Performance Analysis: Benchmarking a NoSQL Database on Bare-Metal and Virtualized Public Cloud

Aerospike NoSQL Database on Internap Bare Metal, Amazon EC2 and Rackspace Cloud

Cloud Spectator – Comparative Performance Report
July 2014

Report Commissioned by Internap 2014

Table of Contents

Abstract

NoSQL databases are now commonly used to provide a scalable system to store, retrieve and analyze large amounts of data. Most NoSQL databases are designed to automatically partition data and workloads across multiple servers to enable easier, more cost-effective expansion of data stores than the single server/scale up approach of traditional relational databases. Public cloud infrastructure should provide an effective host platform for NoSQL databases given its horizontal scalability, on-demand capacity, configuration flexibility and metered billing; however, the performance of virtualized public cloud services can suffer relative to bare-metal offerings in I/O intensive use cases. Benchmark tests comparing latency and throughput of operating a high-performance in-memory (flash-optimized), key value store NoSQL database on popular virtualized public cloud services and an automated bare-metal platform show performance advantages of bare-metal over virtualized public cloud, further quantifying conclusions drawn in prior studies. Normalized comparisons that relate price to performance also suggest bare metal with SSDs is a more efficient choice for data-intensive applications.

Introduction

Effectively managing huge amounts of data has become indispensable for companies delivering applications over the Internet. The Internet's broad network access can quickly expand the potential audience for applications into the millions and their associated attributes into the trillions. Massive user pools, time sensitivity and an increasing volume of data that don't fit into typical organizational schemas – cookie information, social media blurbs, security camera images, weather reports and the like – have necessitated more flexible tools and methods for handling and analyzing data.

Fortunately, a number of technologies developed within the past decade including parallel processing systems, distributed databases and file structures as well as programmatically available virtual and bare metal infrastructure, have enabled companies to address these challenges. NoSQL ("Not Only SQL") databases in particular have expanded the organization's ability to capture, understand and act on information gathered under these demanding circumstances.

The distributed, horizontally scalable nature of NoSQL databases seemingly makes them a good fit for public cloud services. Past studies however, have documented a performance disparity for virtualized Infrastructure-as-a-Service (IaaS) compared with traditional bare-metal hosts for I/O-intensive workloads. In recent years, a few providers have deployed bare-metal cloud platforms that offer many of the automation and scalability features of virtualized IaaS. This paper tests this performance differential premise by benchmarking operating speed, throughput and costs of operating a high-performance in-memory, key value store NoSQL database on popular virtualized public cloud services and an automated bare-metal cloud platform.

Background

NoSQL refers to a class of database technologies that developed in response to the inability of traditional relational databases to effectively handle a rapid increase in the amount and variety of data organizations store and analyze. Relational databases maintain rigid schemas with many connected tables, each with rows and columns that assign a set of attributes to each collected record. Tables and their attributes are associated with each other by assigning related keys. When data is read from or written to relational databases, numerous tables must be accessed and executed before being passed along to the application or stored – resulting in many dependencies, serial processes and potentially long processing times. Expanding infrastructure to support rapidly growing relational databases is typically achieved by scaling vertically (i.e., adding processing power, RAM and storage to the machines that support the database).

NoSQL databases by contrast, are often object or key/value stores that group attributes and data into columns. This approach provides much less structure but is typically faster at performing reads and writes, especially when handling large amounts of data. In addition, NoSQL databases support horizontal scaling through automatic partitioning or "sharding" of the database across an arbitrary number of machines. While sharding is possible with relational databases, NoSQL databases make this horizontal scaling process quicker and easier given the absence of dependencies and automation that balances data load and query across servers.

Previous reports on the performance of NoSQL databases in particular infrastructure environments include a note titled "A Vendor-independent Comparison of NoSQL Databases: Cassandra, HBase, MongoDB, Riak"[1] by Altoros Systems, which provided a useful look into how to properly perform benchmark testing on NoSQL databases. The Altoros report focused on comparing different NoSQL databases while running on identical hardware on Amazon Web Service's public cloud compute service EC2. A more recent report titled Ultra-High Performance NoSQL Benchmarking[2] by Thumbtack Technology examined similar benchmarking using bare-metal hardware for NoSQL database clusters.

A number of papers have also examined the overhead associated with virtualized environments relative to bare metal. These studies point to virtualized network contention due to user concurrency as well as hypervisor burdening caused by guest/host context switches that virtual machines must undergo to securely obtain access to critical hardware functions.[3]

Our study builds on this work by quantifying performance differences when operating an in-memory NoSQL database in both virtualized public cloud and automated bare-metal cloud environments.

[1] http://www.altoros.com/vendor_independent_comparison_of_nosql_databases.html

[2] http://www.aerospike.com/wp-content/uploads/2013/01/Ultra-High-Performance-NoSQL-Benchmarking.pdf

[3] Some examples of this research include:
http://www.dcs.warwick.ac.uk/~sdh/pmbs11/PMBS11/Workshop_Schedule_files/rn-0900.pdf
http://gac.udc.es/~juan/papers/fgcs2013.pdf
http://researcher.ibm.com/researcher/files/il-ABELG/eli_asplos12.pdf

Methodology

NoSQL Database Selection

We employed Aerospike's in-memory key value store to simulate a range of low-latency, data-intensive use cases. Because of its design, Aerospike is often used as a reliable cache and a user store to address real-time data collection, distribution and analysis requirements common with certain Internet-centric businesses (e.g., advertising technologies, e-commerce, online trading and bidding, online gaming). Aerospike can run in pure RAM, but is the first to be uniquely optimized for use with solid-state hard drives (SSDs). For predictable low latency, Aerospike indexes are always stored in RAM, but individual tables or namespaces in Aerospike can be stored in either RAM or directly on SSDs and data on SSDs is accessed in parallel via a proprietary flash optimized file system. Aerospike clusters replicate data synchronously and unlike popular NoSQL databases, Aerospike supports immediate consistency and ACID properties. For this report, we used Aerospike with SSDs and 2x replication.

