Lecture 07

NoSQL , Cassandra & Co.

Zoran B. Djordjević

csci e63 Big Data Analytics

History of the World, Part 1

- Relational Databases – mainstay of business
  - For the longest time (and still true today), the big relational database vendors such as Oracle, IBM, Sybase, and Microsoft were the mainstay of how data was stored.
  - During the Internet boom, startups looking for low-cost RDBMS alternatives turned to MySQL and PostgreSQL.
- Web-based applications caused spikes
  - Could have hundreds of thousands of visitors in a short-time span. Especially true for public-facing e-Commerce sites
- Developers begin to front RDBMS with memcache or integrate other caching mechanisms within the application (ie. Ehcache)
- As datasets grew, the simple memcache/MySQL model (for lower-cost startups) started to become problematic.

Scaling Up

- Best way to provide ACID and a rich query model is to have the dataset on a single machine.
- Issues with scaling up when the dataset is just too big
- RDBMS were not designed to be distributed
- Began to look at multi-node database solutions
- Known as ‘scaling out’ or ‘horizontal scaling’
- Different approaches include:
  - Master-slave
  - Sharding

Scaling RDBMS – Master/Slave

- Master-Slave
  - All writes are written to the master. All reads performed against the replicated slave databases
  - Critical reads may be incorrect as writes may not have been propagated down
  - Large data sets can pose problems as master needs to duplicate data to slaves

Scaling RDBMS - Sharding

- Partitioning or sharding
- Different sharding approaches:
- Vertical Partitioning: Have tables related to a specific feature sit on their own server. May have to rebalance or reshard if tables outgrow server.
- Range-Based Partitioning: When single table cannot sit on a server, split table onto multiple servers. Split table based on some critical value range.
- Key or Hash-Based partitioning: Use a key value in a hash and use the resulting value as entry into multiple servers.
- Directory-Based Partitioning: Have a lookup service that has knowledge of the partitioning scheme . This allows for the adding of servers or changing the partition scheme without changing the application.
  - Scales well for both reads and writes
  - Not transparent, application needs to be partition-aware
  - Can no longer have relationships/joins across partitions
  - Loss of referential integrity across shards

Other ways to scale RDBMS

- Multi-Master replication.
  - The multi-master replication system is responsible for propagating data modifications made by each member to the rest of the group, and resolving any conflicts that might arise between concurrent changes made by different members.
- INSERT only, not UPDATES/DELETES.
  - For INSERT-only, data is versioned upon update.
  - Data is never DELETED, only inactivated.
- No JOINs, thereby reducing query time
  - This involves de-normalizing data
  - Consistency is the responsibility of the application.
- In-memory databases
- In-memory databases have not caught on mainstream and regular RDBMS are more disk-intensive that memory-intensive

What is NoSQL?

- Stands for **N**ot **O**nly **SQL**
- Class of non-relational data storage systems
- Usually do not require a fixed table schema nor do they use the concept of joins
- All NoSQL offerings relax one or more of the ACID properties.
- For data storage, an RDBMS cannot be the only solution.
- Just as there are different programming languages, need to have other data storage tools in the toolbox
- Relational databases offer a very good general purpose solution to many different data storage needs.
- In other words, it is the safe choice and will work in many situations.

How did we get here?

- Explosion of social media sites (Facebook, Twitter) with large data needs. These datasets have high read/write rates.
- Rise of cloud-based solutions such as Amazon S3 (simple storage solution) made NoSQL universally accessible
- Like a move to dynamically-typed languages (Ruby/Groovy), a shift to dynamically-typed data with frequent schema changes
- All of the NoSQL options with the exception of Amazon S3 (Amazon Dynamo) are open-source solutions.

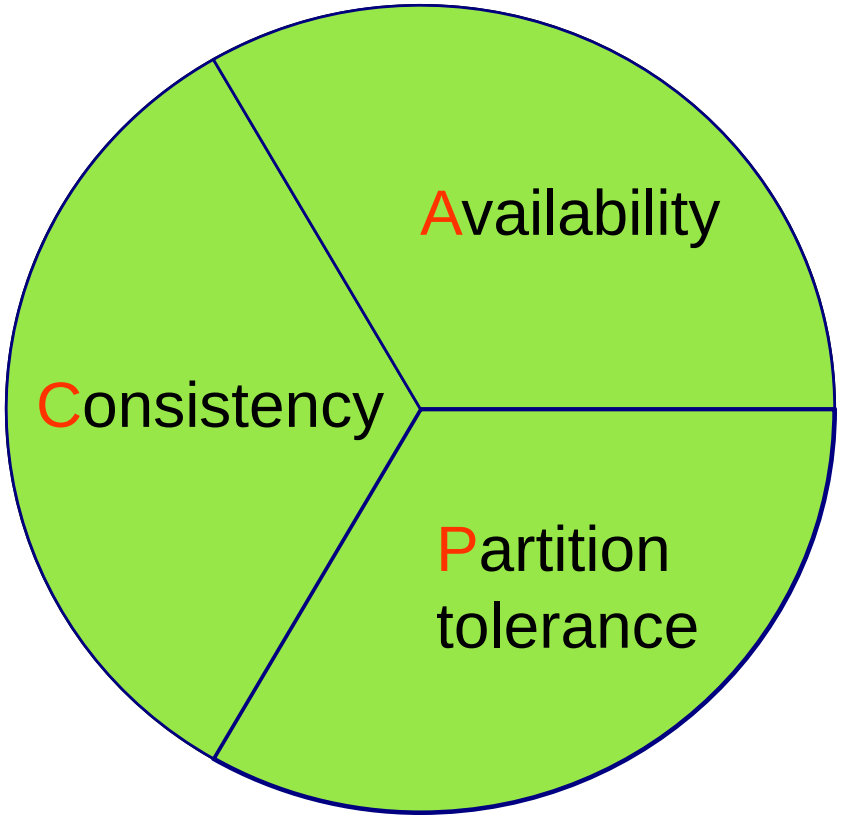
DynamoDB and BigTable

- Three major papers were the seeds of the NoSQL movement
  - BigTable (Google)
    - BigTable: http://labs.google.com/papers/bigtable.html
  - DynamoDB (Amazon)
    - http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html and
    - http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf
    - Gossip protocol (discovery and error detection)
    - Distributed key-value data store
    - Eventual consistency
      - Amazon and consistency
      - * http://www.allthingsdistributed.com/2010/02
      - * http://www.allthingsdistributed.com/2008/12
  - CAP Theorem (discuss in a sec ..)

The Perfect Storm

- Large datasets, acceptance of alternatives, and dynamically-typed data has come together in a perfect storm.
- Industry have reached a point where a read-only cache and write-based RDBMS isn't delivering the throughput necessary to support many internet scale application.
- Not a backlash/rebellion against RDBMS
- The NoSQL databases are a pragmatic response to growing scale of databases and the falling prices of commodity hardware.
- Most likely, 10 years from now, the majority of data is still stored in RDBMS.
- SQL is a rich query language that cannot be rivaled by the current list of NoSQL offerings

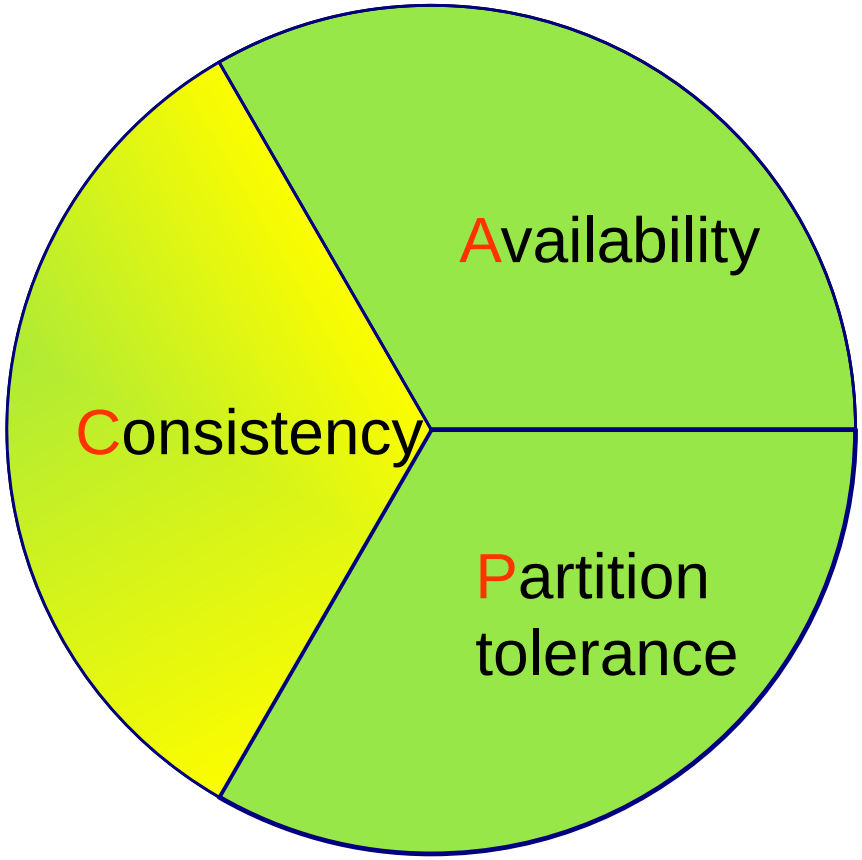
The CAP Theorem



CAP Theorem

- Proposed by Eric Brewer (talk on Principles of Distributed Computing July 2000).
- Three properties of a system: **C**onsistency, **A**vailability and **P**artitionability.
- **You can have at most two of these three properties for any shared-data system**
  - Partitionability: Can divide nodes into small groups that can see other groups, but they can't see everyone.
  - Consistency: write a value and then you read the value you get the same value back. In a partitioned system there are windows where that's not true.
  - Availability: may not always be able to write or read. The system will say you can't write because it wants to keep the system consistent.
- To scale you have to partition, so you are left with choosing either high consistency or high availability for a system. Find the right overlap of availability and consistency. Choose an approach based on the service
- For the checkout process you always want to honor requests to add items to a shopping cart because it's revenue producing. In this case you choose high availability. Errors are hidden from the customer and sorted out later.
- When a customer submits an order you favor consistency because several services--credit card processing, shipping and handling, reporting— are simultaneously accessing the data.

The CAP Theorem



Once a writer has written, all readers will see that write.

- Two kinds of consistency:
  - strong consistency – ACID(Atomicity Consistency Isolation Durability)
  - weak consistency – BASE(Basically Available Soft-state Eventual consistency)

What is NoSQL?

- NoSQL is not a relational database. The reality is that a relational database model may not be the best solution for all situations.
- The easiest way to think of NoSQL, is that of a database which does not adhere to the traditional relational database management system (RDMS) structure.
- Sometimes you will also see it referred to as 'not only SQL'.

What kinds of NoSQL

- NoSQL solutions fall into two major areas:
  - Key/Value or 'the big hash table'.
    - Amazon S3 (Dynamo)
    - Voldemort
    - Scalaris
    - Memcached (in-memory key/value store)
    - Redis
  - Schema-less which comes in multiple flavors, column-based, document-based or graph-based.
    - Cassandra (column-based)
    - CouchDB (document-based, document: views are stored as rows which are kept sorted by key.)
    - MongoDB(document-based)
    - Neo4J (graph-based, is a network database that uses edges and nodes to represent and store data)
    - HBase (column-based)

Key/Value

Pros:

- very fast
- very scalable
- simple model
- able to distribute horizontally

Cons:

- many data structures (objects) can't be easily modeled as key value pairs

Schema-Less

Pros:

- Schema-less data model is richer than key/value pairs
- eventual consistency
- many are distributed
- still provide excellent performance and scalability

Cons:

- typically no ACID transactions or joins

Common Advantages

- Cheap, easy to implement (open source)
- Data are replicated to multiple nodes (therefore identical and fault-tolerant) and can be partitioned
  - **Down nodes easily replaced**
  - **No single point of failure**
  - **As the data is written, the latest version is on at least one node. The data is then versioned/replicated to other nodes within the system.**
- Eventually, the same version is on all nodes.
- Easy to distribute
- Don't require a schema
- Can scale up and down
- Relax the data consistency requirement (CAP)

What am I giving up?

- joins
- group by
- order by
- ACID transactions
- SQL as a sometimes frustrating but still powerful query language
- easy integration with other applications that support SQL

Row Oriented Databases

- A relational database management system maintains data that represents two-dimensional tables, with columns and rows. For example, a database might have table Employee:

EmpId	Lastname	Firstname	Salary
10	Smith	Joe	40000
12	Jones	Mary	50000
11	Johnson	Cathy	44000
22	Jones	Bob	55000

- This two-dimensional format exists only on paper. Storage hardware requires the data to be serialized into a sequence of “cells” and placed onto the hard drives.
- The most expensive operations involving hard drives are seeks. In order to improve overall performance, related data should be stored in a fashion to minimize the number of seeks. Hard drives are organized into a series of blocks of a fixed size, typically enough to store several rows of the table. This minimizes the number of data retrievals.

Row Organized Data

- The common solution to the storage problem is to serialize each row of data, and assign to it a row id. Rows of the previous table could be packaged like this:

```
001:10,Smith,Joe,40000;002:12,Jones,Mary,50000;003:11,Johnson,Cathy,44000;004:22,Jones,Bob,55000;
```

- Indicators `001, 002, 003` and `004` represent row ids. In practice, those identifiers are usually longer, 64-bit or 128-bit strings.
- In OLTP systems, we need the entire raw(s) of data in order to populate entire object(s). It makes every sense to store all components of a row of data together. By storing the record's data in a single block on the disk, along with related records, the system can quickly retrieve records with a minimum of disk operations.
- Row-based systems are not efficient at performing operations that apply to the entire data set, as opposed to a specific record. For instance, in order to find all the records in the example table that have salaries between 40,000 and 50,000, the DBMS would have to seek through the entire data set looking for matching records.

Indexes Help

- To improve the performance of these sorts of operations, most DBMS's support the use of database indexes, which store all the values from a set of columns along with pointers back into the original rowid.
- An index on the salary column would look something like this:

```
001:40000;002:50000;003:44000;004:55000;
```

- As they store only single pieces of data, rather than entire rows, indexes are generally much smaller than the main table stores. By scanning smaller sets of data the number of disk operations is reduced. If the index is heavily used, it can provide dramatic time savings for common operations. However, maintaining indexes adds overhead to the system, especially when new data is written to the database. In this case not only is the record stored in the main table, but any attached indexes have to be updated as well.
- Database indexes on one or more columns are typically sorted by value, which makes operations like range queries fast.

Modern RDBMS

- There is a number of row-oriented databases that are designed to fit entirely in RAM, the so called an in-memory database (Oracle Times Ten).
- RAM is rapidly getting cheaper and big vendors like Oracle, (Microsoft,) IBM are offering specialized hardware with enormous RAM-s (several TB-s)
- These systems do not depend on disk operations, and have equal-time access to the entire dataset. This reduces the need for indexes, as it is required the same amount of operations to full scan the original data as a complete index for typical aggregation purposes. Such systems are simpler and smaller, but can only manage databases that fit into the memory.
- Classical Hard Drives are being replaced by Solid State Drives (SSD-s, Flash Drives) and several vendors (Aerospike, Amazon DynamoDB) are rewriting RDBMS systems to take advantage of new technology. Consistent reads and writes complete in under 1 millisecond on such systems. This is two orders of magnitude faster than on the classical HD systems.
- Traditional databases also try to scale with volume of data by using cluster technology. For whatever reason there are limitations to such scaling.
- **RDBMS are far from dead. They will remain the backbone of all IT systems.**

Column-oriented Systems

- Important column-oriented databases (Vertica, 2005; Statistics Canada RAPID System, 1969) are present for a while. Ra
- A column-oriented database serializes all of the values of a column together, then the values of the next column, and so on. For our example table, the data would be stored in this fashion:

```
10:001,12:002,11:003,22:004;
Smith:001,Jones:002,Johnson:003,Jones:004;
Joe:001,Mary:002,Cathy:003,Bob:004;
40000:001,50000:002,44000:003,55000:004;
```

- Any one of the columns more closely matches the structure of an index in a row-based system. This creates an impression that column-oriented store "is really just" a row-store with an index on every column.
- It is the mapping of the data that differs dramatically. In a row-oriented indexed system, the primary key is the rowid that is mapped to indexed data.

Data organization in Column-oriented systems

- In the column-oriented system primary key is the data, mapping back to rowids. The difference can be seen in the case when we have two rows (users) with the same last name. Column “last_name” would be stored as:

```
…;Smith:001,Jones:002,004,Johnson:003;…
```

- Record `"Jones"` could be saved only once in the column store along with pointers to all of the rows that match it. For many common searches, like "find all the people with the last name Jones", the answer is retrieved in a single operation. Similarly, counting the number of matching records, can be greatly improved through this organization.
- In a column-oriented system operations that retrieve complete data for objects would be slower, requiring numerous disk operations to collect data from multiple columns to build up the record.
- In the many cases, only a limited subset of data is retrieved. If we are collecting the first and last names from many rows in order to build a list of contacts, columnar organization is vastly beneficial.
- This is even more true for writing data into the database, especially if the data tends to be "sparse" with many optional columns.

Benefits

- Column-oriented organizations are more efficient when an aggregate needs to be computed over many rows but only for a notably small subset of all columns of data
- Column-oriented organizations are more efficient when new values of a column are supplied for all rows at once, because that column data can be written efficiently and replace old column data without touching any other columns for the rows.
- Row-oriented organizations are more efficient when many columns of a single row are required at the same time, and when row-size is relatively small, as the entire row can be retrieved with a single disk seek.
- Row-oriented organizations are more efficient when writing a new row if all of the row data is supplied at the same time, as the entire row can be written with a single disk seek.
- In practice, row-oriented storage layouts are well-suited for OLTP-like workloads which are more heavily loaded with interactive transactions. Column-oriented storage layouts are well-suited for OLAP-like workloads.

Compression

- Column data is of uniform type. Many popular modern compression schemes, such as LZW or run-length encoding, make use of the similarity of adjacent data to compress. While the same techniques may be used on row-oriented data, a typical implementation will be less effective.
- To improve compression, sorting rows can also help. For example, using bitmap indexes, sorting can improve compression by an order of magnitude. To maximize the compression benefits of the lexicographical order with respect to run-length encoding, it is best to use low-cardinality columns as the first sort keys. For example, given a table with columns sex, age, name, it would be best to sort first on the value sex (cardinality of two), then age (cardinality of <150), then name.
- Columnar compression achieves a reduction in disk space at the expense of efficiency of retrieval. Retrieving all data from a single row is more efficient when that data is located in a single location, such as in a row-oriented architecture. Further, the greater adjacent compression achieved, the more difficult random-access may become, as data might need to be uncompressed to be read.

