.

Cassandra Open Source and DataStax Enterprise In-Memory

- DataStax recommended that Cassandra DataStax Enterprise In-Memory be implemented. However, the company raised a concern that the benchmark scenario is not optimal for this database.
- Consequently, Avalon decided to test both Open Source Cassandra and Cassandra DataStax Enterprise In-Memory.

Couchbase

- In order to avoid disk IO and obtain the best performance, Avalon verified that the entire Couchbase dataset fit into RAM.

Redis

- As recommended by the vendor, Avalon referred to its documentation for optimal set-up and configuration of the system.

Benchmark Methodology

Setup

Each test run utilized 3 servers:

- WRK Server – used to run the workload and send requests to the App Server:
    - c4.large EC2 instance
- App Server – used to run the Go HTTP server plus the Mockapp server software:
    - c4.8xlarge EC2 instance
    - During initial testing with c4.large, it was clear that the client server was going to be a bottleneck. Instead of scaling out with more servers, Avalon determined that the larger c4.8xlarge would be able to handle the load without additional servers.
- Database Server – used to run each NoSQL database:
    - c4.large EC2 instance

Database Configuration

For all systems, each database server had enough RAM to store the entire dataset in memory. The exact database configuration for each of the vendors was:

Aerospike

- Version: 3.5.9
- OS: Amazon Linux 2015.03
- AWS disks initialized (or pre-warmed) using the dd tool

Cassandra Open Source

- Version: 2.1.3
- OS: Ubuntu 14.04

Cassandra DataStax Enterprise In-Memory

- Datastax Enterprise 4.6.5
- OS: Ubuntu 14.04
- Configured for MemoryOnlyStrategy

Couchbase

- Version: Couchbase Enterprise 3.0.3
- OS: Ubuntu 14.04
- Replicas disabled

Redis Enterprise Cluster

- Version: 0.99.7-3
- OS: Ubuntu 14.04

Benchmark Process

Here is how the overall benchmark was conducted:

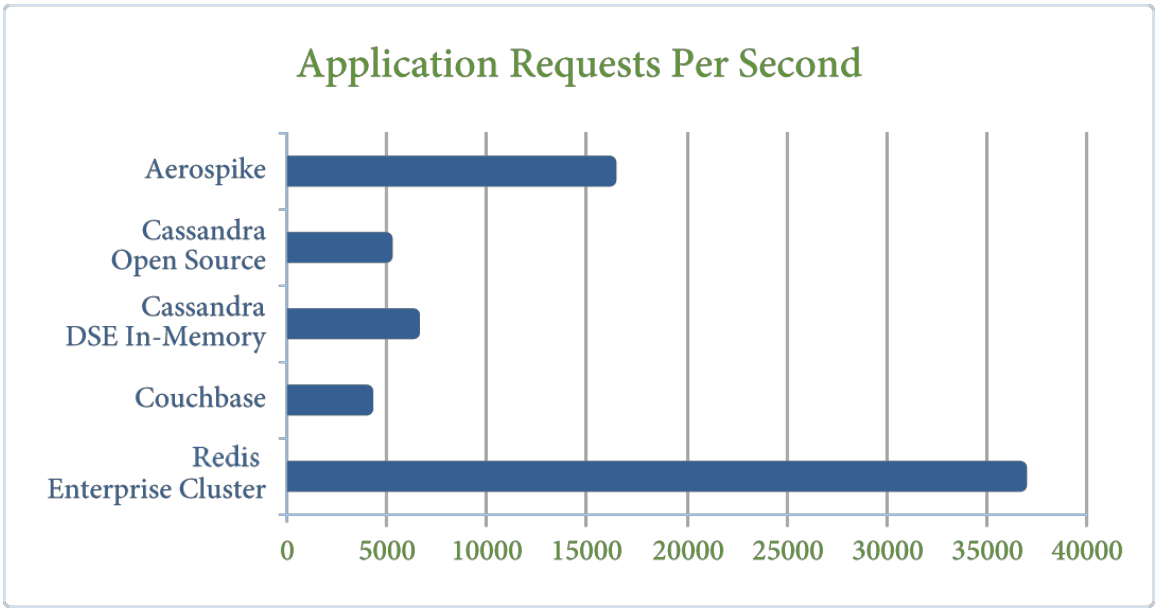
- Emind set up the initial infrastructure for the benchmark. This included setting up the servers for running wrk, the Go HTTP code and each NoSQL database.
- Emind wrote the initial code for running the HTTP server, capturing user input, and writing that user input to each database.
- Avalon Consulting, LLC reviewed and optimized the initial code, making changes where necessary.
- Avalon Consulting, LLC reviewed each server setup. This process included communicating with each vendor to determine optimal settings.
- Avalon Consulting, LLC reviewed the code for running each HTTP server to ensure it was as similar as possible for each system.
- Avalon Consulting, LLC executed every test in each newly configured environment.