Benchmark Tool Selection

The Yahoo! Cloud Serving Benchmark (YCSB) was used to generate the testing datasets and evaluate performance of the databases. The YCSB, developed by Yahoo! Labs, is considered a standard benchmark for testing NoSQL tools. However, the version of YSCB used in this study has been updated since the initial release in 2010. Cloud Spectator originally used Thumbtack's version of the benchmark, with the Aerospike plugin, but we found additional updates were needed to support the latest version of Aerospike (Aerospike 3). Changes to the YCSB Aerospike 3 plugin can be found in *Appendix C*.

Host Selection

The underlying hardware for the database clusters and client machines for simulating users were hosted on three different providers. Internap bare metal was selected as the baseline for the hosted Aerospike database given Aerospike's high performance on bare metal demonstrated by the Thumbtack benchmark conducted in 2013. Amazon EC2's dedicated, I2 Storage Optimized instances and Rackspace's Performance Cloud Servers were chosen as virtualized public cloud comparables due to their popularity for Aerospike users and high level of performance among cloud providers.

Internap
Internap, which commissioned this study, provided the bare-metal servers with SSDs for the comparison with the virtual machine instances. Bare-metal servers were used in both the database cluster and client machines. The size of the database cluster servers was determined by running various configurations on Internap to determine the corresponding performance thresholds. The lowest performing bare metal server was used to attempt to match up against a similarly sized and/or priced cloud server. While Internap has bare-metal cloud configurations that are billable by the hour, the servers used in this study were billed by the month. Internap does not currently offer the selected server size at an hourly billing rate as it was chosen to parallel the size/price of the cloud instance comps rather than for its hourly availability.

Amazon Web Services
Amazon is an obvious choice to compare against when benchmarking cloud providers. Due to its dominant position in the market and wide variety of instance types, there are several suitable hardware resources to compare against for this study.

- The instance type, the I2 class, is storage-optimized and recommended for use in NoSQL databases. The I2 also comes with enhanced networking performance for higher transfer speed and lower latency.
- Cloud Spectator chose Dedicated Instances for Amazon EC2 to ensure isolation of hardware for highest possible performance.

Rackspace

The Rackspace Performance 2 Cloud Servers were chosen due to Cloud Spectator's previous performance testing that has shown high disk performance on their servers.

- Of the Rackspace servers tested, the 120GB Performance 2 servers had the highest throughput in the performance of the load phase and workload a; thus, the 120GB instance size was selected.

See Appendix A for Database and Client Configurations and Appendix B for Testing Process details.

Summary of Findings

The result of the study is that a four node cluster of bare-metal servers with SSDs indeed offer higher performance than virtualized cloud servers. Bare metal outperformed both Amazon and Rackspace virtual public clouds across all three different workloads in both throughput and latency. Amazon outperformed Rackspace across all three workloads in throughput speed, but Rackspace had lower latency in two of the tests.

In the Inserting Data Workload (100% Inserts)

- Internap outperformed Amazon by 51% in throughput speed
- Internap outperformed Rackspace by 5x in throughput speed
- Amazon outperformed Rackspace by 3.3x in throughput speed

- Internap has 77% less latency than Rackspace
- Internap has 56% less latency than Amazon
- Amazon has 49% less latency than Rackspace

In the Balanced workload (50% Read, 50% Update)

- Internap outperformed Amazon by 50% in throughput speed
- Internap outperformed Rackspace by 2.7x in throughput speed
- Amazon outperformed Rackspace by 78% in throughput speed

- Internap has 59% less latency than Amazon
- Internap has 32% less latency than Rackspace
- Rackspace has 39% less latency than Amazon

In the Read Heavy workload (95% Read, 5% Update)

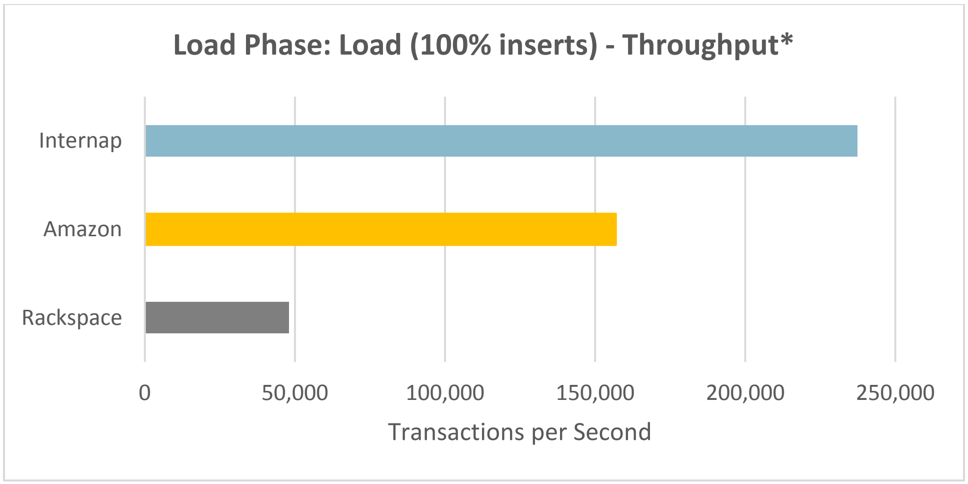
- Internap outperformed Amazon by 61% in throughput speed
- Internap outperformed Rackspace by 2.5x in throughput speed
- Amazon outperformed Rackspace by 52% in throughput speed

- Internap has 51% less latency than Rackspace
- Internap has 48% less latency than Amazon
- Amazon has 7% less latency than Rackspace

Result Details

Load Phase: Load (100% Inserts)