New Breed of Databases

- With large popularity of Big Data analysis, several databases became quite fashionable. Among the most popular are: Cassandra, HBase, MongoDB, CouchDB, etc.
- Those databases are sometimes referred to as key-value pair database, sometimes as columnar-databases and most often as NoSQL database.
- NoSQL databases claim to deliver faster performance than legacy RDBMS systems in various use cases, most notably those involving big data. While this is oftentimes the case, it should be understood that not all NoSQL databases are created alike where performance is concerned.
- System architects and IT managers are wise to compare NoSQL databases in their own environments using data and user interactions that are representative of their expected production workloads before deciding which NoSQL database to use for a new application.
- DataStax performed a benchmark of three top NoSQL databases – Apache Cassandra, Apache HBase, and MongoDB – using a variety of different workloads on AWS clusters.

Benchmark Configuration, DataStax Evaluations

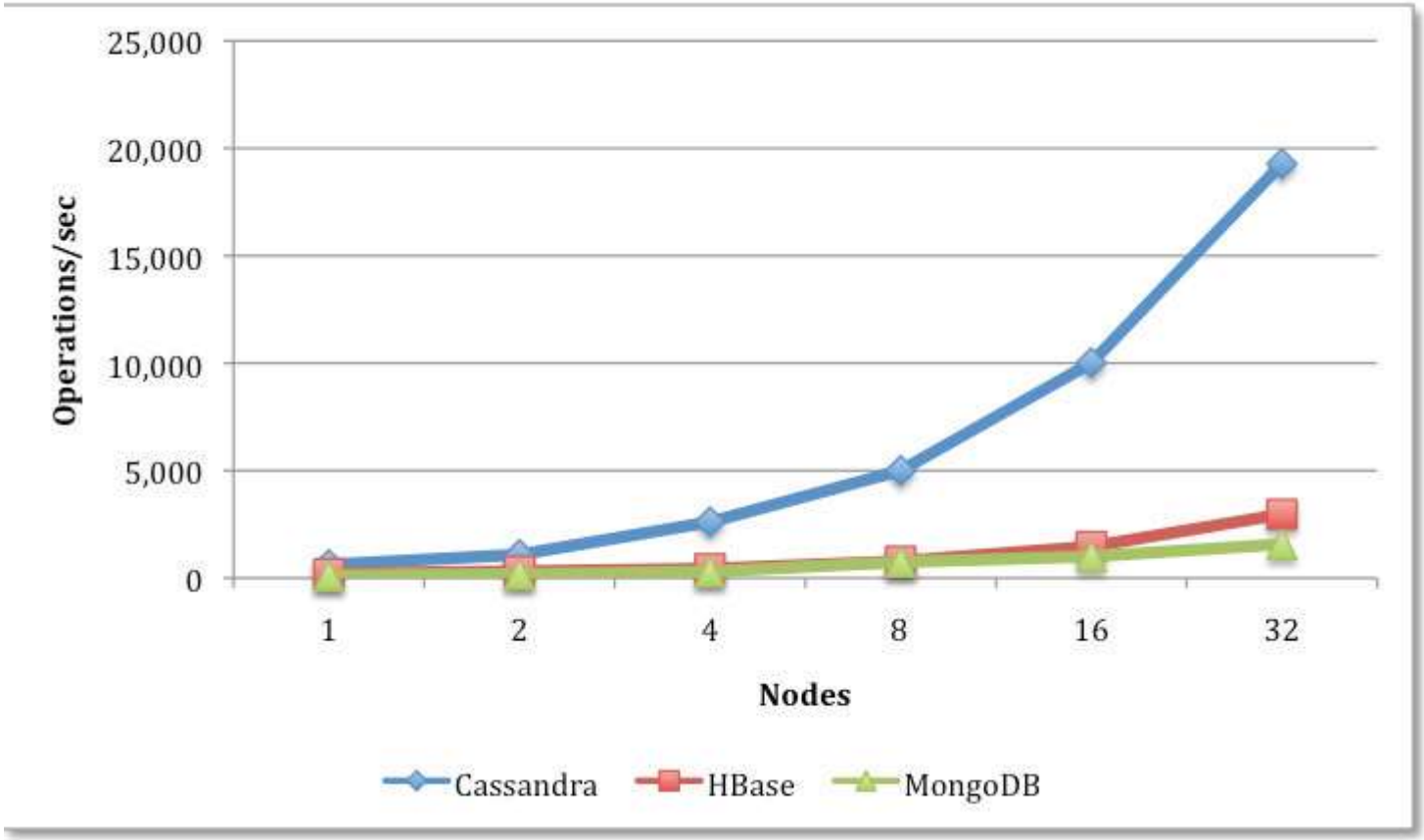
- The tests ran in the cloud on Amazon Web Services (AWS) EC2 instances, using spot instances to ensure cost efficiency while getting the same level of performance.
- The tests ran exclusively on m1.xlarge size instances (15 GB RAM and 4 CPU cores) using local instance storage for performance. The m1.xlarge instance type allows for up to 4 local instance devices; the instances were allocated all 4 block devices, which were then combined on boot into a single 1.7TB RAID-1 volume.
- The instances use customized Ubuntu 12.04 LTS AMI's with Oracle Java 1.6 installed as a base.
- On start up, each instance calls back to a parent instance for its configuration. A customized script was written to drive the benchmark process, including managing the start up, configuration, and termination of EC2 instances, calculation of workload parameters, and driving the clients to run the tests.

Tested Workloads

The following workloads were included in the benchmark:

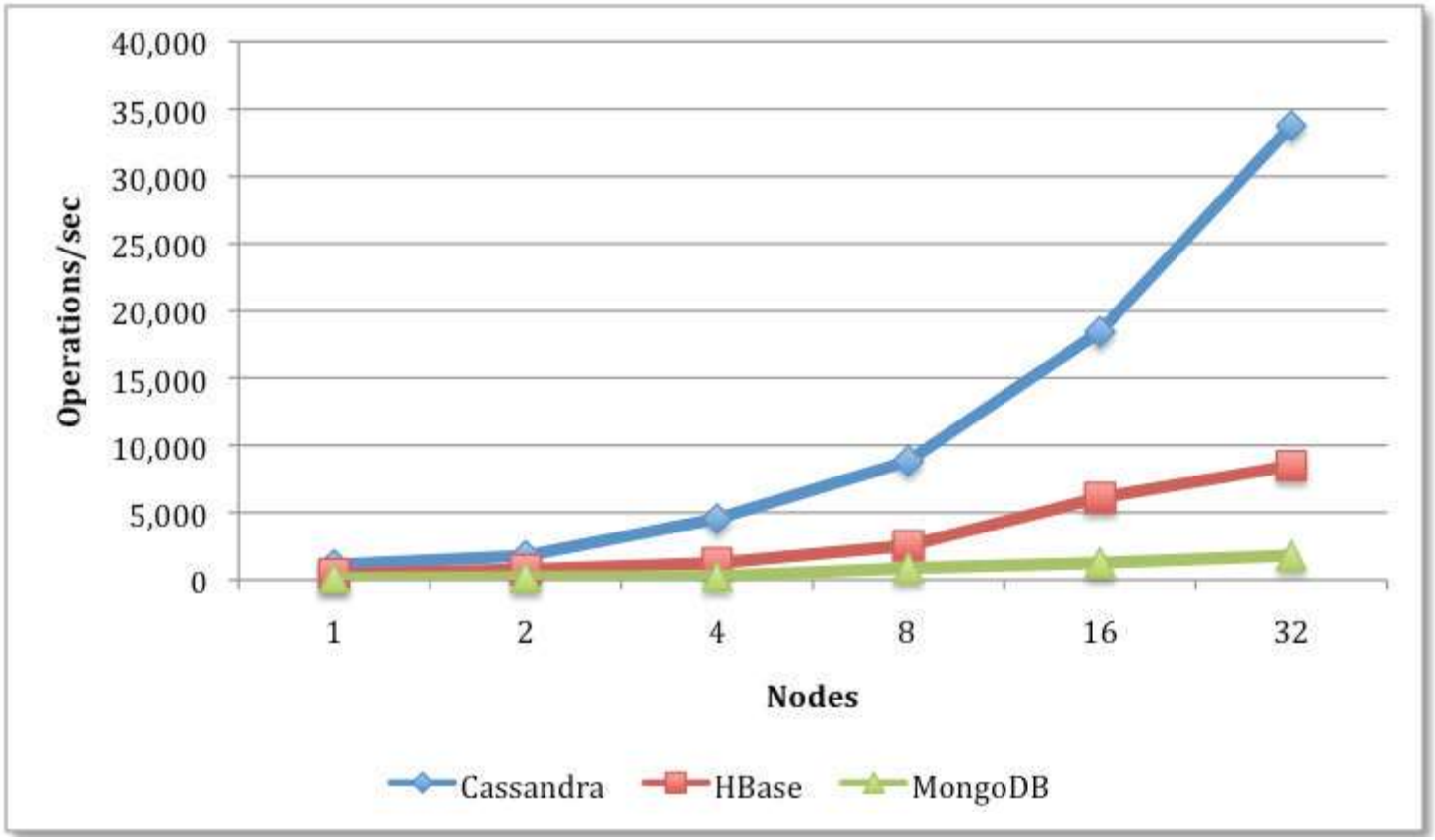
1. Read-mostly workload, based on YCSB's provided workload B: 95% read to 5% update ratio
2. Read/write combination, based on YCSB's workload A: 50% read to 50% update ratio
3. Write-mostly workload: 99% update to 1% read
4. Read/scan combination: 47% read, 47% scan, 6% update
5. Read/write combination with scans: 25% read, 25% scan, 25% update, 25% insert
6. Read latest workload, based on YCSB workload D: 95% read to 5% insert
7. Read-modify-write, based on YCSB workload F: 50% read to 50% read-modify-write

Read-Mostly Workload



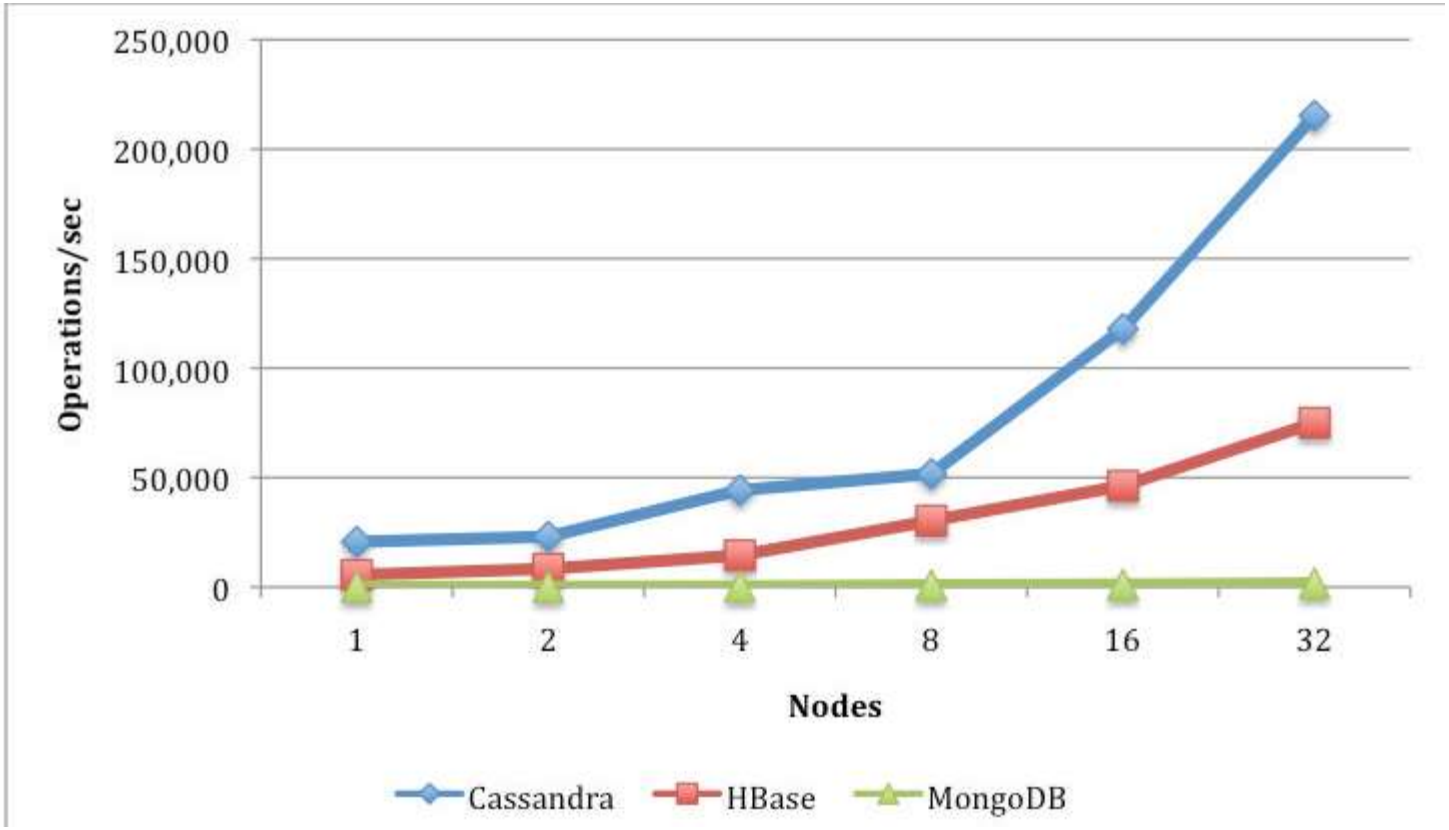
- Different workloads are important for different use cases and one should really examine the intended usage of a NoSQL database before deciding which product to use.

Read/Write Mix Workload



Complete white paper on this series of tests can be found at:
http://www.datastax.com/resources/whitepapers/benchmarking-top-nosql-databases

Write-mostly Workload



- Here we presented only 3 of 8 tested workload scenarios. It appears that in all of them Cassandra demonstrated the highest throughput.

Choose

- Based on results for all tested workloads, it appears that Cassandra is an overwhelming winner.
- Cassandra has some “deficiencies”. For example, it does not implement a very strong consistency model and does not consider ACID properties to be sacrosanct.
- ACID (*Atomicity, Consistency, Isolation, Durability*) is a set of properties that guarantee that database transactions are processed reliably. Old fashioned RDBMS-s like Oracle, DB2, PostgreSQL, MySQL, and others guaranty that those properties will hold in any of your transactions.
- HBase, for example, pays due respect to Consistency property and you might decide to select HBase over Cassandra if data consistency is critical for your application, in spite of HBase’s less-than-stellar performance.
- Also, Cloudera which “owns” Hbase claims that they made significant improvements recently. Also, if your data is all in HDFS or you want it in HDFS, HBase is a natural choice.
- In the following we will look at some properties of Cassandra, HBase and MongoDB.

Cassandra

Cassandra

- In the following we will examine the installation process for Cassandra Database and review some of its properties.
- Cassandra was developed at Facebook.
- Major promoter and contributor to Cassandra is DataStax corporation. The same people who are commercializing Spark.
- Cassandra is an Apache project. The latest stable version is 3.3.0
- Cassandra is available for
  - Windows (Windows Server 2008/2012, Windows 7/8 (64 bit)
  - Mac OS x10.x, Unix
  - Red hat Enterprise Linux, CentoOS 6.5+
  - Latest Ubuntu LTS and Debian Linux
- Client Drivers exist for:
  - Java, Python, Ruby, C#/.NET, Go, Node.js, Clojure, PHP and C++

What is your OS

- You can find out what is the Linux Operating System you are working with by listing the following files:

```
[cloudera@quickstart ~]$ cat /etc/issue
CentOS release 6.4 (Final)
Kernel \r on an \m

[cloudera@quickstart ~]$ cat /etc/centos-release
CentOS release 6.4 (Final)
[cloudera@quickstart ~]$
```

- It appears that Cloudera VM runs `CentOS 6.4`

Create Linux User `cassandra`

- Casandra server should be run by Linux user `cassandra`. Let us create it

```
$ su -          # to become root
$ groupadd cassandr          # greate Linux group cassandra
$ useradd cassandra -g cassandra    # create Linux user cassandra
$ passwd cassandra          # set password for user  cassandra
Enter password: cassandra
```

- Make user `cassandra` a `sudo` user

```
$ chmod + w /etc/sudoers          # make /etc/sudoers writable
$ visudo -f /etc/sudoers          # open the file with special Vi editor
```

- Add a line for user `cassandra similar to the line for user cloudera`. Also, tell the system to remember `JAVA_HOME` and PATH variables when executing as a `sudo` user:

```
cloudera ALL=(ALL) NOPASSWD: ALL
cassandra ALL=(ALL) NOPASSWD: ALL
Defaults  env_keep += "LC_TIME LC_ALL LANGUAGE LINGUAS _XKB_CHARSET XAUTHORITY"
Defaults  env_keep += "JAVA_HOME PATH"
```

- Make file /etc/sudoers read only.

```
$ chmod -w /etc/sudoers
```

Installing Cassandra using `yum`

- For newest versions of Cassandra, DataStax recommends Java 8. Our VM has Java `1.7.0_67`. That is why we have to install an older version of Cassandra, `2.1.13`
- Cassandra 2.1 requires Python 2.6. Cloudera VM has 2.6.6 . We are OK.
- To check which version of Java is installed, run the command:

```
$ java -version
```

- Create file `/etc/yum.repos.d/datastax.repo` referring to the DataStax Community repository. Add the following lines to the file:

```
[datastax]
name = DataStax Repo for Apache Cassandra
baseurl = https://rpm.datastax.com/community
enabled = 1
gpgcheck = 0
```

- Install the packages:

```
$ sudo yum install dsc21-2.1.13-1 cassandra2.1.6-1
$ sudo yum install cassandra21-tools-2.1.13-1  ##Installs optional utilities.
```

- You are done. As user `cassandra` find whether Cassandra executable exists:

```
$ which cassandra
/usr/sbin/Cassandra
```

- Start Cassandra server as Linux user `cassandra` by typing: `$ cassandra &`

Alternate Installations and Prerequisites

- We could also use the binary tarball
- Download the tarball using the URL for Cassandra 2.1.13 which you will find on this page:
  `http://cassandra.apache.org/download/`
- Become user `cassandra`

```
$ su – Cassandra
Password: cassandra
```

Move the file from `/home/cloudera/Downloads` to `/home/cassandra` . Expand it:

```
$ tar -xzvf apache-cassandra-2.1.13-bin.tar.gz
```

- Remove residual tar.gz file and rename directory `apache-cassandra-2.1.13 to cassandra-2.1.13`

```
$  mv apache-cassandra-2.1.13 cassandra-2.1.13
```

- Place `/home/cassandra/cassandra-2.1.13/bin in cassandra's .bash_profile` file. Source the file:

```
$ source .bash_profile
```

Starting Cassandra Server

- To start `Cassandra` server, as Linux user `cassandra`, in directory

`~cassandra/cassandra-2.1.13/bin`, type:

```
$ cassandra –f
```

- The switch `-f` tells Cassandra to stay in the foreground instead of running as a background process. This so that all of the server logs will print to the standard out and you can see them in your terminal window, which is useful for testing.
- To stop `Cassandra` type `Ctrl C` in the console.
- Under normal circumstances, running server will write a fairly long output to the console. The output starts and ends with lines like the ones on the next two page. There is interesting and important information in those line. For example, you can find out where Cassandra writes its logs and data.