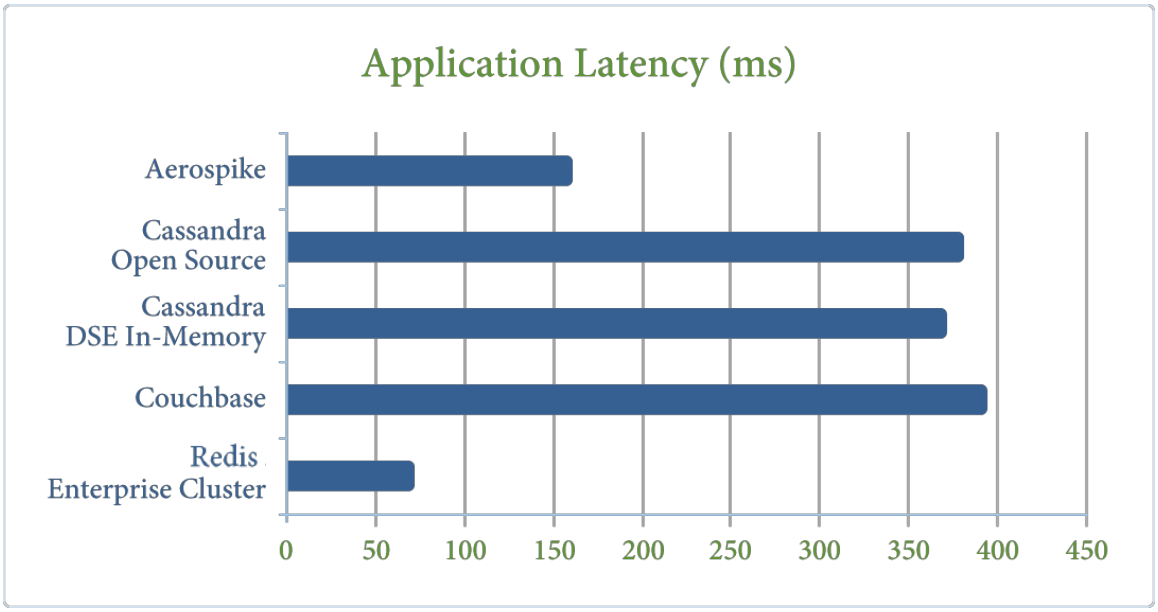
Final Results

Summary of Application Throughput and Latency

Product	App Req/Sec	App Latency(ms)
Aerospike	16481.55	161.94
Cassandra Open Source	5234.52	381.31
Cassandra DSE In-Memory	6659.26	372.31
Couchbase	4417.15	394.42
Redis Enterprise Cluster	37038.82	71.22

** Raw data results are provided in Appendix B*


Application Requests Per Second
Aerospike
Cassandra Open Source
Cassandra DSE In-Memory
Couchbase
Redis Enterprise Cluster
0
5000
10000
15000
20000
25000
30000
35000
40000


Application Latency (ms)
Aerospike
Cassandra Open Source
Cassandra DSE In-Memory
Couchbase
Redis Enterprise Cluster
0
50
100
150
200
250
300
350
400
450

Summary

This benchmark determined which NoSQL database would perform best with the dynamic load of a voting application scenario. For this particular workload, in which a large amount of 'write' requests were made, Redis Enterprise Cluster was the clear winner, Aerospike came in second with about half the throughput and double the latency, and the rest of the NoSQL databases were way behind. The test results are also applicable to other use cases where massive and continuous 'write' operations are required, such as ingestion data-store for a real-time analytics application accessed by API, or for an Internet of Things (IoT) application that receives telemetry data from a large amount of sensors.