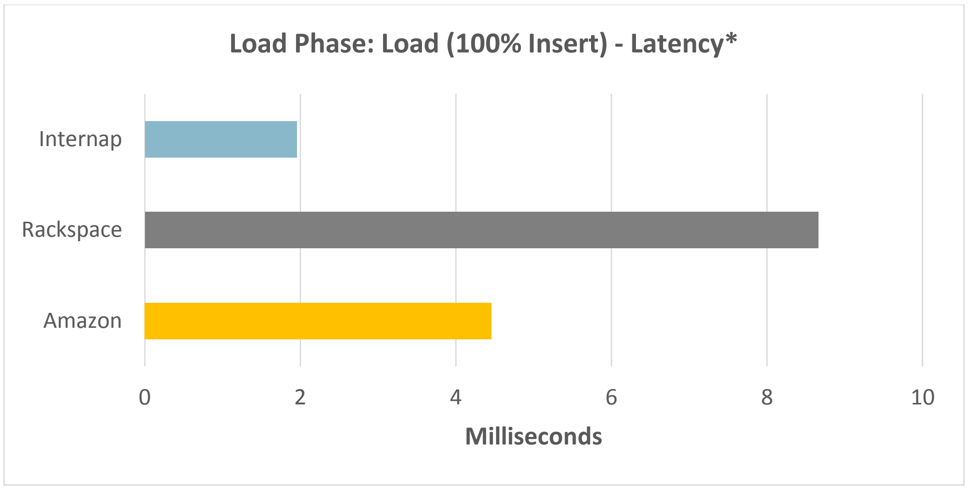
Load Phase is not really a workload, but rather the process of loading data into the database cluster for use in the next two workloads. This analysis tests the write speed to SSD by inserting individual records into the database as fast as it can. An example of this workload in production might be an institution loading customer or membership information from several departmental databases into an operational data store.



*Throughput scores are the aggregate transactions per second (TPS) from all four database hosts

Internap outperforms Amazon and Rackspace in throughput. Internap's bare metal has a throughput of 237,428 transactions per second (TPS). Amazon follows with 156,938 TPS; Rackspace trails with 47,982 TPS.

- Internap outperformed Amazon by 51%
- Internap outperformed Rackspace by 5x
- Amazon outperformed Rackspace by 3.3x



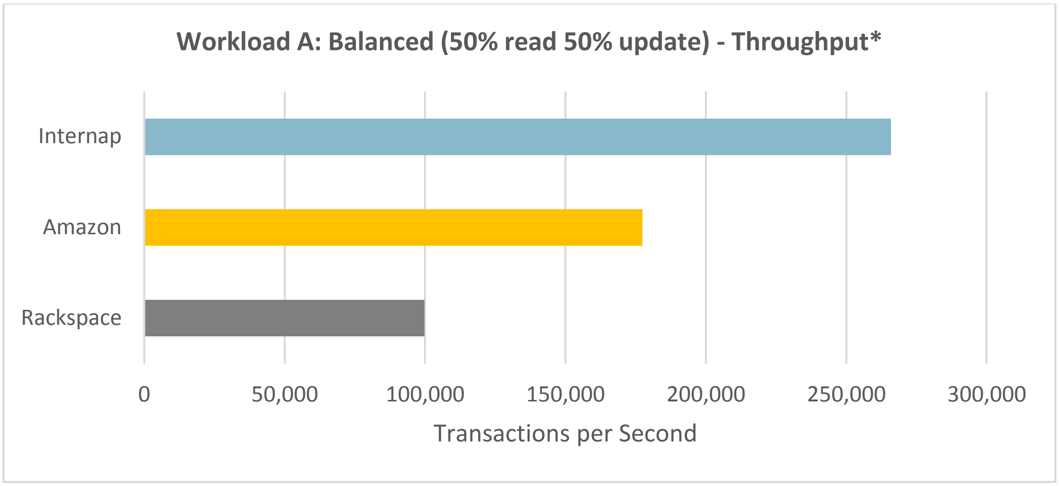
*Lower value is better

The Internap hosts had the least latency when compared with Rackspace and Amazon. Internap's bare metal has an average latency of 1.95ms. Amazon follows with an average latency of 4.46ms; Rackspace trails with an average latency of 8.66ms.

- Internap has 77% less latency than Rackspace
- Internap has 56% less latency than Amazon
- Rackspace has 94% more latency than Amazon

Workload A: Balanced (50% Read, 50% Update)

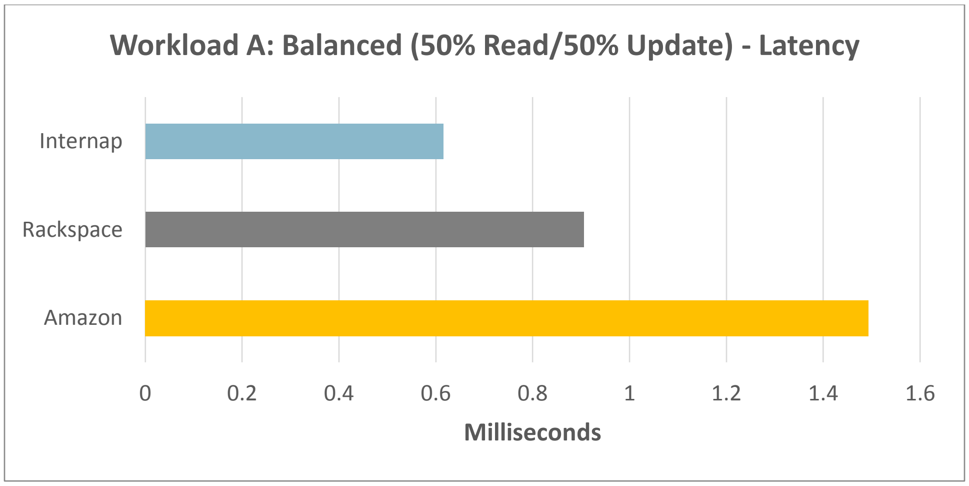
YCSB's Workload A is a balance of both reads and updates. Records were selected using a random Zipfian distribution. An application that tracks the activity of users at e-commerce sites and then personalizes digital advertisements based on their activity is a good example of a real-world workload that mirrors this testing scenario.



*Throughput scores are the aggregate transactions per second (TPS) from all four database hosts

Internap performs approximately 50% better than Amazon and 2.7x the level of Rackspace. Internap's bare metal has a throughput of 265,890 TPS. Amazon follows with 177,498 TPS; Rackspace trails with 99,847 TPS.