Console output, beginning, Normally Running Server

```
[cassandra@quickstart bin]$ ./cassandra -f
CompilerOracle: inline org/apache/cassandra/db/AbstractNativeCell.compareTo
(Lorg/apache/cassandra/db/composites/Composite;)I
CompilerOracle: inline
org/apache/cassandra/db/composites/AbstractSimpleCellNameType.compareUnsigned
(Lorg/apache/cassandra/db/composites/Composite;Lorg/apache/cassandra/db/composites/Composite;
)I
CompilerOracle: inline org/apache/cassandra/io/util/Memory.checkBounds (JJ)V
CompilerOracle: inline org/apache/cassandra/io/util/SafeMemory.checkBounds (JJ)V
CompilerOracle: inline org/apache/cassandra/utils/ByteBufferUtil.compare
(Ljava/nio/ByteBuffer;[B)I
CompilerOracle: inline org/apache/cassandra/utils/ByteBufferUtil.compare
([BLjava/nio/ByteBuffer;)I
CompilerOracle: inline org/apache/cassandra/utils/ByteBufferUtil.compareUnsigned
(Ljava/nio/ByteBuffer;Ljava/nio/ByteBuffer;)I
CompilerOracle: inline
org/apache/cassandra/utils/FastByteOperations$UnsafeOperations.compareTo
(Ljava/lang/Object;JILjava/lang/Object;JI)I
CompilerOracle: inline
org/apache/cassandra/utils/FastByteOperations$UnsafeOperations.compareTo
(Ljava/lang/Object;JILjava/nio/ByteBuffer;)I
CompilerOracle: inline
org/apache/cassandra/utils/FastByteOperations$UnsafeOperations.compareTo
(Ljava/nio/ByteBuffer;Ljava/nio/ByteBuffer;)I
INFO  18:36:12 Hostname: quickstart.cloudera
INFO  18:36:12 Loading settings from file:/home/cassandra/cassandra-
2.1.13/conf/cassandra.yaml
INFO  18:36:12 Node configuration:[authenticator= . . . . . .
```

Console output, end

```
INFO  18:05:51 Completed flushing
/home/cassandra/cassandra/bin/../data/data/system/local-
7ad54392bcdd35a684174e047860b377/system-local-tmp-ka-9-Data.db (0.000KiB)
for commitlog position ReplayPosition(segmentId=1457719550068,
position=105384)
INFO  18:05:51 Node localhost/127.0.0.1 state jump to NORMAL
INFO  18:05:51 Compacted 4 sstables to
[/home/cassandra/cassandra/bin/../data/data/system/local-
7ad54392bcdd35a684174e047860b377/system-local-ka-10,].  6,361 bytes to
5,779 (~90% of original) in 68ms = 0.081048MB/s.  4 total partitions merged
to 1.  Partition merge counts were {4:1, }
INFO  18:05:51 Netty using native Epoll event loop
INFO  18:05:51 Using Netty Version: [netty-buffer=netty-buffer-
4.0.23.Final.208198c, netty-codec=netty-codec-4.0.23.Final.208198c, netty-
codec-http=netty-codec-http-4.0.23.Final.208198c, netty-codec-socks=netty-
codec-socks-4.0.23.Final.208198c, netty-common=netty-common-
4.0.23.Final.208198c, netty-handler=netty-handler-4.0.23.Final.208198c,
netty-transport=netty-transport-4.0.23.Final.208198c, netty-transport-
rxtx=netty-transport-rxtx-4.0.23.Final.208198c, netty-transport-sctp=netty-
transport-sctp-4.0.23.Final.208198c, netty-transport-udt=netty-transport-
udt-4.0.23.Final.208198c]
INFO  18:05:51 Starting listening for CQL clients on
localhost/127.0.0.1:9042...
INFO  18:05:51 Binding thrift service to localhost/127.0.0.1:9160
INFO  18:05:51 Listening for thrift clients...
```

Directories used by cassandra

- When normally installed with yum or apt, Cassandra uses the following directories:
  - For configuration settings `$CASSANDRA_INSTALL/conf/cassandra.yaml`
  - data_file_directories (`/var/lib/cassandra/data`),
  - commitlog_directory (`/var/lib/cassandra/commitlog`), and
  - saved_caches_directory (`/var/lib/cassandra/saved_caches`).
- Make sure these directories exist and can be written to.
- By default, Cassandra will write its logs in `/var/log/cassandra/`.
- All these directories are owned by user `cassandra` and that is the reason why you should start the server as that Linux user.
- In Cassandra 2.1+, the logger in use is `logback`, so change this logging directory in your `conf/logback.xml` file such as:

```
<file>/var/log/cassandra/system.log</file>
```

- JVM-level settings such as heap size can be set in `conf/cassandra-env.sh`.
- In the situation when Casandra is installed from the tarball the way we did it, all the above directories will be under:
  `$CASSANDRA_INSTALL=/home/cassandra/cassandra-2.1.13`

Command Line Interface,

- `$CASSANDRA_INSTALL/bin/cqlsh` is the command line interface for examining data in Cassandra server.
- You can define the schema and interact with data using `cqlsh`.
- Run the following command to connect to your local Cassandra instance:

```
$ ./cqlsh
Connected to Test Cluster at localhost:9160. [cqlsh 5.0.1 | Cassandra
2.1.13 | CQL spec 3.2.1 | Native protocol v3]
Use HELP for help. Commands are terminated with a semicolon (;)
cqlsh>
```

- First, create a keyspace -- a namespace of tables.

```
CREATE KEYSPACE mykeyspace WITH REPLICATION = { 'class' :
'SimpleStrategy', 'replication_factor' : 1 };
USE mykeyspace;      # Move to new keyspace , create a table, insert data
CREATE TABLE users (
 user_id int PRIMARY KEY,
 fname text,
 lname text );
INSERT INTO users (user_id, fname, lname) VALUES (1745, 'john', 'smith');
INSERT INTO users (user_id, fname, lname) VALUES (1744, 'john', 'doe');
INSERT INTO users (user_id, fname, lname) VALUES (1746, 'john', 'smith');
```

CQL, Retrieve Data, Create Index

- We can retrieve data using customary tools and commands

```
cqlsh> SELECT * FROM users;
user_id | fname | lname
--------+-------+-------
   1745 | john  | smith
   1744 | john  | doe
   1746 | john  | smith
```

- You can retrieve data faster (about users whose last name is smith) by creating an index, then querying the table as follows:

```
CREATE INDEX ON users (lname);
SELECT * FROM users WHERE lname = 'smith';
 user_id | fname | lname
---------+-------+-------
    1745 | john  | smith
    1746 | john  | smith
```

- The language we are using here is called CQL of Cassandra Query Language which is very similar to SQL.
- If you watch the console, you will see that the console reports on your activity

Cassandra Data Model

- Cassandra's data model is a partitioned row store with tunable consistency.
- Rows are organized into tables; the first component of a table's primary key is the partition key; within a partition, rows are clustered by the remaining columns of the key. Other columns can be indexed separately from the primary key.
- Tables can be created, dropped, and altered at runtime without blocking updates and queries.
- Cassandra does not support joins or subqueries, except for batch analysis through Hadoop.
- Rather, Cassandra emphasizes denormalization through features like collections

Example, Music Service: Playlist/Songs

- A social music service requires a songs table having a title, album, and artist column, plus a column called data for the actual audio file itself.
- The table uses a UUID as a primary key.

```
CREATE TABLE songs (
id uuid PRIMARY KEY,
title text, album text, artist text, data blob );
# playlist_id  );# In relational design, playlist_id would be the foreign key to playlists
```

- in Cassandra, we denormalize the data. playlist data, is presented by table:

```
CREATE TABLE playlists ( id uuid,
song_order int, song_id uuid, # In relational design these fields
title text, album text, artist text,
PRIMARY KEY (id, song_order ) ); # would not be here
```

- The combination of the id and song_order in the playlists table uniquely identifies a row in the playlists table. We could have more than one row with the same `(playlist) id` as long as the rows contain different `song_order` values.

```
CREATE TABLE playlist_songs (  # Relational design needs another
playlistid uuid,               # table to resent song order
song_id    uuid,
song_order int);
```

Example, Music Service

- If one would like to select playlist content for a particular artist, as things are organized now, Cassandra would have to make a full table scan. To avoid doing that you create an index:

```
CREATE INDEX ON playlists(artist );
```

- Now, you can query the playlists for songs by Fu Manchu, for example:

```
SELECT * FROM playlists WHERE artist = 'Fu Manchu';
```

- and Cassandra will efficiently return all relevant rows.
- Notice that the primary key on playlists table had two elements:

```
PRIMARY KEY (id, song_order ) );
```

- The first portion is called the partition key. The remaining keys are called clustering keys.
- Cassandra stores data on a node by partition key. If you have too much data in a partition and want to spread the data over multiple nodes, use a composite partition key.
- One consideration is whether to use surrogate or natural keys for a table.
- A surrogate key is a generated key (such as a UUID) that uniquely identifies a row, but has no relation to the actual data in the row.

Compound Keys and Clustering

- A compound primary key includes the partition key, which determines on which node data is stored first, and one or more additional columns that determine clustering.
- Cassandra uses the first column name in the primary key definition as the partition key. For example, in the playlists table, `id` is the partition key.
- The remaining column, or columns that are not partition keys in the primary key definition are the clustering columns. In the case of the playlists table, the `song_order` is the clustering column.
- The data for each partition is clustered by the remaining column or columns of the primary key definition.
- On a physical node, when rows for a partition key are stored in order based on the clustering columns, retrieval of rows is very efficient. For example, because the `id` in the playlists table is the partition key, all the songs for a playlist are clustered in the order of the remaining `song_order` column.
- Insertion, update, and deletion operations on rows sharing the same partition key for a table are performed atomically and in isolation.

Collection Columns

- CQL introduces three collection types:
- set, list, and map
- In a relational database, to allow users to have multiple email addresses, you usually create an `email_addresses` table having a many-to-one (joined) relationship to the `users` table.
- CQL handles the classic multiple email addresses use case, and other use cases, by defining columns as collections.
- For example, to add a collection named `tag` with elements of type `text` to table `songs`, we would issue statement like this:

```
ALTER TABLE songs ADD tags set<text>;
```

- One updates collection columns using + operator

```
UPDATE songs SET tags = tags + {'2007'}
WHERE id = 8a172618-b121-4136-bb10-f665cfc469eb;
UPDATE songs SET tags = tags + {'covers'}
WHERE id = 8a172618-b121-4136-bb10-f665cfc469eb;
```

Querying Collections

- To query a collection, include the name of the collection column in the select expression.
- For example, selecting the tags set returns the set of tags, sorted alphabetically in this case because the tags set is of the text data type:

```
SELECT id, tags FROM songs;
```

```
                id                                   | tags
-------------------------------------+----------------
 7db1a490-5878-11e2-bcfd-0800200c9a66 |         {rock}
 a3e64f8f-bd44-4f28-b8d9-6938726e34d4 |  {blues, 1973}
 8a172618-b121-4136-bb10-f665cfc469eb | {2007, covers}
```

Indexing

- An index in Cassandra refers to an index on column values. Cassandra implements an index as a hidden table, separate from the table that contains the values being indexed. Using CQL, you can create an index on a column after defining a table.

```
CREATE INDEX artist_names ON playlists( artist );
```

- An index name is optional. If you provide an index name, such as `artist_idx`, the name must be unique within the keyspace.
- After creating an index for the artist column and inserting values into the playlists table, greater efficiency is achieved when you query Cassandra directly for artist by name, such as Fu Manchu:

```
SELECT * FROM playlists WHERE artist = 'Fu Manchu';
```

- As mentioned earlier, when looking for a row in a large partition, narrow the search. This query, although a contrived example using so little data, narrows the search to a single id.

```
SELECT * FROM playlists WHERE id = 62c36092-82a1-3a00-93d1-
46196ee77204 AND artist = 'Fu Manchu';
```

Using Multiple Indexes

- One can create multiple indexes on different columns of a table. For example:

```
CREATE INDEX album_name ON playlists ( album );
CREATE INDEX title_name ON playlists ( title );
SELECT * FROM playlists
WHERE album = 'Roll Away' AND title = 'Outside Woman Blues' ALLOW
FILTERING ;
```

- When multiple occurrences of data match a condition in a WHERE clause, Cassandra selects the least-frequent occurrence of a condition for processing first for efficiency.
- For example, suppose data for Blind Joe Reynolds and Cream's versions of "Outside Woman Blues" were inserted into the playlists table. Cassandra queries on the album name first if there are fewer albums named Roll Away than there are songs called "Outside Woman Blues" in the database

DataStax Java Driver 2.0 for Apache Cassandra

- Cassandra has many drivers for several popular languages. This is perhaps a testimony to its popularity.
- We will present a Java driver maintained by Datastax Corp. Our hope is that a driver maintained by a commercial entity will have a long shelf life.
- Material on the following slides could be found at:

http://www.datastax.com/documentation/developer/java-driver/2.0/common/drivers/introduction/introArchOverview_c.html

- The Java Driver 2.0 for Apache Cassandra works exclusively with the Cassandra Query Language version 3 (CQL3) and Cassandra's new binary protocol which was introduced in Cassandra version 1.2.
- The driver architecture is a layered. At the bottom lies the driver core. This core handles everything related to the connections to a Cassandra cluster (for example, connection pool, discovering new nodes, etc.) and exposes a simple, relatively low-level API on top of which a higher level layer can be built.
- A Mapping and a JDBC module will be added on top.

DataStax Java Driver 2.0 for Apache Cassandra

- The driver relies on Netty to provide non-blocking I/O with Cassandra for providing a fully asynchronous architecture.
- Netty is *an asynchronous event-driven NIO client-server network application framework* for rapid development of maintainable high performance protocol servers & clients. (see: http://netty.io)
- Multiple queries can be submitted to the driver which then will dispatch the responses to the appropriate client threads.
- The driver has the following features:
  - connection pooling
  - node discovery
  - automatic failover
  - load balancing
- The default behavior of the driver can be changed or fine tuned by using tuning policies and connection options.
- Queries can be executed synchronously or asynchronously, prepared statements are supported, and a query builder auxiliary class can be used to build queries dynamically.

Dependencies

- The Java driver only supports the Cassandra Binary Protocol and CQL3

Cassandra binary protocol

- The driver uses the binary protocol that was introduced in Cassandra 1.2. It only works with a version of Cassandra greater than or equal to 1.2. Furthermore, the binary protocol server is not started with the default configuration file in Cassandra 1.2. Edit the `cassandra.yaml` file for each node and set: `start_native_transport: true.` Then restart the node.

Maven dependencies

- The latest release of the driver is available on Maven Central. You can install it in your application using the following Maven dependency:

```
<dependency>
        <groupId>com.datastax.cassandra</groupId>
        <artifactId>cassandra-driver-core</artifactId>
        <version>2.0.10</version>
</dependency>
```

Download Driver

- From the site

http://downloads.datastax.com/java-driver/cassandra-java-driver-2.1.6.tar.gz

- download binary driver and use provide jar files.
- Binary driver downloads as `cassandra-java-driver-2.1.6.tar.gz`
- Untar the file using command

```
$ tar zxvf cassandra-java-driver-2.1.6.tar
```

- In the resulting directory, you will see two jar files:
  `cassandra-driver-core-2.1.6.jar` and
  `cassandra-driver-dse-2.1.6 lin`

Also, in the subdirectory `lib` you will see a bunch of additional jars:

`bsh-2.0b4.jar, guava-16.0.1.jar, jcommander-1.27.jar, log4j-1.2.17.jar, lz4-1.2.0.jar, metrics-core-3.0.2.jar, netty-3.9.0.Final.jar, slf4j-api-1.7.5.jar, slf4j-log4j12-1.7.6.jar,` and `snappy-java-1.0.5.jar`

- All of those jars need to be referenced in the `CLASSPATH` variable of your Java applications or in the `Build Path` of your Java projects.

Install Eclipse

- It is convenient to create Java applications using Eclipse.
- Eclipse can be installed on your Debian Linux VM. This not difficult at all. Eclipse for 64 bit Linux is packaged as tar.gz file. Download and un-`tar` the file. In the resulting directory "`eclipse`" find executable "`eclipse`". Run the executable from the command line as

```
$./eclipse
```

- the familiar GUI will open. If you want to be fancy, copy your directory `eclipse` to `/usr/local` directory, and then create a symbolic link in `/usr/bin` directory pointing to the `eclipse` executable:

```
$ sudo ln –s /usr/local/eclipse/eclipse /usr/bin/eclipse
```

- Make sure that any Linux user can run the executable by issuing command:

```
$ sudo +x /usr/bin/eclipse
```

- Now you can open start `eclipse` from any directory.
- Client applications will connect to Cassandra cluster and run some database commands: create database tables, modify tables, insert, update or delete objects (rows) in those tables.

Creating Client

- In Eclipse, before creating the client class, create a project. You can call the project `CassandraClient`, for example.
- Our client will be the class, `edu.hu.cassandra.SimpleClient`
- We will need an instance field, `cluster`, to hold a Cluster reference.

```
private Cluster cluster;
```

- We will add an instance method, `connect()`, to our new class.

```
public void connect(String node) {}
```

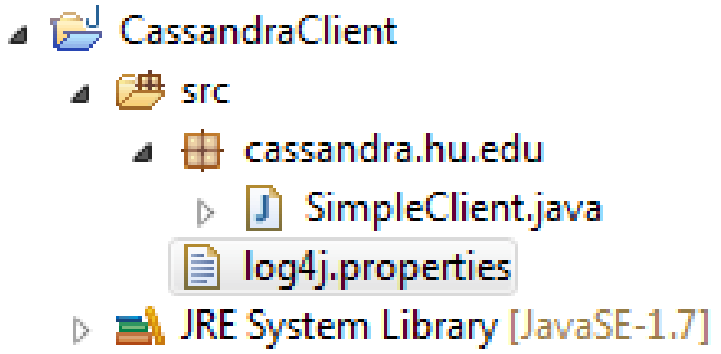
- The `connect()` method:
  - adds a contact point (node IP address) using the `Cluster.Build` auxiliary class, builds a cluster instance,
  - retrieves metadata from the cluster and prints out:
    - the name of the cluster, the datacenter, host name or IP address, and rack for each of the nodes in the cluster
- We could run the client class as a Java application from the Eclipse itself.
- Complete code of this class is presented on one of the following slides.