Appendix A

Intro Email to Vendors

Dear Vendor,

I am with Avalon Consulting, LLC. We are known for providing clients a superior engagement experience through a combination of business acumen, intellectual curiosity, collaborative work style, and strong partnerships with award-winning vendors. In an effort to better understand the performance of cloud database servers available on the market we are in the process of benchmarking the performances a few of the cloud database servers; Couchbase, Aerospike, Cassandra and Redis Labs. We would like your help to ensure we configure your product properly to obtain the best results. We are hoping to have the configuration setup and start running the test fairly quickly. Below you will find the planned Amazon Web Services (AWS) configuration.

Can you please provide us with engineering assistance to properly configure your product and achieve the best result based on your recommendation?

Amazon Web Services Server Configurations

- *Your Database Server* : **c4.large** - 2 CPUs w3.75GB RAM - Moderate Network Performance and 500 Mbps Dedicated EBS Throughput
- *Application Server* - **c4.8xlarge** - 36 CPUs w60 GB RAM - 10Gbps Network Performance and 4,000 Mbps Dedicated EBS Throughput
- *WRK* (HTTP Benchmarking Tool) - **c4.large** - 2 CPUs w3.75GB RAM - Moderate Network Performance and 500 Mbps Dedicated EBS Throughput (2 Instances)

We have also recently conducted benchmark tests between MongoDB 3.0 and Couchbase Server 3.0.2. See the results here MongoDB 3.0 with Wired Tiger: New Benchmark Measures Performance vs. Couchbase Server 3.0.2. This benchmark test was also mention in a story published by *Information Week* titled Couchbase Claims Performance Gains Against NoSQL Rivals which ran on March 23, 2015. We would greatly appreciate your help. Thank you in advance.

Appendix B

Raw Data Results

Aerospike

```
Running 1m test @ http://172.31.62.126/vote?u=1&v=y
  35 threads and 5000 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   161.94ms  123.90ms   1.99s    67.74%
    Req/Sec   475.56    386.61    14.18k    77.07%
  990530 requests in 1.00m, 99.19MB read
  Socket errors: connect 0, read 0, write 0, timeout 56
Requests/sec:  16481.55
Transfer/sec:      1.65MB
```

Cassandra Open Source

```
Running 1m test @ http://172.31.62.126/vote?u=1&v=y
  35 threads and 5000 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   381.31ms  466.79ms   2.00s    83.33%
    Req/Sec   174.43    160.60     1.75k    73.46%
  314594 requests in 1.00m, 31.50MB read
  Socket errors: connect 0, read 0, write 0, timeout 16849
Requests/sec:   5234.52
Transfer/sec:    536.74KB
```

Cassandra DataStax Enterprise In-Memory

```
Running 1m test @ http://172.31.62.126/vote?u=1&v=y
  35 threads and 5000 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   372.31ms  390.91ms   2.00s    83.05%
    Req/Sec   204.09    153.23     1.15k    66.63%
  400219 requests in 1.00m, 40.08MB read
  Socket errors: connect 0, read 0, write 0, timeout 6826
Requests/sec:   6659.26
Transfer/sec:    682.83KB
```

Couchbase

```
Running 1m test @ http://172.31.62.126/vote?u=1&v=y
  35 threads and 5000 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   394.42ms  469.91ms   2.00s    82.95%
    Req/Sec   187.50    153.00     1.76k    64.79%
  265463 requests in 1.00m, 26.58MB read
  Socket errors: connect 0, read 0, write 0, timeout 11358
Requests/sec:   4417.15
Transfer/sec:    452.93KB
```

Redis Enterprise Cluster

Running 1m test @ http://172.31.62.126/vote?u=1&v=y

```
  35 threads and 5000 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    71.22ms   43.85ms 661.01ms   73.02%
    Req/Sec     1.06k   732.44     9.92k    75.22%
  2225247 requests in 1.00m, 222.96MB read
Requests/sec:  37038.82
Transfer/sec:      3.71MB
```