- Internap outperformed Amazon by 50%
- Internap outperformed Rackspace by 2.7x
- Amazon outperformed Rackspace by 78%



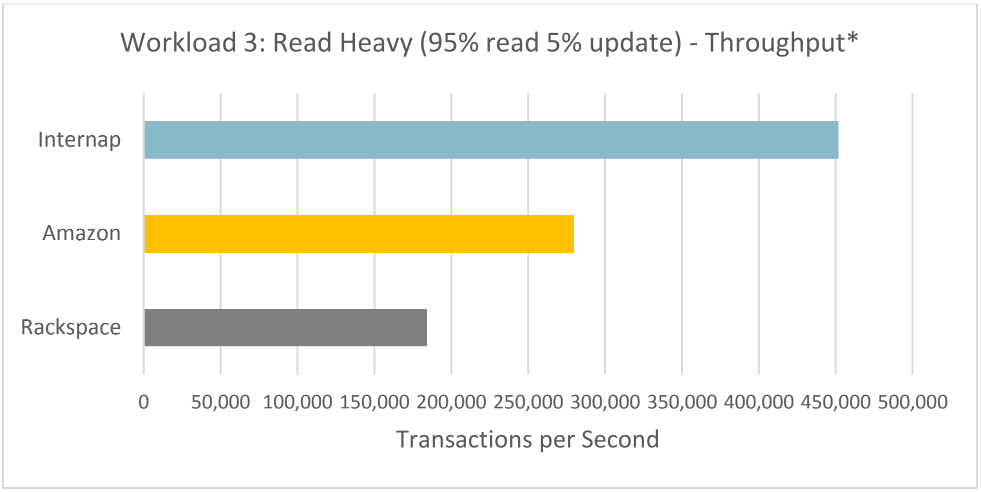
*Lower value is better

Internap has the least amount of latency relative to Rackspace and Amazon. Internap's bare metal has an average latency of 0.62ms. Rackspace follows with an average latency of 0.91ms; Amazon trails with an average latency of 1.49ms.

- Internap has 59% less latency than Amazon
- Internap has 32% less latency than Rackspace
- Rackspace has 39% less latency than Amazon

Workload B: Read Heavy (95% Read, 5% Update)

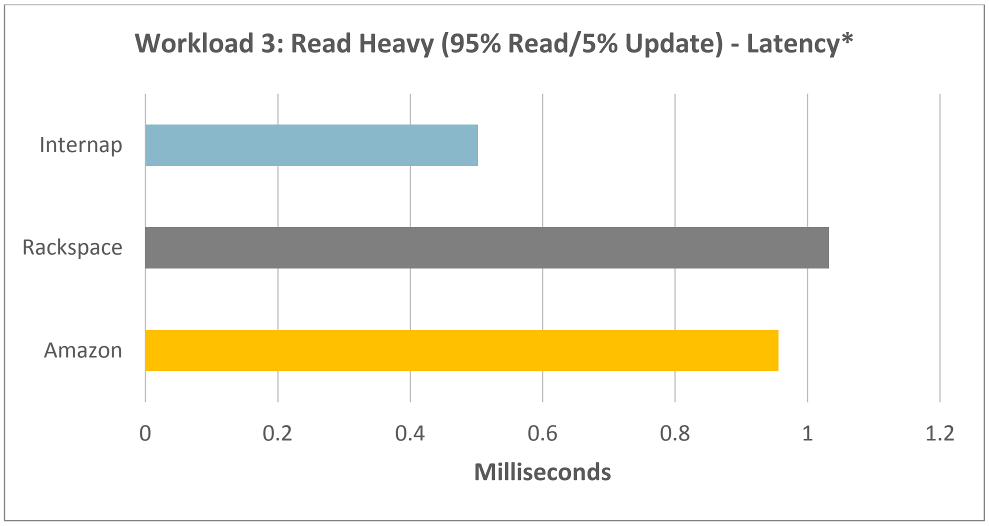
YCSB's Workload B is a read-intensive test with minimal updates. Records were selected using a random Zipfian distribution. An example of a read-intensive workload is a content distribution scheme that uses content tagging for search engines.



*Throughput scores are the aggregate transactions per second (TPS) from all four database hosts

Internap bare metal generated 61% more TPS compared with Amazon and 2.4x more than Rackspace. Internap's bare metal has a throughput of 451,561 TPS. Amazon follows with 279,661 TPS; Rackspace trails with 184,188 TPS.

- Internap outperformed Amazon by 61%
- Internap outperformed Rackspace by 2.5x
- Amazon outperformed Rackspace by 52%



*Lower value is better

Internap has the lowest latency compared to Rackspace and Amazon. Internap's bare metal has an average latency of 0.50ms. Amazon follows with an average latency of 0.96ms; Rackspace trails with an average latency of 1.03ms.

- Internap has 51% less latency than Amazon
- Internap has 48% less latency than Rackspace
- Amazon has 7% less latency than Rackspace

Price/Performance Values

The overall performance of Internap's bare-metal platform with SSDs has shown both higher throughput and lower latency than the two similarly-equipped virtual cloud providers. It is also prudent to evaluate the overall value that the different offerings present, given both price and performance metrics. When running a server environment with the capacity to scale far outwards, the price/performance value is an important number to track, as hosting costs can quickly grow as well.

Server Pricing

Provider	Server/Instance	Hourly Cost (Individual)	Monthly Cost (Individual)	Hourly Cost (DB Cluster)	Monthly Cost (DB Cluster)
Internap	E3-1230	$1.68*	$874	$6.72	$3,496
Amazon	I2.8xlarge (dedicated)	$6.82	$4,992	$27.28	$19,969
Rackspace	120GB Performance 2	$5.44	$3,971	$21.76	$15,885

*Illustrative

The CloudSpecs Score Calculation

The CloudSpecs Score is an indexed, comparable score ranging from 0-100 indicative of value based on a combination of cost and performance. The value is scaled; e.g., a Cloud Service Provider (CSP) with a score of 100 gives 4x the value of a CSP with a score of 25.