Create Eclipse Project, add `log4j.properties` File

- Before running the client class, create a file named: `log4j.properties`, with a content similar to the following:

```
log4j.rootLogger=debug, stdout, R
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
# Pattern to output the caller's file name and line number.
log4j.appender.stdout.layout.ConversionPattern=%5p [%t](%F:%L) - %m%n
log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=example.log
log4j.appender.R.MaxFileSize=100KB
# Keep one backup file
log4j.appender.R.MaxBackupIndex=1
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n
```

- Place file `log4j.properties` in the `src` directory of your Eclipse project, like in the image to the right. This is important. Otherwise, the driver will complain that your Log4J is not properly initialized.



edu.hu.cassandra.SimpleClient

```
package edu.hu.cassandra;
import com.datastax.driver.core.Cluster; import com.datastax.driver.core.Host;
import com.datastax.driver.core.Metadata;
public class SimpleClient {
   private Cluster cluster;
   public void connect(String node) {
      cluster = Cluster.builder().addContactPoint(node).build();
      Metadata metadata = cluster.getMetadata();
      System.out.printf("Connected to cluster: %s\n",
            metadata.getClusterName());
      for ( Host host : metadata.getAllHosts() ) {
         System.out.printf("Datatacenter: %s; Host: %s; Rack: %s\n",
               host.getDatacenter(), host.getAddress(), host.getRack());
      }
   }
   public void close() {
      cluster.close(); }
   public static void main(String[] args) {
      SimpleClient client = new SimpleClient();
      client.connect("127.0.0.1");
      client.close();
   }
}
```

Running your Client

- When you right clink on your Java client class, select Run as Java Application.
- The result should look something like this:
- ```
  DEBUG [main] (Cluster.java:930) - Starting new cluster with
  contact points [/127.0.0.1]
  DEBUG [main] (ControlConnection.java:226) - [Control connection]
  Refreshing node list and token map
  DEBUG [main] (ControlConnection.java:229) - [Control connection]
  Refreshing schema
  DEBUG [main] (ControlConnection.java:154) - [Control connection]
  Successfully connected to /127.0.0.1
  Connected to cluster: Test Cluster
  Datatacenter: datacenter1; Host: /127.0.0.1; Rack: rack1
  DEBUG [main] (Cluster.java:1037) - Shutting down
  ```

- This basically tells you that your Java client did connect to Cassandra server (cluster). Not bad at all.
  ```

Client that executes SQL Commands

- We will create a client class, `CQLClient` and enable it to execute CQL (SQL like) commands. We start building `CQLClient` by copying the content of SimpleClient class. Subsequently, we add new code.
- In order to execute CQL commands we need to create an object of type:

```
com.datastax.driver.core.Session;
```

- We do that by adding an instance field, `session`, of type `Session`:

```
private Session session;
```

- We get the session from the `cluster` object using method `connect()`:

```
session = cluster.connect();
```

- Queries are executed by calling the `execute()` method on the `session` object. The `session` maintains multiple connections to the cluster nodes, provides policies to choose which node to use for each query (round-robin on all nodes of the cluster by default), and handles retries for failed queries when it makes sense.
- A given `session` can only be set to one keyspace at a time, so one instance per `keyspace` is necessary. Your application typically only needs a single cluster object, unless you're dealing with multiple physical clusters

Create Schema

- We need to add an instance method that will create Schema.

```
public createSchema() { }
```

- Inside this method we will execute an CQL command to create a keyspace `simplex`, i.e.

```
session.execute("CREATE KEYSPACE simplex WITH replication " +
  "= {'class':'SimpleStrategy', 'replication_factor':3};");
```

- We will also execute statements to create two new tables, `songs` and `playlists`.

```
session.execute( "CREATE TABLE simplex.songs (" + "id uuid
PRIMARY KEY," + "title text," + "album text," + "artist text," +
"tags set<text>," + "data blob" + ");");
session.execute( "CREATE TABLE simplex.playlists (" + "id uuid,"
+ "title text," + "album text, " + "artist text," + "song_id
uuid," + "PRIMARY KEY (id, title, album, artist)" + ");");
```

Load Data

- Data will be loaded using instance method `loadData()`

```
public void loadData() { }
```

- The code that performs Insert statements for two table looks like the following:

```
session.execute(
"INSERT INTO simplex.songs (id, title, album, artist, tags) " +
"VALUES (" + "756716f7-2e54-4715-9f00-91dcbea6cf50," +
"'La Petite Tonkinoise'," + "'Bye Bye Blackbird'," +
"'Joséphine Baker'," + "{'jazz', '2013'})" + ";");

session.execute(
"INSERT INTO simplex.playlists (id, song_id, title, album, " +
" artist) " +
"VALUES (" + "2cc9ccb7-6221-4ccb-8387-f22b6a1b354d," +
"756716f7-2e54-4715-9f00-91dcbea6cf50," +
"'La Petite Tonkinoise'," +
"'Bye Bye Blackbird'," + "'Joséphine Baker'" + ");");
```

Select content of `playlists` table

- Lastly, we will query the content of the `playlists` table. For that purpose we add instance method `querySchema()`.
- Inside querySchema() method we execute query (select statement) which returns an object of type: `com.datastax.driver.core.ResultSet`. `ResultSet` object contains objects of type `com.datastax.driver.core.Row` which represents rows returned by the select statement.

```
ResultSet results = session.execute("SELECT * FROM
simplex.playlists " + "WHERE id = 2cc9ccb7-6221-4ccb-8387-
f22b6a1b354d;");
```

- To see individual rows we iterate through the `ResultSet`

```
for (Row row : results) {
   System.out.println(String.format("%-30s\t%-20s\t%-20s",
        row.getString("title"), row.getString("album"),
        row.getString("artist")));
}
```

`main()` Method

- Method main() organizes above instance methods in the proper order:

```
public static void main(String[] args) {
   CQLClient client = new CQLClient();
   client.connect("127.0.0.1");
   client.createSchema();
    client.loadData();
   client.querySchema();
   client.close();
}
```

- We should note that it is wise and necessary to close or end cluster connection. We do that using method `close()`, defined as:

```
public void close() {
   cluster.close(); // .shutdown();
}
```

- Complete code of class CQLClient is presented on the following slides.

CQLClient.java

```
package cassandra.hu.edu;

import com.datastax.driver.core.Cluster;
import com.datastax.driver.core.Host;
import com.datastax.driver.core.Metadata;
import com.datastax.driver.core.Session;
import com.datastax.driver.core.ResultSet;
import com.datastax.driver.core.Row;

public class CQLClient {
   private Cluster cluster;
   private Session session;
   public void connect(String node) {
      cluster = Cluster.builder()
            .addContactPoint(node).build();
      session = cluster.connect();
   }
```

CQLClient.createSchema()

```
public void createSchema() {
   session.execute("CREATE KEYSPACE simplex WITH replication " +
      "= {'class':'SimpleStrategy', 'replication_factor':1};");
   session.execute(
      "CREATE TABLE simplex.songs (" +
            "id uuid PRIMARY KEY," +
            "title text," + "album text," +
            "artist text," + "tags set<text>," +
            "data blob" +
            ");");
  session.execute(
      "CREATE TABLE simplex.playlists (" +
            "id uuid," +
            "title text," + "album text, " +
            "artist text," + "song_id uuid," +
            "PRIMARY KEY (id, title, album, artist)" +
            ");");
  }
```

CQLClient.loadData()

```
public void loadData() {
  session.execute("INSERT INTO simplex.songs (id, title,album," 
+ "artist, tags)" +
      "VALUES (" +
          "756716f7-2e54-4715-9f00-91dcbea6cf50," +
          "'La Petite Tonkinoise'," +
          "'Bye Bye Blackbird'," +
          "'Joséphine Baker'," +
          "{'jazz', '2013'})" +
          ";");
session.execute( "INSERT INTO simplex.playlists (id, song_id," +
" title, album, artist) " +
      "VALUES (" +
          "2cc9ccb7-6221-4ccb-8387-f22b6a1b354d," +
          "756716f7-2e54-4715-9f00-91dcbea6cf50," +
          "'La Petite Tonkinoise'," +
          "'Bye Bye Blackbird'," +
          "'Joséphine Baker'" +
          ");");
}
```

CQLClient.querySchema()

```
public void querySchema(){
  ResultSet results = session.execute(
"SELECT * FROM simplex.playlists " +
        "WHERE id = 2cc9ccb7-6221-4ccb-8387-
f22b6a1b354d;");System.out.println(String.format(
"%-30s\t%-20s\t%-20s\n%s", "title", "album", "artist",
"------------------------------+--------------------" +
"---+---------------------"));
   for (Row row : results) {
    System.out.println(String.format("%-30s\t%-20s\t%-20s",
    row.getString("title"),
    row.getString("album"),  row.getString("artist")));
   }
   System.out.println();
}
public void close() {
      cluster.close(); // .shutdown();
}
```

CQLClieNT.main(String[] arg)

```
public static void main(String[] args) {
        CQLClient client = new CQLClient();
        client.connect("127.0.0.1");
        client.createSchema();   client.loadData();
        client.querySchema();   client.close();
 }
}
```

- The output on the Eclipse console is presented on the next slide. We have removed a few DEBUG lines to fit the output into one slide.
- One can open `cqlsh` prompt, issue command:

```
cqlsh> user simplex;
```

- and then, examine the content of table `playlists`:

```
cqlsh:simplex> select * from playlists;

 id                                   | title                | album
 | artist          | song_id
--------------------------------------+----------------------+-----------------
-+-----------------+--------------------------------------
 2cc9ccb7-6221-4ccb-8387-f22b6a1b354d | La Petite Tonkinoise | Bye Bye Blackbird
 | Joséphine Baker | 756716f7-2e54-4715-9f00-91dcbea6cf50

(1 rows)
```

Console Output of CQLClient

```
DEBUG [main] (Cluster.java:930) - Starting new cluster with contact points [/127.0.0.1]
DEBUG [Cassandra Java Driver worker-0] (SessionManager.java:238) - Adding /127.0.0.1 to list of queried hosts
Connected to cluster: Test Cluster
Datatacenter: datacenter1; Host: /127.0.0.1; Rack: rack1
DEBUG [New I/O worker #1] (Cluster.java:1466) - Received event EVENT CREATED simplex, scheduling delivery
DEBUG [New I/O worker #1] (Cluster.java:1466) - Received event EVENT CREATED simplex.songs, scheduling delivery
DEBUG [New I/O worker #2] (Cluster.java:1433) - Refreshing schema for simplex
DEBUG [New I/O worker #1] (Cluster.java:1466) - Received event EVENT CREATED simplex.playlists, scheduling delivery
DEBUG [Cassandra Java Driver worker-1] (ControlConnection.java:242) - [Control connection] Refreshing schema for simplex
DEBUG [New I/O worker #2] (Cluster.java:1433) - Refreshing schema for simplex
DEBUG [Cassandra Java Driver worker-0] (ControlConnection.java:507) - Checking for schema agreement: versions are [d5a20e46-0062-35fd-b148-180af989ae5f]
title                        album                artist
-----------------------------+--------------------+-------------------
La Petite Tonkinoise       Bye Bye Blackbird  Joséphine Baker
DEBUG [main] (Cluster.java:1037) - Shutting down
DEBUG [main] (HostConnectionPool.java:393) - Shutting down pool
```

Drivers for Cassandra

Cassandra has a large number of drivers. The following is a partial list (04/2014)

- *Ruby:*
  - *Cassandra: http://github.com/fauna/cassandra/tree/master*
  - *Cassandra_object*: http://github.com/NZKoz/cassandra_object/tree/master (for Rails)
  - Small Record: http://github.com/astrails/smallrecord/tree/master (for ruby/ActiveModel, Rails)
- Perl:
  - Net-Cassandra: http://search.cpan.org/dist/Net-Cassandra/lib/Net/Cassandra.pm
  - Net-Cassandra-Easy: http://search.cpan.org/dist/Net-Cassandra-Easy/ (A simpler, much less Thrift-oriented interface than Net::Cassandra; includes a CLI called cassidy.pl)
- Python:
  - Telephus: http://github.com/driftx/Telephus/tree/master (Twisted)
  - Pycassa: http://github.com/vomjom/pycassa (version 0.3.0)
  - Tragedy: http://github.com/enki/tragedy/
  - Lazy Boy: http://github.com/digg/lazyboy/tree/master

Drivers for Cassandra

- Scala:
  - Scromium: http://github.com/cliffmoon/scromium
  - Cascal: http://github.com/shorrockin/cascal
  - Cassandra4o: http://code.google.com/p/cassandra4o/ (works with Java, includes hooks for Hibernate-like Object-mapping)
  - Akka: http://akkasource.org/ (Akka includes a Cassandra client but is more than that)
  - Cassie: http://github.com/codahale/cassie
- Java :
  - Hector: http://github.com/rantav/hector
  - Pelops: http://code.google.com/p/pelops/
  - HelenaORM: http://github.com/marcust/HelenaORM (ORM layer built on Hector)
  - OCM: http://github.com/charliem/OCM (higher level client built on Hector)
  - Datanucleus-Cassandra plug-in: http://github.com/PedroGomes/datanucleus-cassandra (Persistence of objects through the JDO/JPA APIs under the Datanucleus platform).
  - Jassandra: http://code.google.com/p/jassandra/
  - Kundera: http://code.google.com/p/kundera/

Drivers for Cassandra

- PHP :
  - PHP Cassandra Client Library: http://github.com/kallaspriit/Cassandra-PHP-Client-Library
  - Pandra: http://github.com/mjpearson/Pandra/tree/master
  - PHP Cassa: http://github.com/hoan/phpcassa [port of pycassa to PHP]
- Clojure :
  - CLJ-Cassandra: http://github.com/robertluo/clj-cassandra
  - Grails : Grails-Cassandra: http://github.com/wolpert/grails-cassandra (Download 0.5.4 from the github site for 0.6 compatibility)
- C++ :
  - LibCassandra: http://github.com/posulliv/libcassandra
- C# / .NET
  - Aquiles: http://aquiles.codeplex.com/
  - Hector Sharp: http://www.hectorsharp.com
  - Fluent Cassandra: http://github.com/managedfusion/fluentcassandra

References

- Moderately complete documentation on CQL language could be found at

http://www.datastax.com/documentation/cql/3.0/cql/aboutCQL.html#./cql/aboutCQL.html

Drivers for Cassandra could be found at:

- JDBC:

http://code.google.com/a/apache-extras.org/p/cassandra-jdbc/

- DATASTAX Java Driver Similar to JDBC

http://www.datastax.com/documentation/developer/java-driver/1.0/common/drivers/introduction/introArchOverview_c.html

- Thrift is the low level driver used by many other drivers

http://wiki.apache.org/cassandra/ThriftExamples

- Hector

http://hector-client.github.io/hector/build/html/index.html

What is MongoDB?

MongoDB (from "humongous") is an

- Open-source document database
- Written in C++
- Agile and scalable.

History of MongoDB

- First developed by 10gen (now MongoDB Inc.) in October 2007 as a component of a planned platform as a service product, the company shifted to an open source development model in 2009, with 10gen offering commercial support and other services.
- Since then, MongoDB has been adopted as backend software by a number of major websites and services, including Craigslist, eBay, Foursquare, SourceForge, and The New York Times, among others. MongoDB is the most popular NoSQL database system.

- BSON is a computer data interchange format used mainly as a data storage and network transfer format in the MongoDB database.
- It is a binary form for representing simple data structures and associative arrays (called objects or documents in MongoDB).
- The name "BSON" is based on the term JSON and stands for "Binary JSON".

Example JSON Schema:

```
{
    "name": "Product",
    "properties": {
        "id": {
            "type": "number",
            "description": "Product identifier",
            "required": true
        },
        "name": {
            "type": "string",
            "description": "Name of the product",
            "required": true
        },
        "price": {
            "type": "number",
            "minimum": 0,
            "required": true
        },
        "tags": {
            "type": "array",
            "items": {
                "type": "string"
            }
        },
        "stock": {
            "type": "object",
            "properties": {
                "warehouse": {
                    "type": "number"
                },
                "retail": {
                    "type": "number"
                }
            }
        }
    }
}
```

Terminology and Concepts

SQL Terms/Concepts	MongoDB Terms/Concepts
database	*database*
table	*collection*
row	*document* or *BSON* document
column	*field*
index	*index*
table joins	embedded documents and linking
primary key Specify any unique column or column combination as primary key.	*primary key* In MongoDB, the primary key is automatically set to the *_id* field.

SQL to MongoDB Mapping

SQL Schema Statements	MongoDB Schema Statements
CREATE TABLE users (id MEDIUMINT NOT NULL AUTO_INCREMENT, user_id Varchar(30), age Number, status char(1), PRIMARY KEY (id))	db.users.insert({ user_id: "abc123", age: 55, status: "A" }) Implicitly created on first insert() operation. The primary key _id is automatically added if _id field is not specified.
SELECT * FROM users	db.users.find()

MongoDB Server-Side JavaScript

- JavaScript may be executed in the MongoDB server processes for various functions, such as query enhancement and map/reduce processing.
- Example:

```
for (var i = 1; i <= 25; i++) db.testData.insert( { x : i } )
```

`db.testData.find()` displays first 20 docs in the collection