The calculation of the CloudSpecs score (Throughput):

1. provider_value = [Provider Performance Score] / [Provider Cost]
2. best_provider_value = max(provider_values)
3. CSP's CloudSpecs Score = 100 * provider_value / best_provider_value

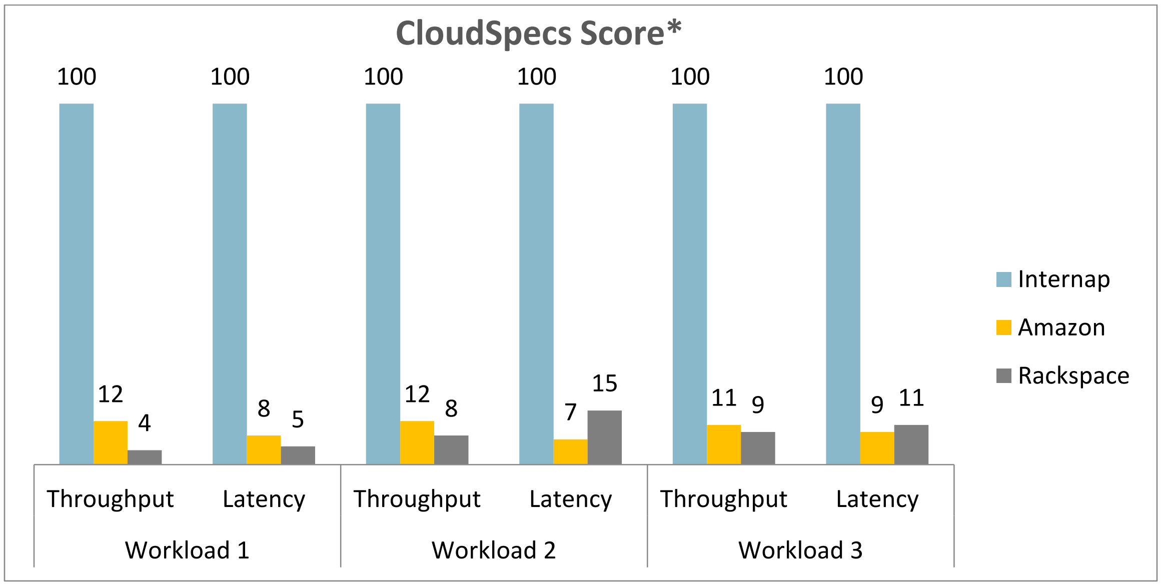
The calculation of the CloudSpecs score (Latency):

1. provider_value = [Provider Performance Score] * [Provider Cost]
2. best_provider_value = min(provider_values)
3. CSP's CloudSpecs Score = 100 * best_provider_value / provider_value

For example, the CloudSpecs Score calculation for the throughput of Load Phase is shown below:

Offering	Score Divided by Price	Divided by Highest Score	Multiply by 100	Throughput CloudSpecs Score
Internap	237428TPS / $3,496 = 67.91	67.91 / 67.91 = 1	1 x 100 = 100	100
Amazon	156938TPS / $19,969 = 7.86	7.86/ 67.91 = 0.12	0.12 x 100 = 12	12
Rackspace	47982TPS / $15,885 = 3.02	3.02 / 67.91 = 0.04	0.04 x 100 = 4	4

CloudSpecs Score



	Load Phase		Workload A		Workload B	
	Throughput	Latency	Throughput	Latency	Throughput	Latency
Internap	100	100	100	100	100	100
Amazon	12	8	12	7	11	9
Rackspace	4	5	8	15	9	11

*Based on Monthly Pricing

Conclusion

Study Results

Internap's bare-metal server delivers dedicated processing resources of a physical server, providing superior and more consistent performance than corresponding virtual cloud servers. Typical use cases for bare-metal servers with SSDs include data intensive applications for analytics platforms involving digital marketing, personalization and/or advertising. In these scenarios, NoSQL databases store and process user profiles, cookies, clickstreams and segmented behavioral data in real time.

The study confirmed the prediction that bare metal would outperform cloud servers on the YCSB test. Despite the virtualized cloud servers offering 2x the number of cores, 2-4x the amount of RAM, and also utilizing SSDs, the bare-metal servers with SSDs still outperformed the virtual cloud servers by large margins. Bare metal significantly outperforms on writes to disk, as seen on the load and balanced workloads. Performance differentials were not as pronounced with the read-intensive workload.

Compared with the large differential between Internap and Amazon for throughput speeds, Amazon instance throughput performed close to Rackspace's cloud servers. Latency is also relatively similar between the Amazon and Rackspace servers. Perhaps this is an indicator that at the larger instance sizes, there is a certain cap that performance reaches, whether due to throttling or physical constraints of the hardware.

Implications for Organizations Running NoSQL Workloads

For a high-performance, transaction-intensive NoSQL database such as Aerospike that parallelizes processing across cores and SSDs, dedicated hardware and strong internal network connections deliver a marked difference in performance relative to virtualized public cloud environments. NoSQL database clusters demand higher network throughput for interconnected nodes, and individual nodes benefit from the allocation of dedicated hardware over shared environments. Associated use cases like real-time, contextual personalization and promotion for example, might benefit from the direct access to physical hardware and unimpeded server-to-server networking that automated bare-metal platforms provide. Organizations that have maintained long-term hosted or owned environments to ensure adequate host performance might also look at bare-metal cloud as a way to maintain quality while improving asset utilization and agility.

Areas for Additional Research

The testing in this study was conducted on SSD, which makes it dependent on IOPS in storage. Aerospike offers an in-memory option as well, which should provide higher performance with the lack of dependency on disk performance. Future experiments can incorporate in-memory performance to study the effects of disk/memory on the overall performance of Aerospike across providers.

This experiment was conducted according to guidelines from Aerospike's documentation, which also recommends the use of Amazon Linux. Upon further testing, though, Amazon EC2's HVM images seem to provide a higher performance output than the Amazon Linux images can sustain. Further experimentation with HVM images on the EC2 platform may offer a further improved performance on top of Amazon EC2's dedicated environment.

Future testing should also assess automated bare metal cloud services across providers. Such a study would provide an indication on the degree of performance differences when hosting on vendors' bare metal cloud management and hardware platforms. With more cloud providers offering bare metal servers, being able to offer a product that can differentiate on performance would be key.

Another interesting area for future study would be testing NoSQL on a larger number of cloud providers. Cloud servers are an appealing hosting option for NoSQL databases as they can scale horizontally with ease to accommodate changing data sets. With SSD storage and high RAM offerings becoming commonplace among cloud vendors; the providers are lowering barriers to hosting NoSQL environments in the cloud.