```
{ "_id" : ObjectId("51a7dc7b2cacf40b79990be6"), "x" : 1 } { "_id" :
ObjectId("51a7dc7b2cacf40b79990be7"), "x" : 2 } { "_id" :
ObjectId("51a7dc7b2cacf40b79990be8"), "x" : 3 }
```

Data Models

- Data in MongoDB has a *flexible schema*.
- *Collections* do not enforce *document* structure.
- This flexibility gives you data-modeling choices to match your application and its performance requirements.
- In other words: Data Modeling for MongoDB Applications documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold **different types** of data.

Indexes

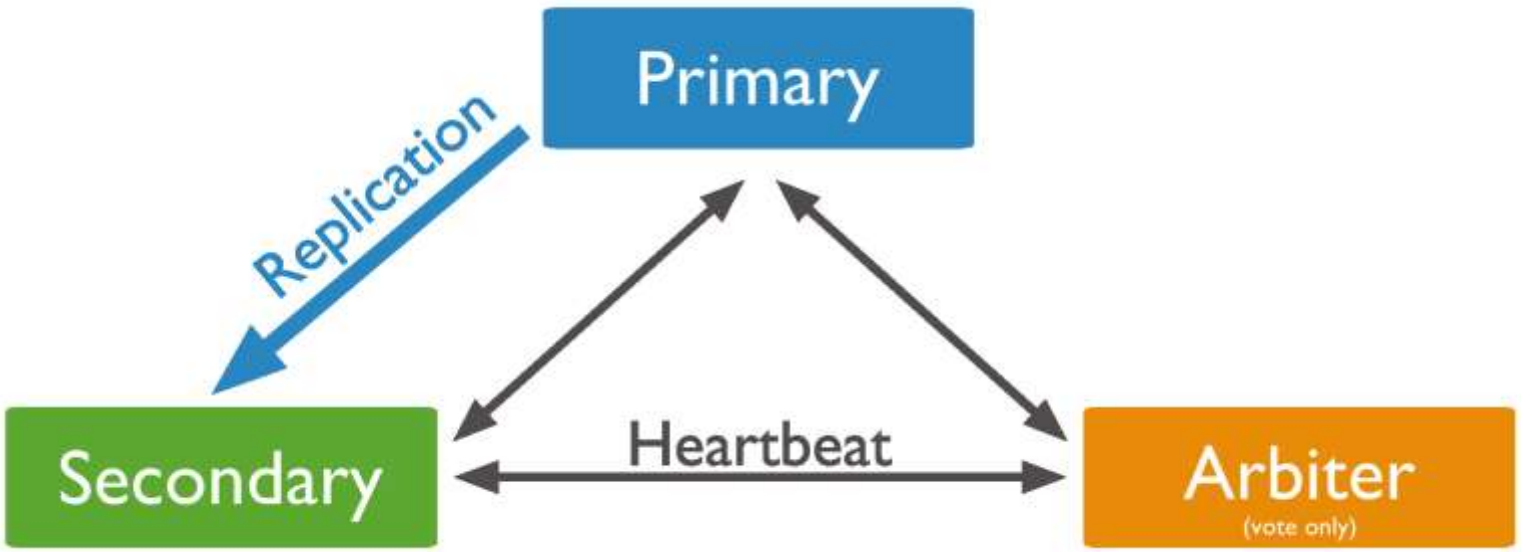
- Indexes provide high performance read operations for frequently used queries.
- Full Index Support - Index on any attribute, just like you're used to.
- Example, **Create an Index on a Single Field:**

```
db.people.ensureIndex( { "phone-number": 1 } )
```

- A value of 1 specifies an index that orders items in ascending order.
- A value of -1 specifies an index that orders items in descending order

Replication

- Replication provides redundancy and increases data availability. Example:



- The primary is the only member in the replica set that receives write operations. A secondary maintains a copy of the primary's data set. Replica sets may have arbiters to add a vote in elections of for primary.

Sharding

- Sharding is the process of storing data records across multiple machines and is MongoDB's approach to meeting the demands of data growth.
- As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput.
- Sharding solves the problem with horizontal scaling.
- With sharding, you add more machines to support data growth and the demands of read and write operations.

Installation

- Release 2.4.8, available for:

  32-bit version has limitations – max 2Gb of data

  Windows Linux Mac OS X Solaris

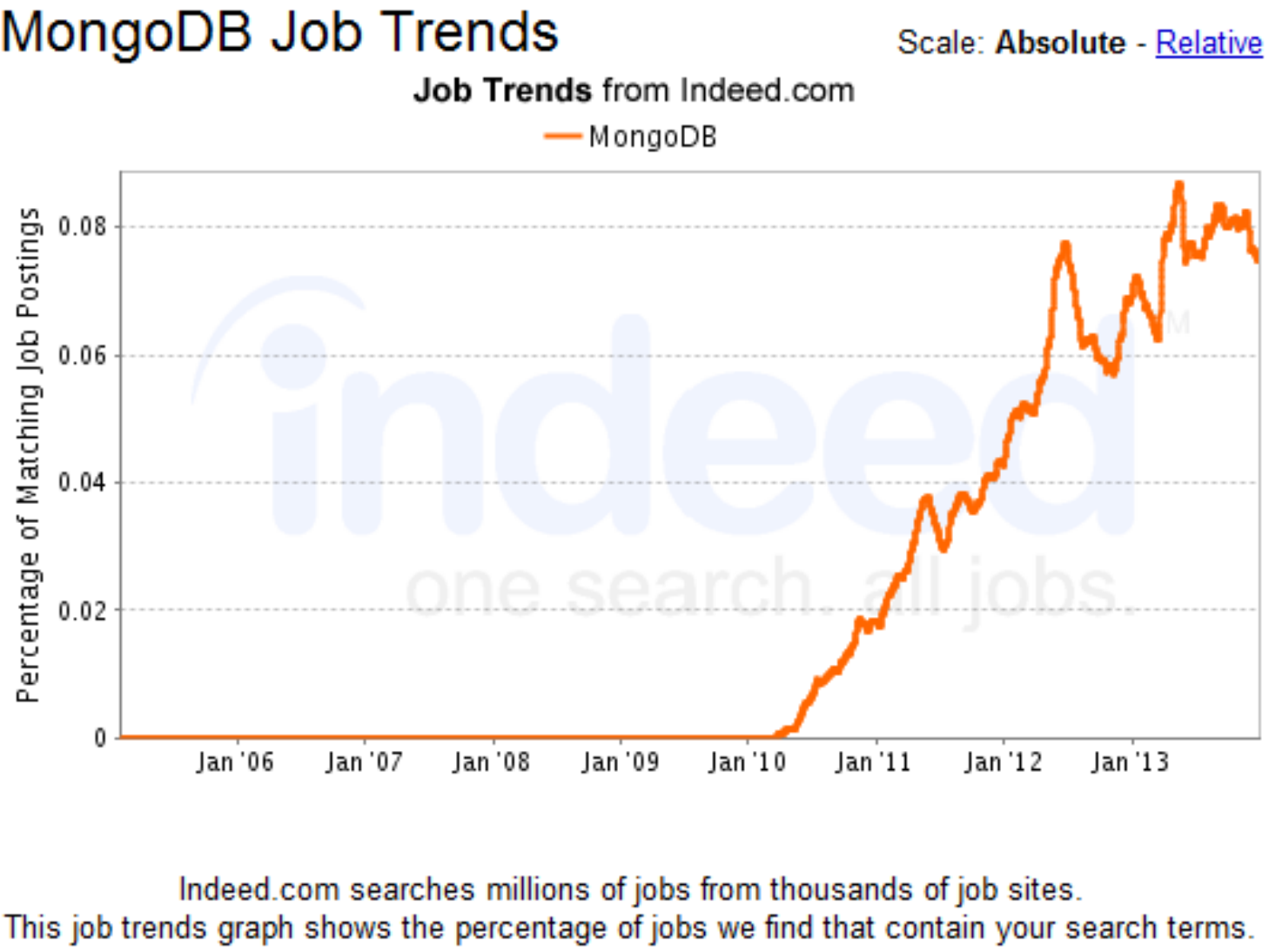
- Drivers for: C, C++, C#, Erlang, Java, Perl, PHP, Python, Ruby, Scala

- Source for MongoDB and mongodb.org supported drivers are open source

Java Driver API

```
MongoClient mongoClient = new MongoClient();
mongoClient.close();
DB db = mongoClient.getDB(dbName);
DBCollection dBCollection = db.getCollection(c);
BasicDBObject doc = new BasicDBObject();
doc.put(string1, string2);
newCollection.insert(doc);
collection.ensureIndex(new BasicDBObject("STR", 1),
indexName);
BasicDBObject query = new BasicDBObject();
query.put("_id", new ObjectId(id));
DBObject dbObj = dbCollection.findOne(query);
```

indeed.com job trends for MongoDB

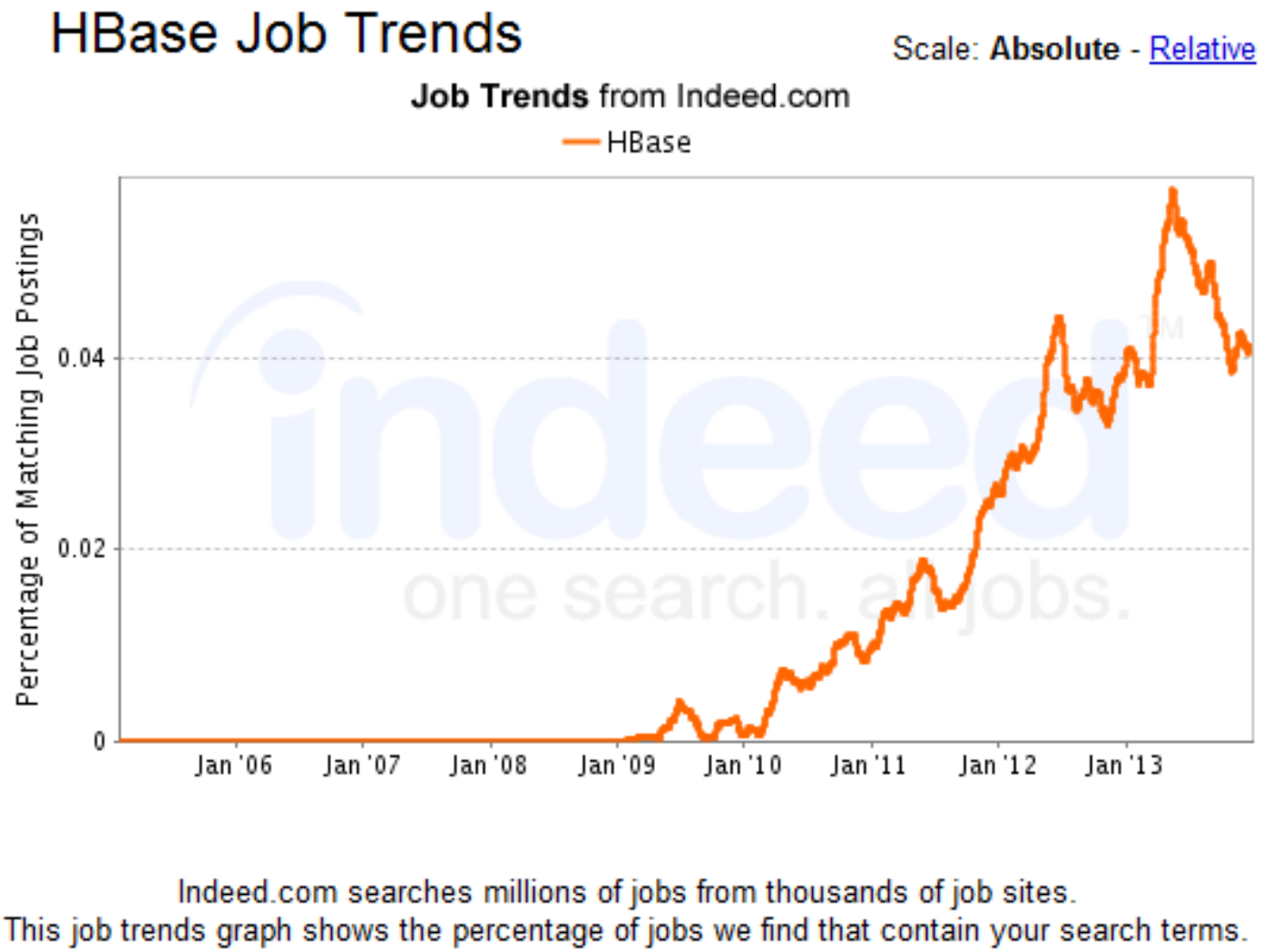


Cassandra Job Trends, indeed.com

- I have impression that number of MongoDB jobs is somewhat higher than Cassandra's

Indeed.com job trend for HBase

- If I read this right, HBase skill's are least sought for



HBase

- HBase is a clone of Google's Bigtable, originally created for use with Hadoop (it's actually a subproject of the Apache Hadoop project). In the way that Google's Bigtable uses the Google File System (GFS), HBase provides database capabilities for Hadoop, allowing you to use it as a source or sink for MapReduce jobs. Unlike some other columnar databases that provide eventual consistency, HBase is strongly consistent.
- Perhaps it is interesting to note that Microsoft is a contributor to HBase, following their acquisition of Powerset.
- **Website**: *http://hbase.apache.org*
- **Orientation**: Columnar
- **Created**: HBase was created at Powerset in 2007 and later donated to Apache.
- **Implementation language**: Java
- **Distributed**: Yes. You can run HBase in standalone, pseudodistributed, or fully distributed mode. Pseudodistributed mode means that you have several instances of HBase, but they're all running on the same host.

HBase

- **Storage**: HBase provides Bigtable-like capabilities on top of the Hadoop File System.
- **Schema**: HBase supports unstructured and partially structured data. To do so, data is organized into column families (a term that appears in discussions of Apache Cassandra). You address an individual record, called a “cell” in HBase, with a combination of row key, column family, cell qualifier, and timestamp. As opposed to RDBMS, in which you must define your table well in advance, with Hbase you can simply name a column family and then allow the cell qualifiers to be determined at runtime. This lets you be very flexible and supports an agile approach to development.
- **Client**: You can interact with HBase via Thrift, a RESTful service gateway, Protobuf (see “Additional Features” below), or an extensable JRuby Shell

Hbase (C+P)

- HBase is strongly consistent. Each row is hosted by a single region server at a time, and a combination of row locks and multi-version concurrency control is used to provide consistency within a row.
- During failover, we take care to not allow writes from a new region server until the previous one has been blocked out.
- Replication is taken care of at the HDFS layer.
- HBase will also be completely durable - no edit will be acknowledged to the client until it has been flushed to three HDFS replicas.

HBase is ..

- HBase is an **open-source**, **distributed**, **column-oriented** database built on top of HDFS based on concept developed by Google BigTable!
- A distributed data store that can scale horizontally to 1,000s of commodity servers and petabytes of indexed storage.
- Designed to operate on top of the Hadoop distributed file system (HDFS) for scalability, fault tolerance, and high availability.

HBase Deficiencies

- Tables have one primary index, the *row key*.
- No join operators.
- Scans and queries can select a subset of available columns, perhaps by using a wildcard.
- There are three types of lookups:
  - Fast lookup using row key and optional timestamp.
  - Full table scan
  - Range scan from region start to end.

HBase Is Not …(2)

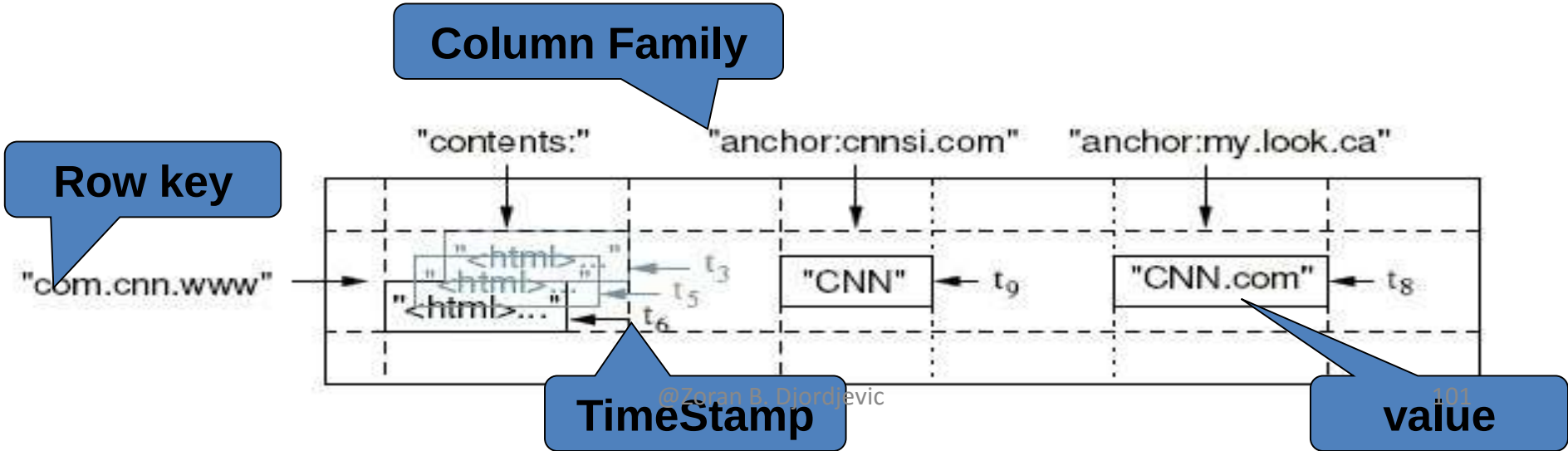
- Limited atomicity and transaction support.
  - HBase supports multiple batched mutations of single rows only.
  - Data is unstructured and untyped.
- Not accessed or manipulated via SQL.
  - Programmatic access via Java, REST, or Thrift APIs.
  - Scripting via JRuby.

Why HBase ?

- HBase is a Bigtable clone.
- It is open source
- It has a good community and promise for the future
- It is developed on top of and has good integration for the Hadoop platform, if you are using Hadoop already.
- *No real indexes*
- *Automatic partitioning*
- *Scale linearly and automatically with new nodes*
- *Commodity hardware*
- *Fault tolerance*
- *Batch processing*

Data Model

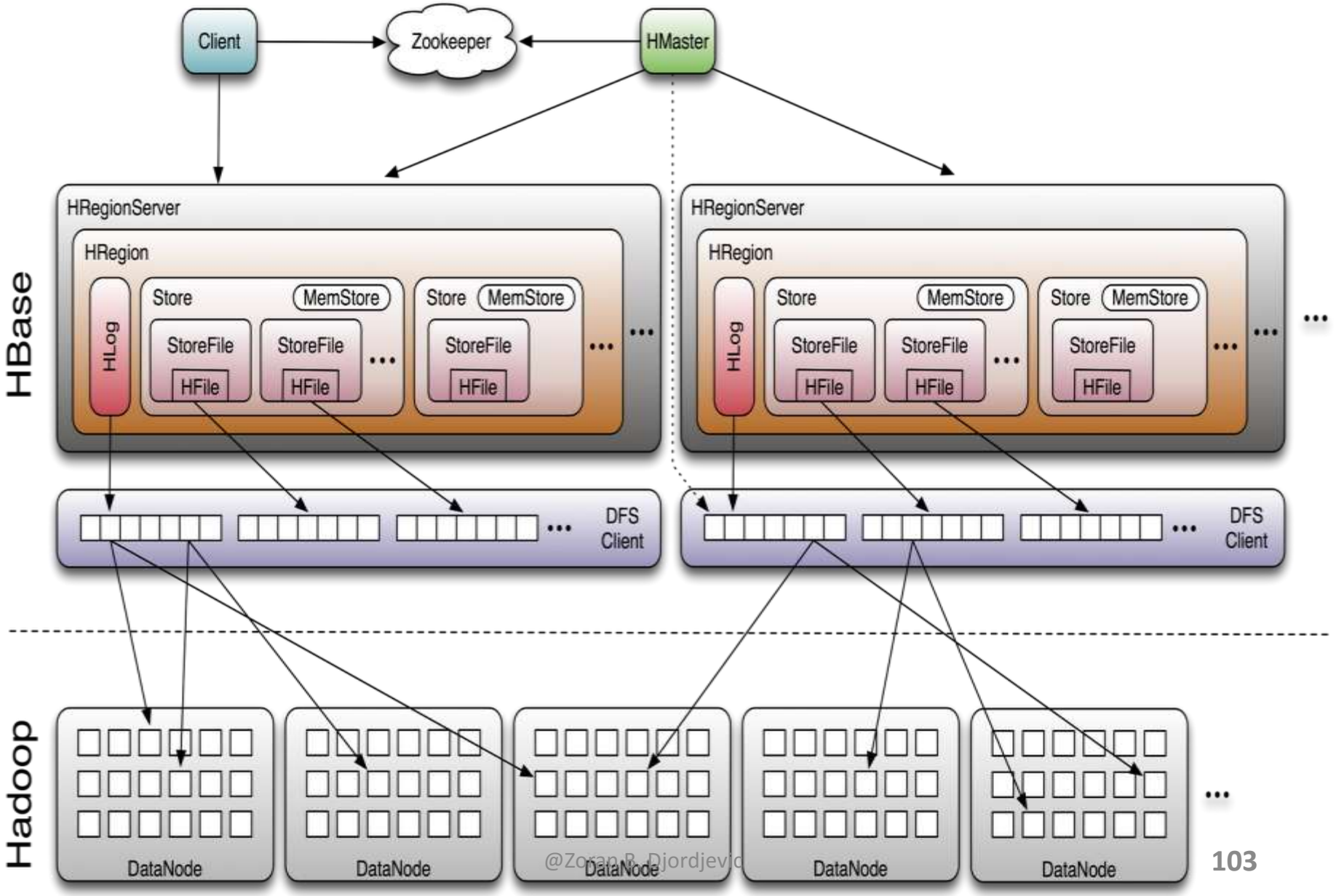
- Tables are sorted by Row
- Table schema only define it's *column families* .
  - Each family consists of any number of columns
  - Each column consists of any number of versions
  - Columns only exist when inserted, NULLs are free.
  - Columns within a family are sorted and stored together
- Everything except table names are byte[]
- (Row, Family: Column, Timestamp) → Value



Hbase Components

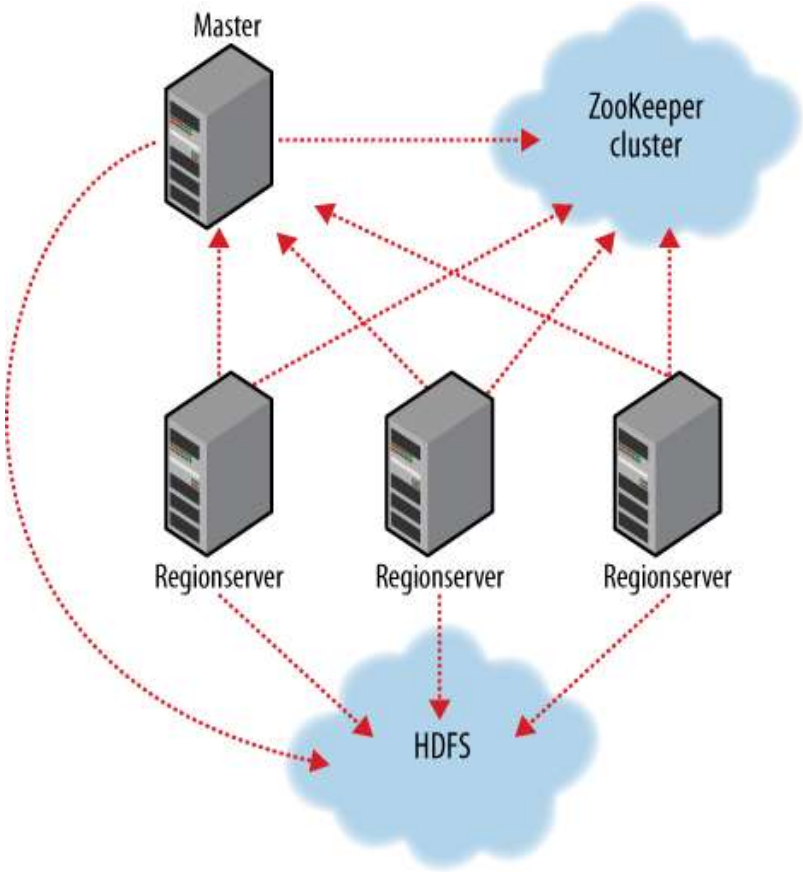
- *Master*
  - Responsible for monitoring region servers
  - Load balancing for regions
  - Redirect client to correct region servers
  - The current SPOF
- *Region Server* slaves
  - Serving requests(Write/Read/Scan) of Client
  - Send HeartBeat to Master
  - Throughput and Region numbers are scalable by region servers

Architecture



ZooKeeper

- Zookeeper is an open source distributed configuration service, providing synchronization and naming registry for large distributed systems.
- HBase depends on ZooKeeper and by default it manages a ZooKeeper instance as the authority on cluster state



Installation (1)

- If you want a distributed system, must have Hadoop in place.
- HBase could work as a standalone system and that is what you want to use for development and initial testing.
- Download the tarball from the Apache HBase website's download section (`http://hbase.apache.org/`):