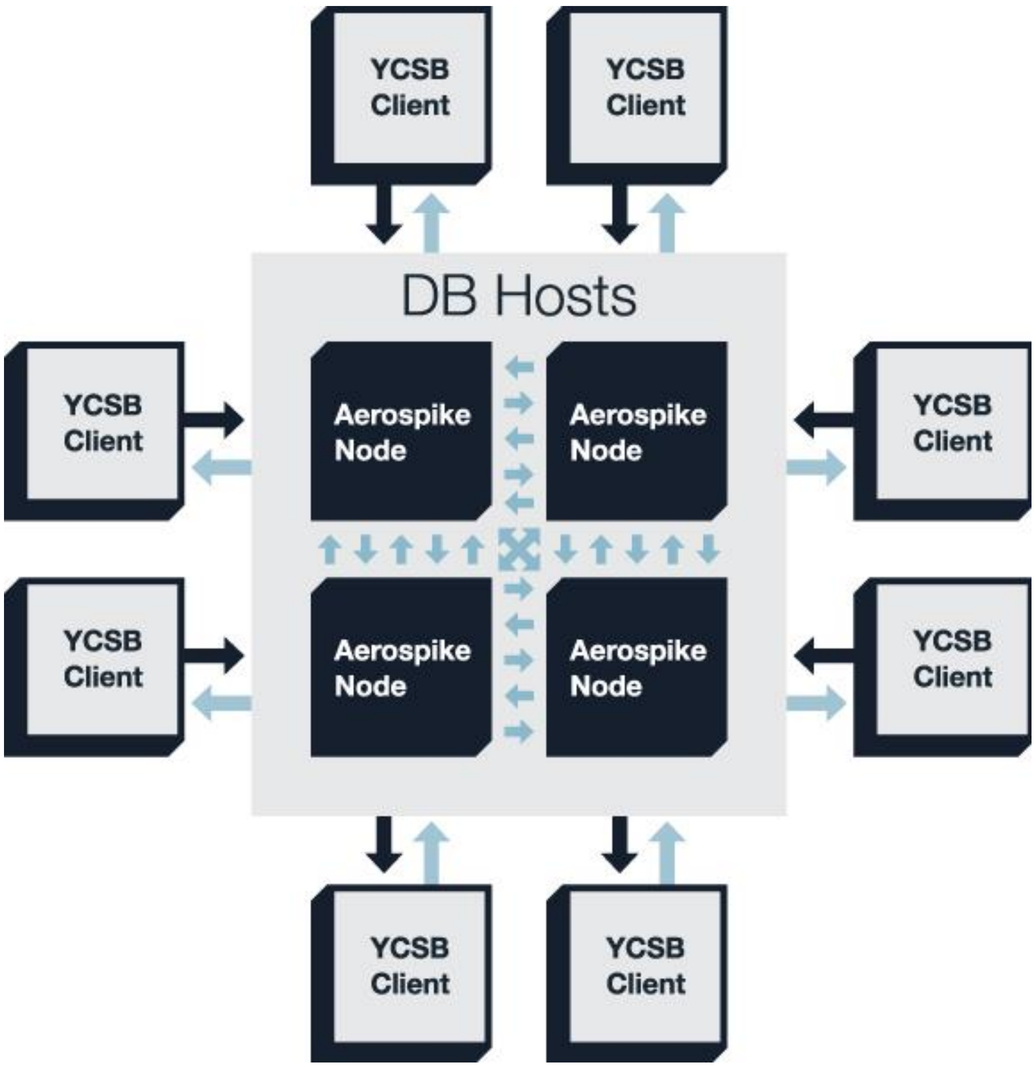
About Cloud Spectator

Cloud Spectator is the premier, international cloud analyst group focused on infrastructure pricing and server performance. Since 2011, Cloud Spectator has monitored the cloud Infrastructure industry on a global scale and continues to produce research reports for businesses to make informed purchase decisions by leveraging its CloudSpecs utility, an application that automates live server performance tests 3 times a day, 365 days a year with use of open source benchmark tests. Currently, the CloudSpecs system actively tracks 20 of the top IaaS providers around the world.

Cloud Spectator
800 Boylston Street, 16th Floor
Boston, MA 02199
Website: www.cloudspectator.com
Phone: (USA) 1-617-300-0711

Appendix

Appendix A: Database and Client Configuration



Aerospike Node - Single instance of an Aerospike database process.

DB Host - Actual host (bare metal or virtualized) where the Aerospike node resides – four DB hosts were utilized, one for each Aerospike node.

YCSB Client – Hosts that are making queries to the Aerospike database – these simulate calls and puts to the database usually by an application server or web server. Eight YSCB client servers were used in this analysis.

The intra-cluster communication between the Aerospike nodes is used as a heartbeat mechanism to determine the size of the cluster and maintain its integrity. The communication between the Aerospike nodes and client hosts indicate standard queries for reads/writes/updates to the database cluster.

The selection of database nodes were originally based on matching by cores, but the results of testing required Amazon EC2 and Rackspace servers to be scaled up to get within a closer range of performance to the Internap Bare Metal machine. Thus, Amazon EC2 and Rackspace servers were scaled to 32 vCPUs.

Client machines were selected based on load. Internap provided 12-core bare metal machines to us in testing. Local VMs were used on Amazon EC2 and Rackspace Cloud. No client machines exceeded 50% utilization throughout the experiment.

Database Setup
(4 Nodes)

Internap
OS: Ubuntu 12.04
JDK: n/a
Database: Aerospike 3.2.9

CPU: 4 CPU
RAM: 32GB
HDD: 2x240GB SSD & 1x480GB SSD

Amazon
OS: Amazon Linux
JDK: n/a
Database: Aerospike 3.2.9

CPU: 32vCPU
RAM: 244GB
HDD: 8x800GB SSD

Rackspace
OS: Ubuntu 12.04
JDK: n/a
Database: Aerospike 3.2.9

CPU: 32vCPU
RAM: 120GB
HDD: 4x300GB SSD

Client Setup
(8 Client Machines)

Internap
OS: Ubuntu 12.04
JDK: Openjdk-6
Benchmark: YCSB modified (*See Appendix C for changes*)

CPU: 12 CPU
RAM: 64GB
HDD: 1x480GB SSD

Amazon
OS: Amazon Linux
JDK: Openjdk-6
Benchmark: YCSB modified (*See Appendix C for changes*)

CPU: 16vCPU
RAM: 30GB
HDD: 2x160GB SSD

Rackspace
OS: Ubuntu 12.04
JDK: Openjdk-6
Benchmark: YCSB modified (*See Appendix C for changes*)

CPU: 8vCPU
RAM: 30GB
HDD: 300GB SSD

Appendix B: Testing Configuration

Testing Process

1. Provision the machines, install the operating system (OS) and configure the environment
    a. Ubuntu 12.04 used for Internap and Rackspace; Ubuntu 14.04 on Amazon
    b. Para-virtualization Hardware Virtualized Machines (PV-HVM) used on Amazon and Rackspace
        i. Amazon Linux HVM AMI used on Amazon

2. Setup the database cluster machines, install and configure the database
    a. Wipe data drives to create a fresh cluster
        i. Overprovision Internap SSDs using hdparm
    b. Install the Aerospike server software
    c. Configure the cluster using Aerospike config and adjust parameters
    d. Adjust TCP parameters for faster reuse of TCP ports

3. Setup the client machines, install YCSB
    a. Setup the network configuration

4. Appropriate the data set size
    a. Select data set properties and usage

5. Run the benchmark
    a. Select the workloads to run
    b. Run warm-up session to achieve steady state for accurate results
        i. Thumbtack modified YCSB has warm-up option

6. Aggregate the results
    a. Retrieve merged report from “master” client

7. Tear down servers, rebuild machines and environment for next run

*Notes on Testing – *See Appendix B*

Database Configuration

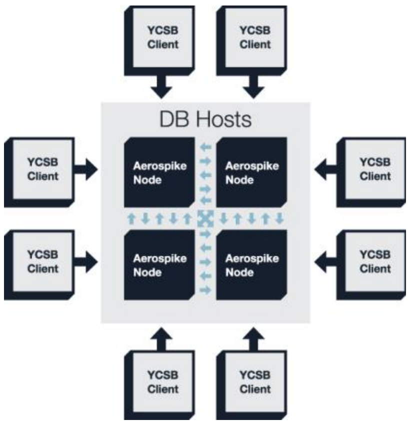
- 21% of disk left unpartitioned to overprovision disks (recommended by Aerospike)
- 4 nodes will be configured to function as a single cluster with distributed tasks
- Database set to 2x replication
- Dataset size will begin at 45% of the total remaining space on disk

Client and Workload Configuration

- Data Properties
  Record Description: 10-string fields at 10 bytes per field
    - 2 bytes additional for name

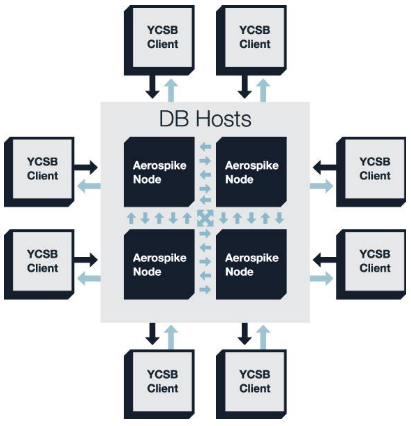
  Record Size: 120 bytes
  Key Size: 23 bytes
  * Data set should at least double the amount of RAM to keep performance in the storage

Number of Records: 900 million on 60GB RAM servers & 1.5 billion on 120GB RAM servers
Number of Threads to Load the Data Set: 32 Threads per Client Machine



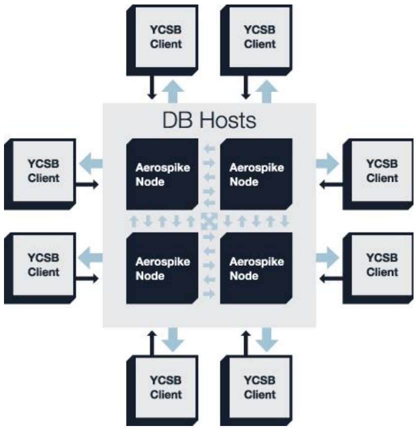
Load Phase: Load
Load Phase loads the data set for the follow-up workloads involving read and updates. The YCSB installed on the clients generate the data sets which are subsequently loaded onto the cluster.

- 100% Inserts



Workload A: Balanced

- 50% Read
- 50% Update



Workload B: Read-Heavy

- 95% Read
- 5% Update

The workloads were selected with a zipfian distribution to simulate Internet-like NoSQL usage. Before each workload is tested, a warm-up session equal to the amount of time for testing is run to prime any cache and establish steady state results as expected in a live production environment.

Appendix C: YCSB Customization

Cloud Spectator forked from latest commit 073c59348d from github (https://github.com/thumbtack-technology/ycsb)

For official changes to the YCSB client that are supported by Aerospike, visit their github repository (https://github.com/aerospike/ycsb)