```
$ mkdir hbase-install
$ cd hbase-install
$ wget http://apache.claz.org/hbase/hbase-0.92.1/hbase-0.92.1.tar.gz
$ tar xvfz hbase-0.92.1.tar.gz
$ export HBASE_HOME=`pwd`/hbase-0.92.1
```

- Once that's done, you can spin up HBase using the provided scripts:

```
$ $HBASE_HOME/bin/start-hbase.sh
starting master, logging to .../hbase-0.92.1/bin/../logs/...-master out
```

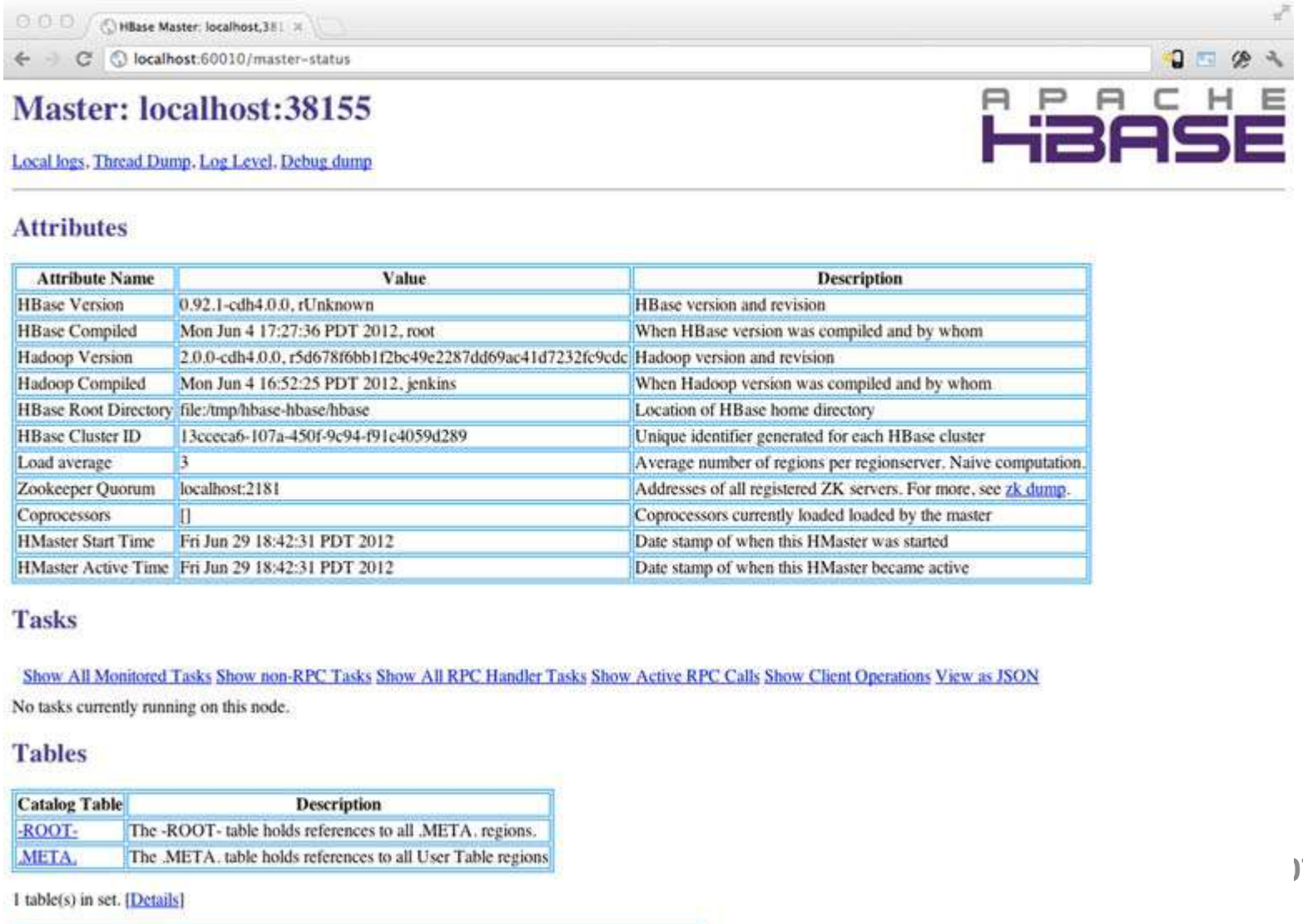
Setup

- If you want, you can also put `$HBASE_HOME/bin` in your PATH so you can simply run `hbase` rather than `$HBASE_HOME/bin/hbase` next time.
- That's all there is to it. You just installed HBase in *standalone* mode.
- The configurations for HBase primarily go into two files: `hbase-env.sh` and `hbase-site.xml`.
- These exist in the `/etc/hbase/conf/` directory.
- By default in standalone mode, HBase writes data into `/tmp`, which isn't the most durable place to write to.
- You can edit the `hbase-site.xml` file and put the following configuration into it to change that location to a directory of your choice:

```
<property>
   <name>hbase.rootdir</name>
   <value>file:///home/user/myhbasedirectory/</value>
</property>
```

Examine HBase, Master Status Page

- Your HBase install has a management console of sorts running on `http://localhost:60010`.



Testing (4)

```
$ hbase shell
> create 'test', 'data'
0 row(s) in 4.3066 seconds
> list
test
1 row(s) in 0.1485 seconds
> put 'test', 'row1', 'data:1', 'value1'
0 row(s) in 0.0454 seconds
> put 'test', 'row2', 'data:2', 'value2'
0 row(s) in 0.0035 seconds
> put 'test', 'row3', 'data:3', 'value3'
0 row(s) in 0.0090 seconds
```

```
> scan 'test'
ROW COLUMN+CELL
row1 column=data:1, timestamp=1240148026198,
        value=value1
row2 column=data:2, timestamp=1240148040035,
        value=value2
row3 column=data:3, timestamp=1240148047497,
        value=value3
3 row(s) in 0.0825 seconds
> disable 'test'
09/04/19 06:40:13 INFO client.HBaseAdmin: Disabled
        test
0 row(s) in 6.0426 seconds
> drop 'test'
09/04/19 06:40:17 INFO client.HBaseAdmin: Deleted
        test
0 row(s) in 0.0210 seconds
> list
0 row(s) in 2.0645 seconds
```

Connecting to HBase

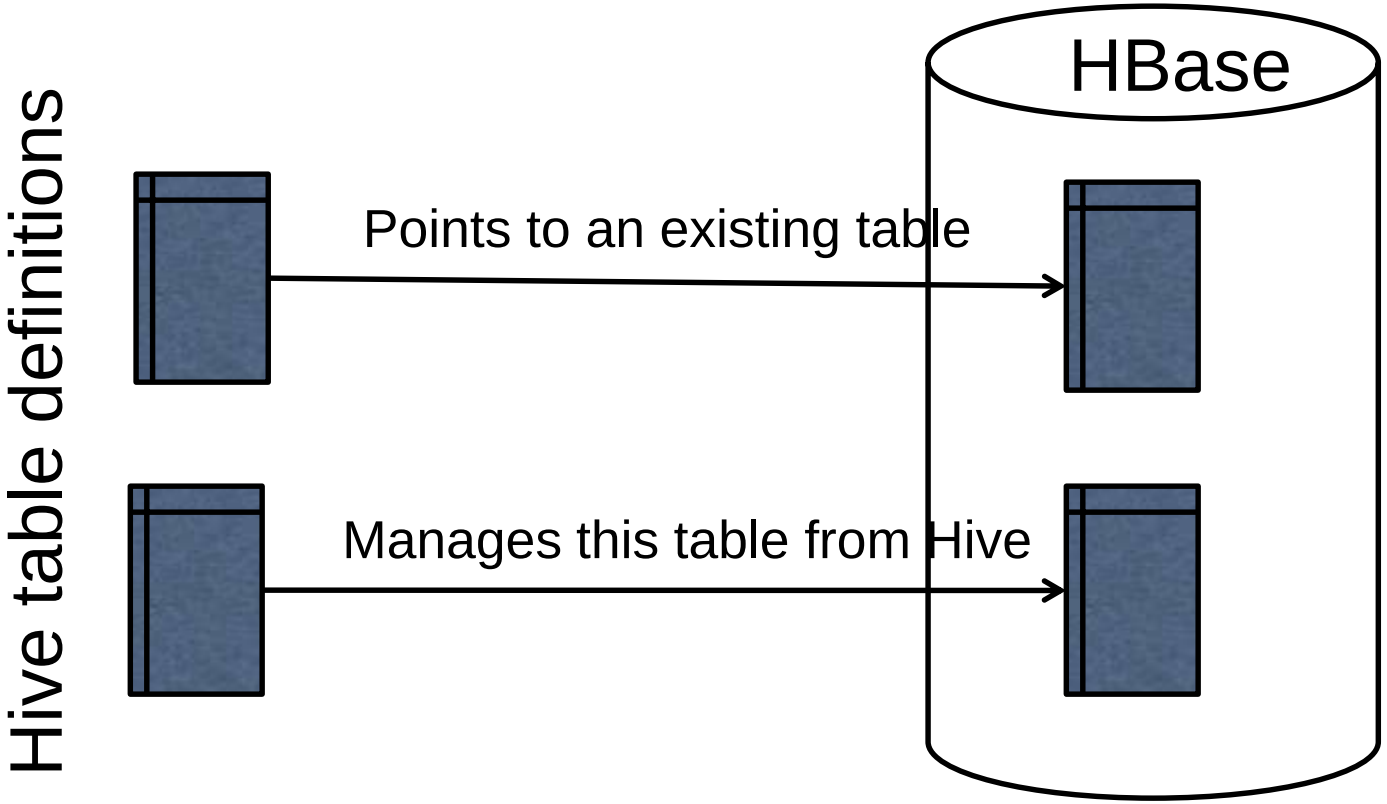
- Java client
  - *get(byte [] row, byte [] column, long timestamp, int versions);*
- Non-Java clients
  - Thrift server hosting HBase client instance
- Sample ruby, c++, & java (via thrift) clients
  - REST server hosts HBase client
- TableInput/OutputFormat for MapReduce
  - HBase as MR source or sink
- HBase Shell
  - JRuby IRB with “DSL” to add get, scan, and admin
  - *./bin/hbase shell YOUR_SCRIPT*

Hive and HBase Could be Integrated

- Reasons to use Hive on HBase:
  - A lot of data sitting in HBase due to its usage in a real-time environment, but never used for analysis
  - Give access to data in HBase usually only queried through MapReduce to people that don’t code (business analysts)
  - When needing a more flexible storage solution, so that rows can be updated live by either a Hive job or an application and can be seen immediately to the other

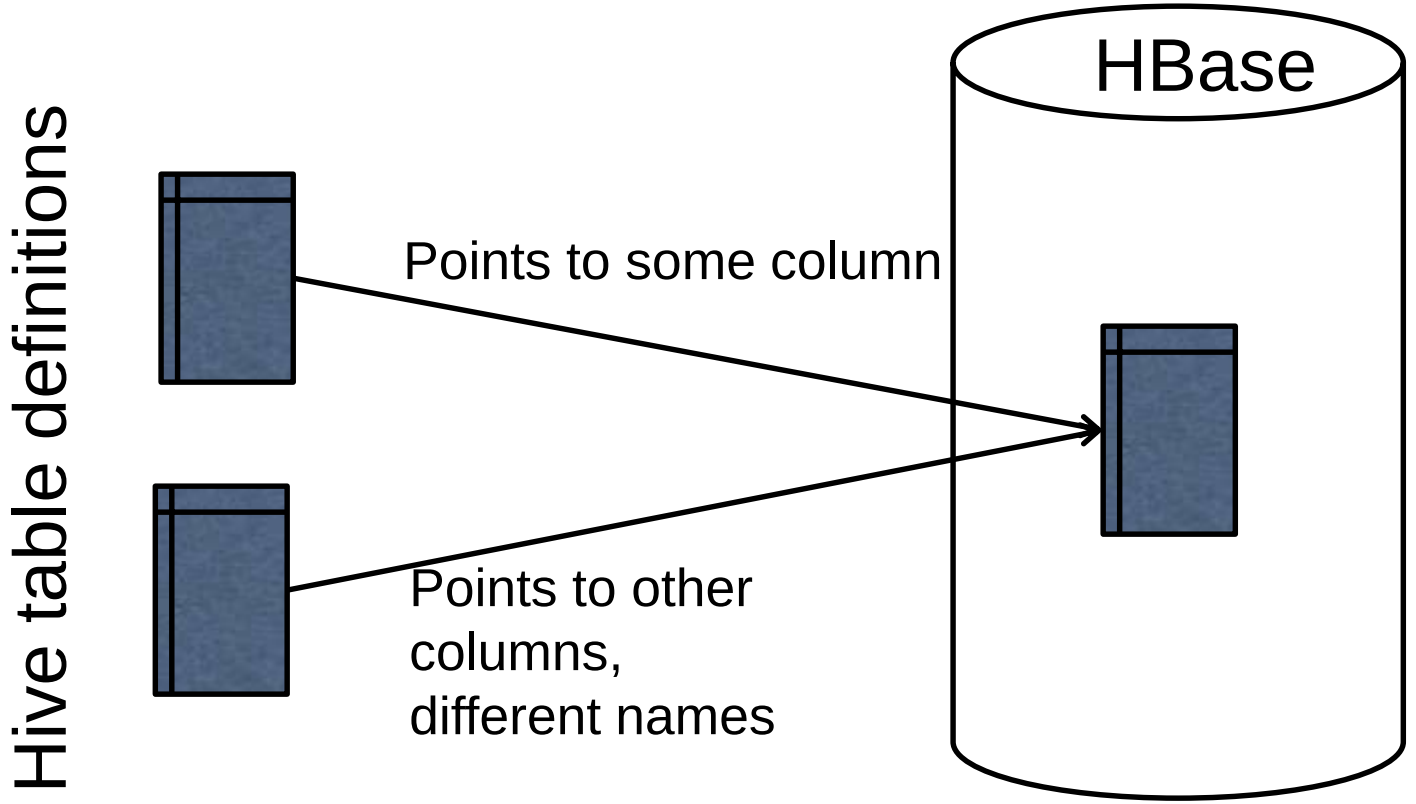
Integration

- Hive can use tables that already exist in HBase or manage its own ones, but they still all reside in the same HBase instance



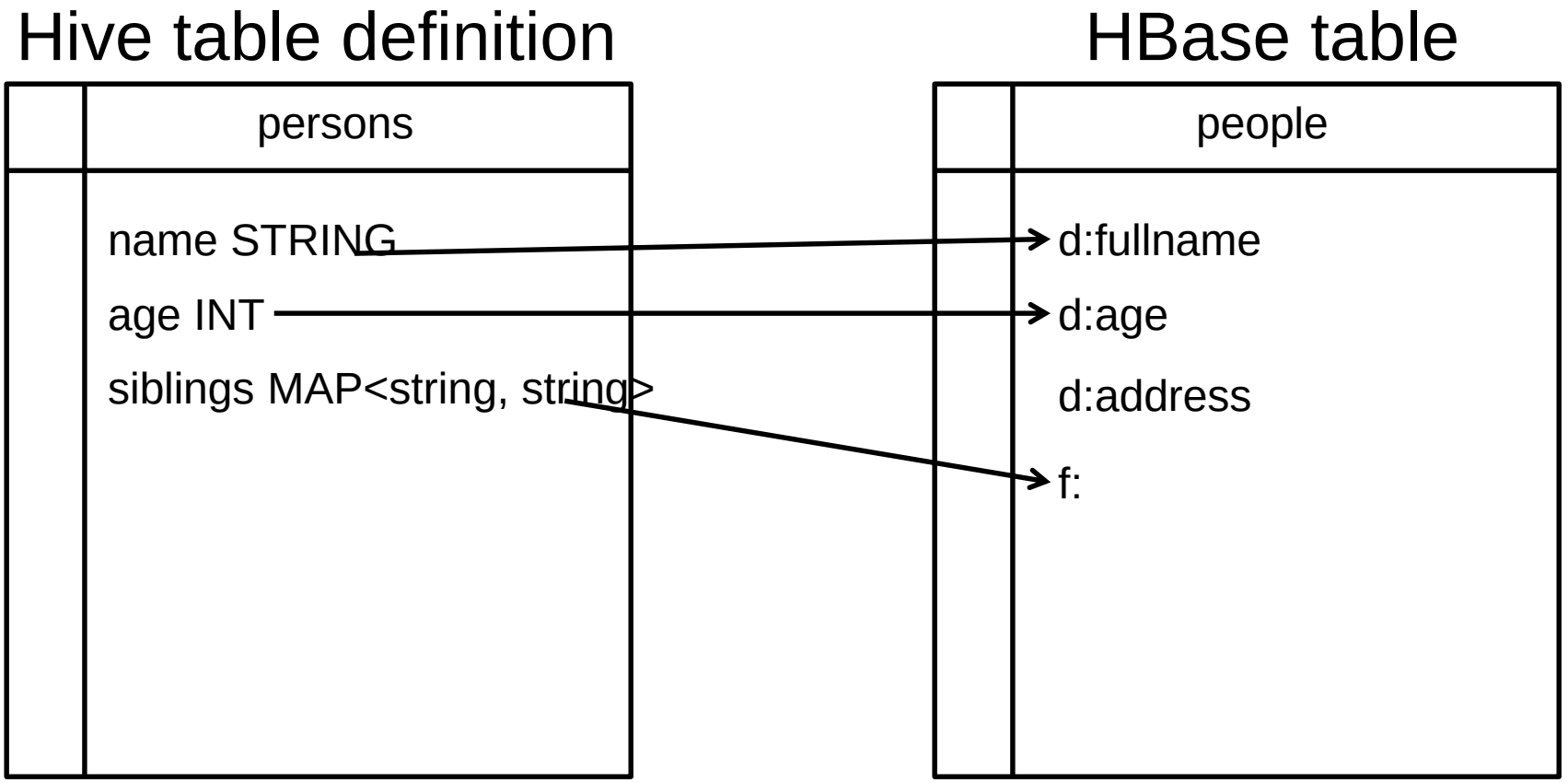
Integration

- When using an already existing table, defined as EXTERNAL, you can create multiple Hive tables that point to it



Integration

- Columns are mapped however you want, changing names and giving types



Using a simple table in HBase:

```
CREATE EXTERNAL TABLE blocked_users(
 userid INT,
 blockee INT,
 blocker INT,
 created BIGINT)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
 ":key,f:blockee,f:blocker,f:created")
TBLPROPERTIES("hbase.table.name" = "m2h_repl-
userdb.stumble.blocked_users");
```

- HBase is a special case here, it has a unique row key map with :key
- Not all the columns in the table need to be mapped

Using a complicated table in HBase:

```
CREATE EXTERNAL TABLE ratings_hbase(
 userid INT,
 created BIGINT,
 urlid INT,
 rating INT,
 topic INT,
 modified BIGINT)
STORED BY
'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
":key#b@0,:key#b@1,:key#b@2,default:rating#b,default:topic#
b,default:modified#b")
TBLPROPERTIES("hbase.table.name" = "ratings_by_userid");
```

#b means binary, @ means position in composite key

HBase Provides Low-latency Random Access

HBase provides Low-latency Random Access

Writes:

- 1-3ms, 1k-10k writes/sec per node

Reads:

- 0-3ms cached, 10-30ms disk
- 10-40k reads / second / node from cache

Cell size:

- 0-3MB preferred
- Read, write and insert data anywhere in the table
- No sequential write limitations

CouchDB

- As a database, CouchDB is perhaps most similar to Lotus Notes. Its creator, Damien Katz, worked on Lotus Notes at IBM before deciding to build a database "*of* the Web." The documents stored by CouchDB do not need to share the same schema, and query capabilities are made available via views, which are constructed as JavaScript functions.
- CouchDB is interesting in part for what it terms Multi-Version Concurrency Control (MVCC). MVCC means that readers will not block writers and writers will not block readers. All writes occur as appends to the document store, making it much harder to corrupt datafiles. This is similar to Cassandra; using an append-only model means that files can grow very large quickly, requiring a background process to run compactions.
- *CouchDB: The Definitive Guide,* by J. Chris Anderson, Jan Lehnardt, Noah Slater.
- **Website**: *http://couchdb.apache.org*
- **Orientation**: Document
- Work was begun in 2005. In 2008, it became an Apache IncubatorHBase is a Bigtable clone.

CouchDB

- **Implementation language**: Erlang
- **Distributed**: Yes. Data can be read and updated by users and the server while disconnected, and any changes can then be replicated bi-directionally later.
- **Schema**: There is no required schema. Documents are stored in their entirety using JSON. Each document is assigned a unique ID.
- **Client**: RESTful JSON API that allows access from any language capable of making HTTP requests.
- **CAP**: Eventually consistent. Replication is used to synchronize multiple copies of data on different nodes. CouchDB features ACID semantics similar to many relational database systems.
- **Production use**: CouchDB is not yet in a 1.0 release as of this writing, but it is used in production in a variety of social websites and software applications. See *http://bit.ly/dn73DY* for a list of specific production instances.
- **Additional features**: MapReduce, incremental replication, and fault-tolerance are all supported. Comes with a web console.

RIAK

- Riak is a hybrid database based on Amazon Dynamo that acts as a document-oriented database and also a distributed key-value store. It's fault-tolerant and scales linearly, and it's intended for use in web applications. It is similar to Cassandra in that it does not have a central controller, and therefore no single point of failure.
- The design of Riak includes three basic elements: buckets, keys, and values. Data is organized into buckets, which are little more than flat namespaces for logically grouping key-value pairs. This much is similar in design and terminology to the Google Storage system.
- Basho Technologies, the maker of Riak, offers both a commercial version and an open source version.
- Riak runs on most Unix-based systems, but is not supported on Windows.
- **Website**: *http://wiki.basho.com*
- **Orientation**: Document and key-value store
- **Created**: Basho Technologies in Cambridge, Massachusetts. This company was formed in 2008 by architects from Akamai.

RIAK

- **Implementation language**: Primarily Erlang, with some C and JavaScript
- **Distributed**: Yes
- **Replication**: Replication can be set at the bucket level.
- **Schema**: Riak is schema-less and doesn't use specific data types. The values associated with keys are objects. All data is stored as opaque BLOBs, so you can store just about any kind of data in Riak.
- **Client**: Riak offers three primary ways of interacting with it: via a JSON over HTTP interface; drivers for Erlang, Python, Java, PHP, JavaScript, and Ruby; and, finally, a Protocol Buffers client interface. Protocol Buffers is a Google project that they use internally for very fast RPC, and is available at *http://code.google.com/p/ protobuf/*.
- **CAP**: Riak is similar to Cassandra in that the database allows for "tuneability" for desired levels of consistency, availability, and partition tolerance.

MongDB

- MongoDB is perhaps most similar to CouchDB. It purports to combine the best of keyvalue stores, document databases, object databases, and RDBMS. That is, it shards automatically as with a key-value store, allows JSON-based dynamic schema documents, and offers a rich query language in the manner of a relational database.
- *MongoDB: The Definitive Guide,* by Kristina Chodorow & Michael Dirolf.
- **Website**: *http://www.mongodb.org*
- **Orientation**: Document
- **Created**: Developed at 10gen by Geir Magnusson and Dwight Merriman
- **Implementation language**: C++
- **Distributed**: Yes
- **Schema**: JSON-style documents are stored, and you can use dynamic schemas.
- **CAP**: MongoDB uses a single master for any shard, making it completely
- consistent.
- **Production use**: MongoDB is used in production at SourceForge, Bit.ly, Foursquare, GitHub, Shutterfly, Evite, The New York Times, Etsy, and more

FlockDB

- In April 2010, Twitter announced that they were open-sourcing to GitHub their new graph database called FlockDB. They created FlockDB to store the adjacency lists for followers on Twitter, so they could readily understand who follows whom and who blocks whom. It scales horizontally and is designed for online, low-latency, high throughput environments. The Twitter FlockDB cluster stores 13+ billion edges and sustains peak traffic of 20,000 writes per second and 100,000 reads per second.
- **Website**: *http://github.com/twitter/flockdb*
- **Orientation**: Graph
- **Created**: Created in 2010 by Twitter
- **Implementation language**: Scala
- **License**: Apache License v2
- **Distributed**: Yes

FlockDB

- **Schema**: The schema is very straightforward, as FlockDB does not attempt to solve every database problem, but only those relating to the set of problems Twitter faces with their relationship graphs and the size of their dataset. The graph contains entries with four attributes: a source ID, a destination ID, a position, and a state.
- **Client**: FlockDB uses the Thrift 0.2 client, and Twitter has also written a Ruby frontend that offers a richer interface.
- **Replication**: Yes
- **Storage**: MySQL
- **Production use**: Twitter
- **Additional features**: FlockDB allows you to quickly page through result sets that contain millions of entries and to archive and later restore previously archived graph edges. It uses Kestrel as a loosely coupled, reliable message queue that picks a server at random to write to, so there is no cross-server communication (no clustering, no multicast, etc.

Cassandra on Debian Linux

Zoran B. Djordjević

csci e63 Big Data Analytics

Debian Linux

- Cassandra works with Debian Linux. Ubunutu is a derivative of Debian Linux. It appears that Mac OS is also. One prefers to work with original.
- The best illustration of the trash we are dealing with here is the fact that neither on the site `cassandra.apache.org`, nor on the site `datastax.com`, run by people who are commercializing this technology, one could find any mention of the version of various Linux OS-s or Cassandra's packages which are compatible one to another. This is bad even by the very low standards of Hadoop community. Cassandra creators
- think that if you have to ask, you do not deserve to be told.
- There is a site called `http://www.debian.org/`, where one could download CD ISO Images. The latest stable Debian Linux is 7.4. I do not know whather that version is a good choice.
- On the page `http://www.debian.org/distrib/netinst` I found links that appear to make it possible to download proper ISO image. I chose **`amd64`** architecture, hoping it would give me 64-bit Debian Linux.
- Creation of VM is relatively uneventful. Answer Yes whenever asked. Install desktop environment when asked.

Adjust User rights, Display

- During the installation process you will be asked to set the password for user root. Record the password. You will be also asked to create another user. You can create `cassandra` with password `cassandra`.
- Before you start working with your new VM, change display settings.
- Select

`Applications > System Tools > Preferences> System Settings > Displays`

- On the resulting screen select resolution to your liking, e.g. `1280X1024(5:4)`
- Make the other user (`cassandra`) a sudo user. Just like on CentOS, add user `cassandra` to `/etc/sudoers` file

Install Java 8 on Ubuntu

- Check which version of the JDK your system is using:

```
$ java -version
java version "1.8.0_65"
```

- If you have older Java, go to Oracle Java SE Downloads, accept the license agreement, and download the installer for your distribution.
- Make a directory for the JDK:

```
$ sudo mkdir -p /usr/lib/jvm
```

- Unpack the tarball and install the JDK:

```
$ sudo tar zxvf jdk-8u65-linux-x64.tar.gz -C /usr/lib/jvm
```

- The JDK files are in directory called `/usr/lib/jvm/jdk-8u_version.`
- Tell the system that there's a new Java version available:

```
$ sudo update-alternatives --install "/usr/bin/java" "java"
"/usr/lib/jvm/jdk1.8.0_version/bin/java" 1
```

- If updating from a previous version you many need to execute the above command twice. Set the new JDK as the default using command:

```
$ sudo update-alternatives --config java
```

- Make sure your system is using the correct JDK:
- `$ java -version`
- `java version "1.8.0_65"`

Download and Install Java 7

- Under `Applications > Internet` select `IceWeasel Web Browser`
- Before you can start anything go to `oracle.com` and download `jdk-7u51-linux-x64.tar.gz`
- Untar `jdk-..` archive into the directory `/usr/local`
- In `.bash_profile` file in `/home/cassandra` directory, set `JAVA_HOME` environmental variable to

```
JAVA_HOME=/usr/local/jdk1.7.0_51
export JAVA_HOME
PATH=$PATH:$JAVA_HOME/bin
export PATH
```

- As user `root`, remove existing `/usr/bin/java` and create new symbolic link to `$JAVA_HOME/bin/java`

```
$ ln –s /usr/local/jdk1.7.0_51/bin/java /usr/bin/java
$ java –version
Java version “1.7.0_51”
```

Download and Start `cassandra` server

- Under `Applications > Internet` select `IceWeasel Web Browser`
- On the site `cassandra.apache.org` select download options for release `2.1.13` and download `apache-cassandra-2.1.13-bin.tar.gz`
- Copy `apache-cassandra-2.1.13-bin.tar.gz file` to `/usr/local`

```
$ sudo apache-cassandra-2.1.13-bin.tar.gz file /usr/local
```

- On the command prompt in directory `/usr/local` type:

```
$ sudo tar zxvf apache-cassandra-2.1.13-bin.tar.gz
$ sudo rm *.tar.gz
```

- To start `cassandra` server in the directory `/user/local/apache-cassandra-2.1.13/bin` type:

```
$ sudo cassandra –f
```

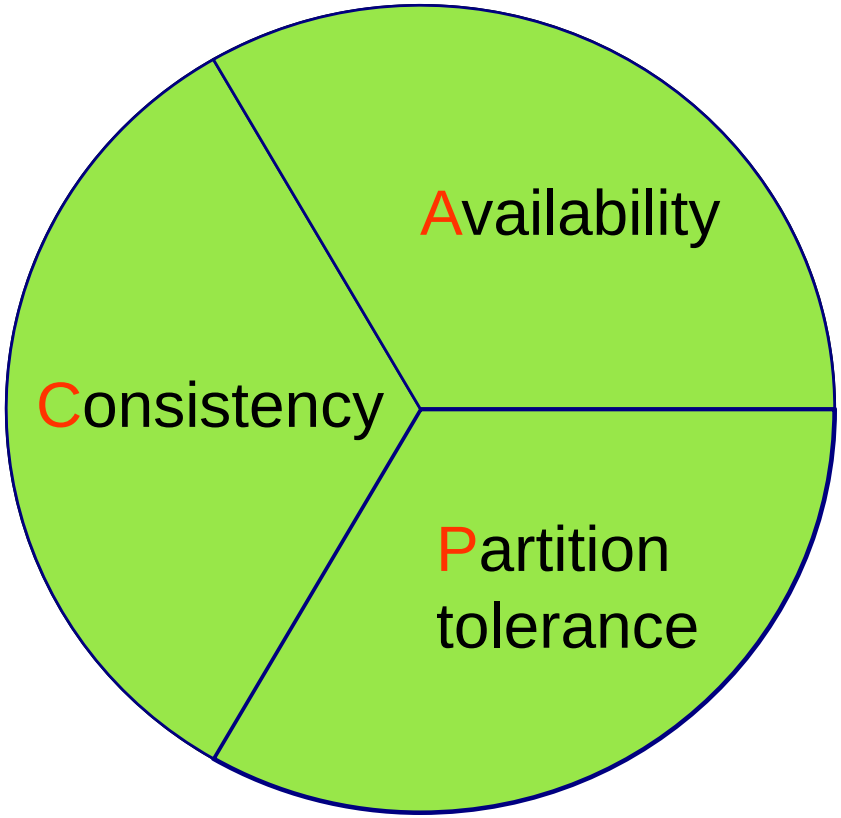
- Using the -f switch tells Cassandra to stay in the foreground instead of running as a background process, so that all of the server logs will print to standard out and you can see them in your terminal window, which is useful for testing. To stop `cassandra` type `Ctrl C` in the console.
- Under normal circumstances, running server will write the following output to the console:

Console output of Normally Running Server