Cloud Spectator contributions are committed to github (https://github.com/cloudspectator/YCSB_Aerospike3)

```
package com.yahoo.ycsb.db;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;
import java.util.Set;
import java.util.Vector;

import com.aerospike.client.AerospikeException;
import com.aerospike.client.Bin;
import com.aerospike.client.Key;
import com.aerospike.client.Record;
import com.aerospike.client.ResultCode;
import com.aerospike.client.policy.Policy;
import com.aerospike.client.policy.WritePolicy;
import com.yahoo.ycsb.ByteArrayByteIterator;
import com.yahoo.ycsb.ByteIterator;
import com.yahoo.ycsb.DBException;

public class AerospikeClient extends com.yahoo.ycsb.DB{

        public static final int OK = 0;
        public static final int NULL_RESULT = -20;
        /**
         *  Mapping of ResultCodes to documented Client Return Error codes
         *  @see http://www.aerospike.com/quick-install/documentation/troubleshooting-guide/#clientcodes
         */
        private static final Map<Integer, Integer> RESULT_CODE_MAPPER;
        static {
                RESULT_CODE_MAPPER = new HashMap<Integer, Integer>();

                RESULT_CODE_MAPPER.put(ResultCode.SERVER_ERROR, 1);
                RESULT_CODE_MAPPER.put(ResultCode.KEY_NOT_FOUND_ERROR, 2);
                RESULT_CODE_MAPPER.put(ResultCode.GENERATION_ERROR, 3);
                RESULT_CODE_MAPPER.put(ResultCode.PARAMETER_ERROR, 4);
                RESULT_CODE_MAPPER.put(ResultCode.KEY_EXISTS_ERROR, 5);
                RESULT_CODE_MAPPER.put(ResultCode.BIN_EXISTS_ERROR, 6);
                RESULT_CODE_MAPPER.put(ResultCode.CLUSTER_KEY_MISMATCH, 7);
                RESULT_CODE_MAPPER.put(ResultCode.SERVER_MEM_ERROR, 8);
                RESULT_CODE_MAPPER.put(ResultCode.TIMEOUT, 9);
```

```
		RESULT_CODE_MAPPER.put(ResultCode.NO_XDS, 10);
		RESULT_CODE_MAPPER.put(ResultCode.SERVER_NOT_AVAILABLE, 11);
		RESULT_CODE_MAPPER.put(ResultCode.BIN_TYPE_ERROR, 12);
		RESULT_CODE_MAPPER.put(ResultCode.RECORD_TOO_BIG, 13);
		RESULT_CODE_MAPPER.put(ResultCode.KEY_BUSY, 14);

		RESULT_CODE_MAPPER.put(ResultCode.OK, OK);
		//RESULT_CODE_MAPPER.put(ResultCode.CLIENT_ERROR, -4);
		RESULT_CODE_MAPPER.put(ResultCode.SERIALIZE_ERROR, -10);

		//RESULT_CODE_MAPPER.put(ResultCode.NOT_SET, -1);
	}

	private com.aerospike.client.AerospikeClient cl;

	public static  String NAMESPACE = "test";

	public static  String SET = "YCSB";

	public void init() throws DBException {
		Properties props = getProperties();
		int port;

		//retrieve port
		String portString = props.getProperty("port");
		if (portString != null) {
			port = Integer.parseInt(portString);
		}
		else {
			port = 3000;
		}

		//retrieve host
		String host = props.getProperty("host");
		if(host == null) {
			host = "aecluster";
		}

		//retrieve namespace
		String ns = props.getProperty("ns");
		if(ns !=  null ) {
			NAMESPACE = ns;
		}

		//retrieve set
		String st = props.getProperty("set");
		if(st != null) {
			SET = st;
		}

		try {
			cl = new com.aerospike.client.AerospikeClient(host, port);
		} catch (AerospikeException e) {
			e.printStackTrace();
			throw new DBException("Can't create an AerospikeClient", e);
		}
		try {
			//Sleep so that the partition hashmap is created by the client
			Thread.sleep(2000);
		} catch (InterruptedException ex) {
		}

		if (!cl.isConnected()) {
			throw new DBException(String.format("Failed to add %s:%d to cluster.",
					host, port));
		}

	}

	public void cleanup() throws DBException {
		cl.close();
	}

	@Override
	public int read(String table, String key, Set<String> fields,
			HashMap<String, ByteIterator> result) {
		try {
			Policy _policy = new Policy();
			//_policy.timeout = 50;  // 50 millisecond timeout.
			Key _key = new Key(NAMESPACE, SET, key);

			if (fields != null) {
				for (String bin : fields) {
					Record res = cl.get(_policy, _key, bin);
					if (res != null) {
						result.put(bin, new
ByteArrayByteIterator(res.getValue(bin).toString().getBytes()));
					} else {
						return NULL_RESULT;
					}
				}
			} else {
				Record record = cl.get(_policy, _key);
				if (record != null) {
					for (Map.Entry<String,Object> entry : record.bins.entrySet()) {
						result.put(entry.getKey(), new
ByteArrayByteIterator(entry.getValue().toString().getBytes()));
					}
				}
			}
		} catch(AerospikeException aex) {
			aex.printStackTrace();
			return RESULT_CODE_MAPPER.get(aex.getResultCode());
		}
		return OK;
	}

	@Override
	public int scan(String table, String startkey, int recordcount,
			Set<String> fields, Vector<HashMap<String, ByteIterator>> result) {
		// TODO Auto-generated method stub
		return -1;
	}

	@Override
	public int update(String table, String key,
			HashMap<String, ByteIterator> values) {
		return insert(table, key, values);
	}

	@Override
	public int insert(String table, String key,
			HashMap<String, ByteIterator> values) {
		try {
			WritePolicy _policy = new WritePolicy();
			//policy.timeout = 50;  // 50 millisecond timeout.
			Key _key = new Key(NAMESPACE, SET, key);
			ArrayList<Bin> bins = new ArrayList<Bin>();

			for (Map.Entry<String, ByteIterator> entry : values.entrySet()){
				bins.add(new Bin(entry.getKey(), entry.getValue().toString()));
			}

			cl.put(_policy, _key, bins.toArray(new Bin[bins.size()]));

			return OK;

		} catch(AerospikeException aex) {
			aex.printStackTrace();
			return RESULT_CODE_MAPPER.get(aex.getResultCode());
		}
	}

	@Override
	public int delete(String table, String key) {
		try {
			WritePolicy _policy = new WritePolicy();
			//policy.timeout = 50;  // 50 millisecond timeout.
			Key _key = new Key(NAMESPACE, SET, key);
			cl.delete(_policy, _key);
			return OK;

		} catch(AerospikeException aex) {
			aex.printStackTrace();
			return RESULT_CODE_MAPPER.get(aex.getResultCode());
		}
	}

}
```

Appendix D: Provider Value Results

Throughput Calculation: Provider Throughput Score / Provider Price

Latency Calculation: Provider Latency Score x Provider Price

	Load Phase		**Workload A**		**Workload B**	
	Throughput	Latency	Throughput	Latency	Throughput	Latency
Internap	67.91	6832.88	76.06	2151.34	129.16	1753.14
Amazon	7.96	89010.86	8.89	29833.25	14	19079.37
Rackspace	3.02	137509.51	6.29	14380.53	11.6	16396.13