```
$ sudo ./cassandra -f
INFO 23:50:54,079 Logging initialized
INFO 23:50:54,156 Loading settings from file:/usr/local/apache-cassandra-
2.1.13/conf/cassandra.yaml
INFO 23:50:54,901 Data files directories: [/var/lib/cassandra/data]
INFO 23:50:54,902 Commit log directory: /var/lib/cassandra/commitlog
INFO 23:50:54,902 DiskAccessMode 'auto' determined to be mmap, indexAccessMode
INFO 23:50:54,903 disk_failure_policy is stop INFO 23:50:54,903
commit_failure_policy is stop
INFO 23:50:54,914 Global memtable threshold is enabled at 251MB INFO
23:50:55,161 Not using multi-threaded compaction
INFO 23:50:55,612 JVM vendor/version: Java HotSpot(TM) 64-Bit Server INFO
23:50:55,612 Heap size: 1052770304/1052770304
INFO 23:50:55,613 Code Cache Non-heap memory:
INFO 23:50:55,612 Heap size: 1052770304/1052770304
INFO 23:50:55,613 Code Cache Non-heap memory: init = 2555904(2496K) used =
650560(635K) committed = 2555904(2496K) max = 50331648(49152K) INFO
23:50:55,613 Par Eden Space Heap memory: init = 167772160(163840K) used =
57089352(55751K) committed = 167772160(163840K) max = 167772160(163840K)
INFO 23:50:55,614 Par Survivor Space Heap memory: init = 20971520(20480K) used
= 0(0K) committed = 20971520(20480K) max = 20971520(20480K)
INFO 23:50:55,614 CMS Old Gen Heap memory: init = 864026624(843776K) used =
0(0K) committed = 864026624(843776K) max = 864026624(843776K) .
. . . . .
```

CAP Theorem

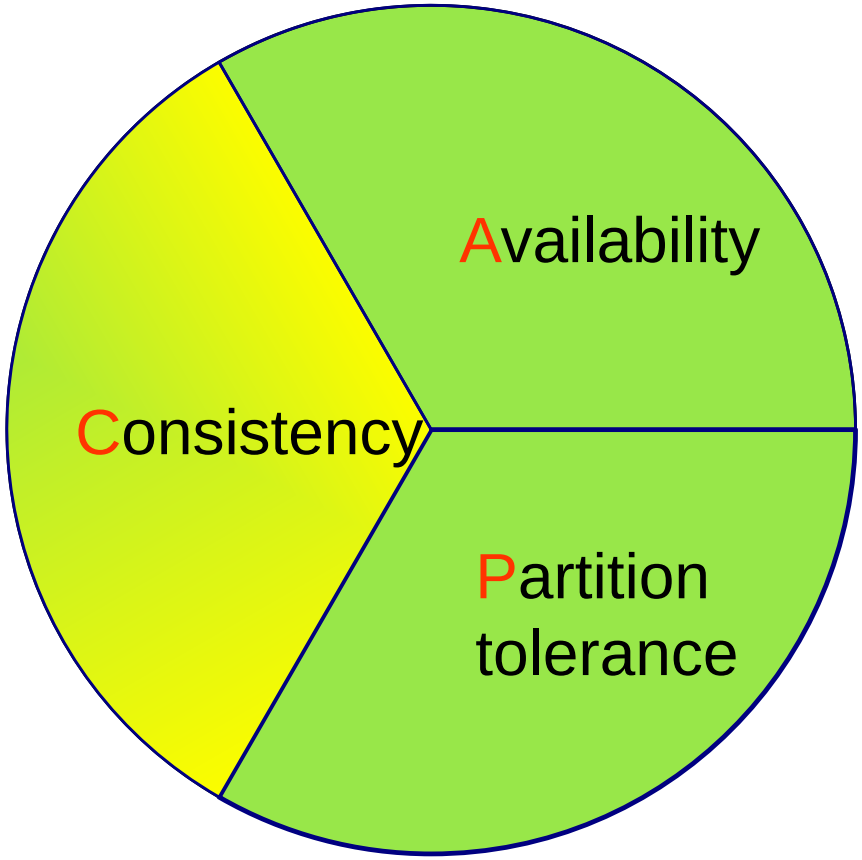
The CAP Theorem



CAP Theorem

- Proposed by Eric Brewer (talk on Principles of Distributed Computing July 2000).
- Three properties of a system: **C**onsistency, **A**vailability and **P**artitionability.
- **You can have at most two of these three properties for any shared-data system**
  - Partitionability: Can divide nodes into small groups that can see other groups, but they can't see everyone.
  - Consistency: write a value and then you read the value you get the same value back. In a partitioned system there are windows where that's not true.
  - Availability: may not always be able to write or read. The system will say you can't write because it wants to keep the system consistent.
- To scale you have to partition, so you are left with choosing either high consistency or high availability for a system. Find the right overlap of availability and consistency. Choose an approach based on the service
- For the checkout process you always want to honor requests to add items to a shopping cart because it's revenue producing. In this case you choose high availability. Errors are hidden from the customer and sorted out later.
- When a customer submits an order you favor consistency because several services--credit card processing, shipping and handling, reporting— are simultaneously accessing the data.

The CAP Theorem



Once a writer has written, all readers will see that write.

- Two kinds of consistency:
  - strong consistency – ACID(Atomicity Consistency Isolation Durability)
  - weak consistency – BASE(Basically Available Soft-state Eventual consistency)

ACID Transactions

- A DBMS is expected to support "*ACID transactions*," processes that are:
  - *Atomic* : Either the whole process is done or none is.
  - *Consistent* : Database constraints are preserved.
  - *Isolated* : It appears to the user as if only one process executes at a time.
  - *Durable* : Effects of a process do not get lost if the system crashes.

Atomicity

- A real-world event either happens or does not happen
  - Student either registers or does not register
- Similarly, the system must ensure that either the corresponding transaction runs to completion or, if not, it has no effect at all
  - Not true of ordinary programs. A crash could leave files partially updated on recovery

Commit and Abort

- If the transaction successfully completes it is said to commit
  - The system is responsible for ensuring that all changes to the database have been saved
- If the transaction does not successfully complete, it is said to abort
  - The system is responsible for undoing, or rolling back, all changes the transaction has made

Database Consistency

- Enterprise (Business) Rules limit the occurrence of certain real-world events
  - Student cannot register for a course if the current number of registrants equals the maximum allowed
- Correspondingly, allowable database states are restricted

  cur_reg <= *max_reg*
- These limitations are called (static) integrity constraints*:* assertions that must be satisfied by all database states (state invariants).

Database Consistency (state invariants)

- Other static consistency requirements are related to the fact that the database might store the same information in different ways
  - *cur_reg* = |*list_of_registered_students*|
  - Such limitations are also expressed as integrity constraints
- Database is consistent if all static integrity constraints are satisfied

Transaction Consistency

- A consistent database state does not necessarily model the actual state of the enterprise
  - A deposit transaction that increments the balance by the wrong amount maintains the integrity constraint *balance* ≥ 0, but does not maintain the relation between the enterprise and database states
- A consistent transaction maintains database consistency and the correspondence between the database state and the enterprise state (implements its specification)
  - Specification of deposit transaction includes

    $balance' = balance + amt\_deposit$,

    (*balance′* is the next value of *balance*)

Dynamic Integrity Constraints
(transition invariants)

- Some constraints restrict allowable state transitions
  - A transaction might transform the database from one consistent state to another, but the transition might not be permissible
  - Example: A letter grade in a course (A, B, C, D, F) cannot be changed to an incomplete (I)
- Dynamic constraints cannot be checked by examining the database state

Transaction Consistency

- Consistent transaction: if DB is in consistent state initially, when the transaction completes:
  - All static integrity constraints are satisfied (but constraints might be violated in intermediate states)
    - Can be checked by examining snapshot of database
  - New state satisfies specifications of transaction
    - Cannot be checked from database snapshot
  - No dynamic constraints have been violated
    - Cannot be checked from database snapshot

Isolation

- Serial Execution: transactions execute in sequence
  - Each one starts after the previous one completes.
    - Execution of one transaction is not affected by the operations of another since they do not overlap in time
  - The execution of each transaction is isolated from all others.
- If the initial database state and all transactions are consistent, then the final database state will be consistent and will accurately reflect the real-world state, *but*
- Serial execution is inadequate from a performance perspective

Isolation

- Concurrent execution offers performance benefits:
  - A computer system has multiple resources capable of executing independently (*e.g.,* cpu's, I/O devices), *but*
  - A transaction typically uses only one resource at a time
  - Hence, only concurrently executing transactions can make effective use of the system
  - Concurrently executing transactions yield interleaved schedules

Durability

- The system must ensure that once a transaction commits, its effect on the database state is not lost in spite of subsequent failures
  - Not true of ordinary programs. A media failure after a program successfully terminates could cause the file system to be restored to a state that preceded the program's execution

Implementing Durability

- Database stored redundantly on mass storage devices to protect against media failure
- Architecture of mass storage devices affects type of media failures that can be tolerated
- Related to Availability: extent to which a (possibly distributed) system can provide service despite failure
  - Non-stop DBMS (mirrored disks)
  - Recovery based DBMS (log)

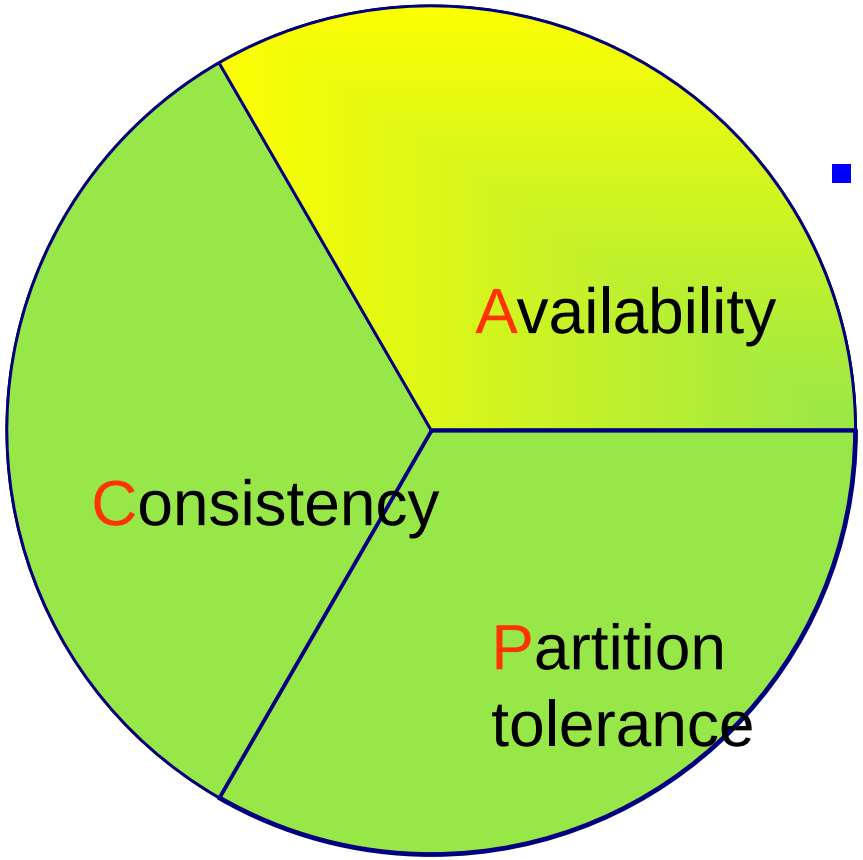
Consistency Model

- A consistency model determines rules for visibility and apparent order of updates.
- For example:
  - Row X is replicated on nodes M and N
  - Client A writes row X to node N
  - Some period of time t elapses.
  - Client B reads row X from node M
  - Does client B see the write from client A?
  - Consistency is a continuum with tradeoffs
  - For NoSQL, the answer would be: maybe
  - CAP Theorem states: Strict Consistency can't be achieved at the same time as availability and partition-tolerance.

Eventual Consistency

- When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent
- For a given accepted update and a given node, eventually either the update reaches the node or the node is removed from service
- The types of large systems based on CAP aren't **ACID** they are **BASE** : **B**asically **A**vailable, **S**oft state, **E**ventual consistency.
  - Basically Available - system seems to work all the time
  - Soft State - it doesn't have to be consistent all the time
  - Eventually Consistent - becomes consistent at some later time

The CAP Theorem

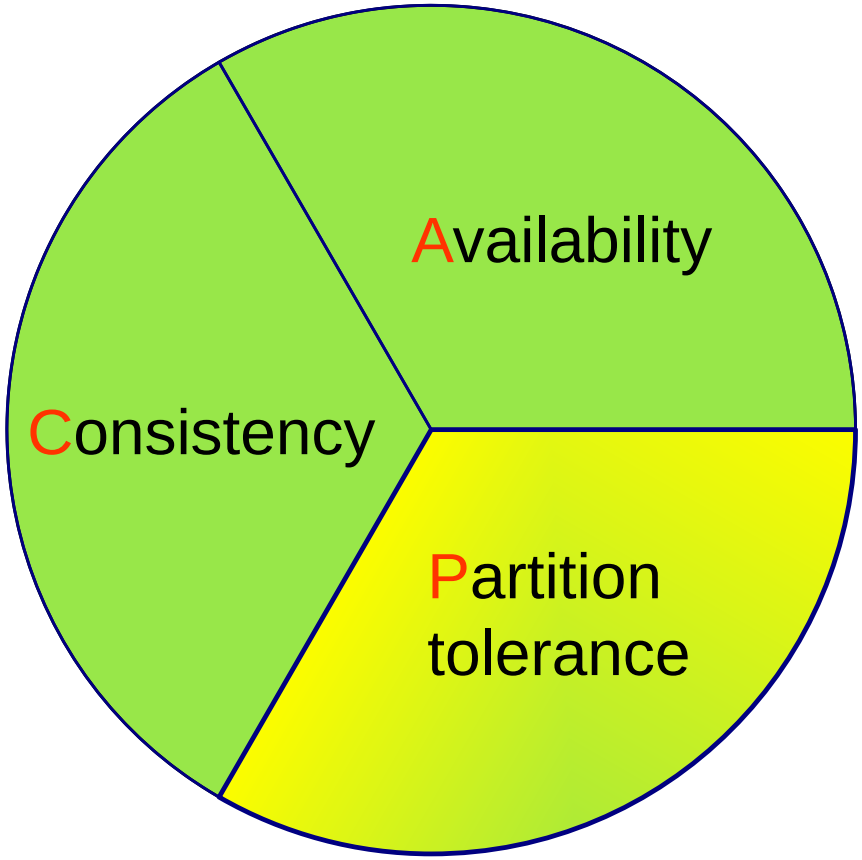


System is available during software and hardware upgrades and node failures.

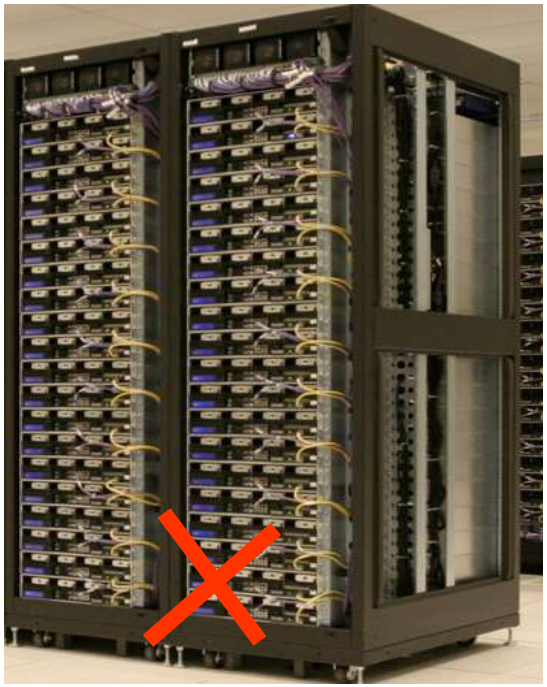
- Traditionally, thought of as the server/process available five 9's (99.999 %).

  However, for large node system, at almost any point in time there's a good chance that a node is either down or there is a network disruption among the nodes.
  - Want a system that is resilient in the face of network disruption

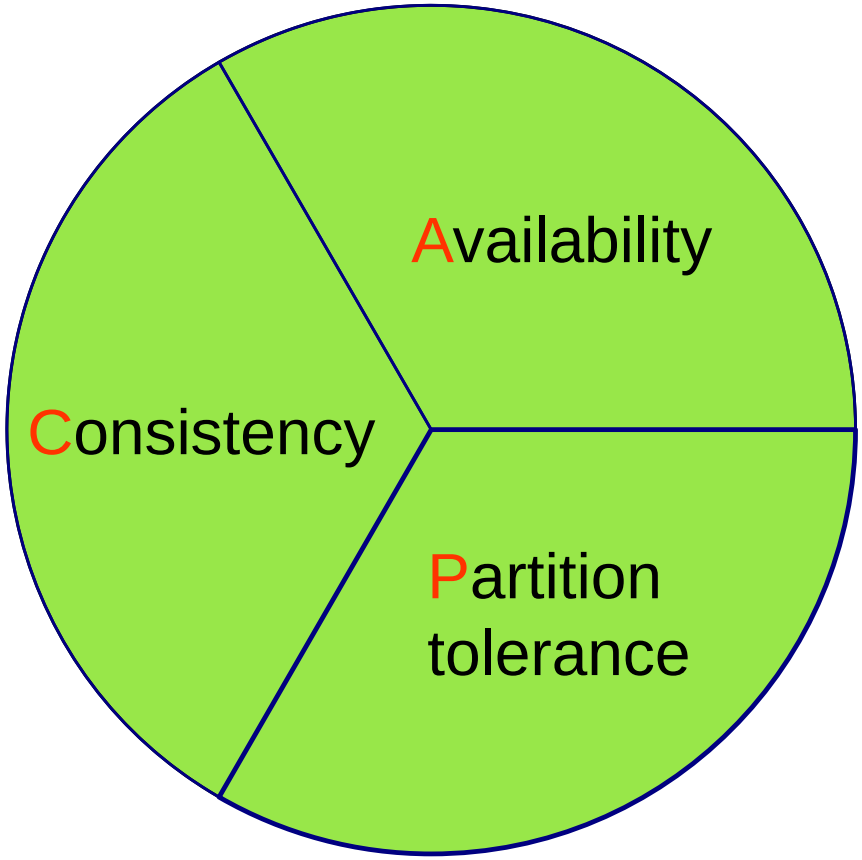
The CAP Theorem



A system can continue to operate in the presence of a network partitions.



The CAP Theorem



Theorem: You can have at most **two** of these properties for any shared-data system