**Big Data**

**WHAT IS BIG DATA?**

**1. What are the definitions of Big Data (5 Vs)**

**Volume** – Size –

* How Big is the data, Terabytes ? Petabytes?

**Velocity** – Speed

* How fast is it coming in ? Can we store it fast enough and then use it ?

**Variety** - Difference, Changeable

* Is it the same sort of data, what about blobs ? Does it change ?
* **Value –** If something meaningful can be extracted
* **Viability**

**2 Not Atomic –** Velocity and volume is the essence. Data is stored quickly with the hopes that algorithms. .

**3. Governmental**

* Multiple sources of data and bringing them together
* E.g. social media data, NHC, Police, Fire Brigade
* Bringing disparate data sets together and gleaning meaningful data from them.

**Defination of Big Data**

A collection of [data sets](http://en.wikipedia.org/wiki/Data_set) so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications.

**Tables Vs Big Data**

**Business Data** is often tabular and atomic

* But may not be relational
* Stored in tables, but not always normalised.

**Normalisation**

* **1NF** – Primary Key
* **2nf** – Related to whole key
* **3nf** – Moved if dependant on another field
* **4nf** – removes unwanted data structures
* **5NF**- every joining dependency is implied by candidate keys

**What is Atomic Data? E.g. is Date Atomic**

**Real world examples -** Pictures, Music, and Machine Data

Tablular

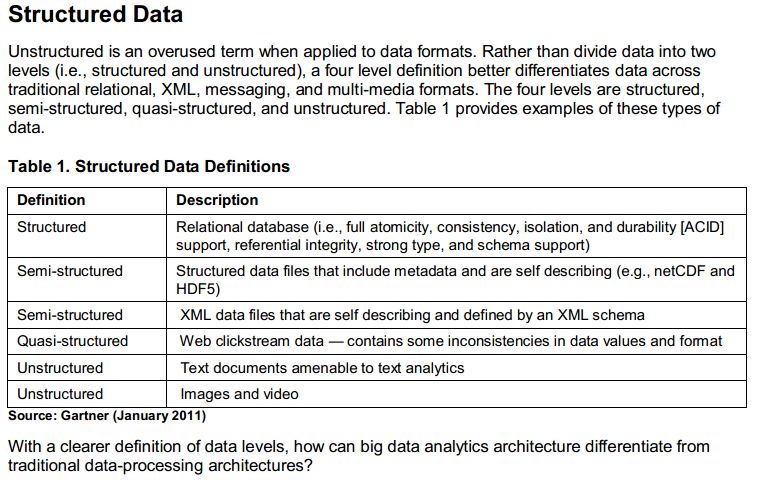
Programmer View

Impedance mismatch ?

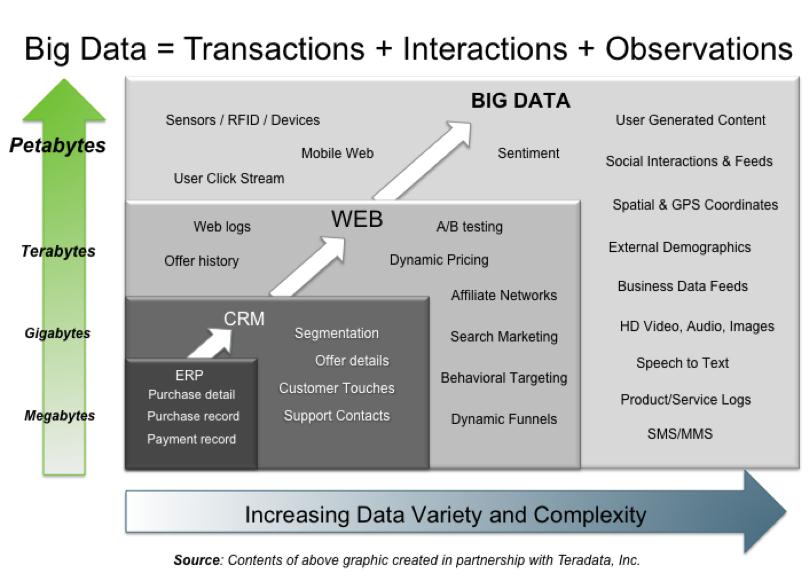
**Is big data unstructured?**

* Is that music unstructured ?
* Is the picture unstructured ?
* Is anything unstructured ?

**The Experts opinion**



**Overview of big Data**



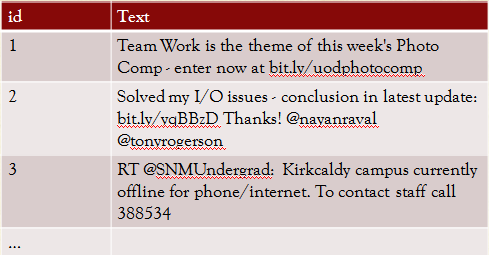
**What is big data ?**

“In information technology, big data[1] consists of datasets that grow so large that they become awkward to work with using on-hand database management tools.”

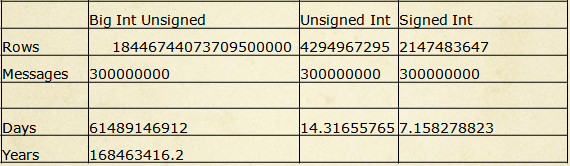
**N.B.** Any definition that relies on data “size” will become obsolete very quickly as data storage capabilities grows.

* Big data can be defined by the 3 V’s
* But it can also be data that is not atomic
* More realistically it can be defined as data that needs horizontal scaling to handle it.

**Examples of Big Data – Tweets**



**Data looks small but large volume**



**The twitter problem - Twitpocalypse**

* Overflow of status ids for 32 bit signed integers
* But beyond that, can we physically store data fast enough ?

**How big is a twitter message?**

A lot of extra data – retweets, urls, who sent, etc

**Example volume**

* Add at 100 per second
* 0.7 Terabyte per year
* Add at 1 million per second that’s
* 7 petabytes per year
* This is volume

**Variability**

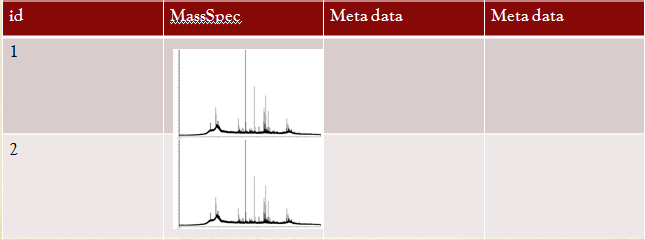
* Data is sparse and can be different sizes
* Over time the type of data changes
* Consider click through data, as pages evolve new data types and fields need to be stored
* Example include Farmville – Fencing Problem

**Fridges and Meters**

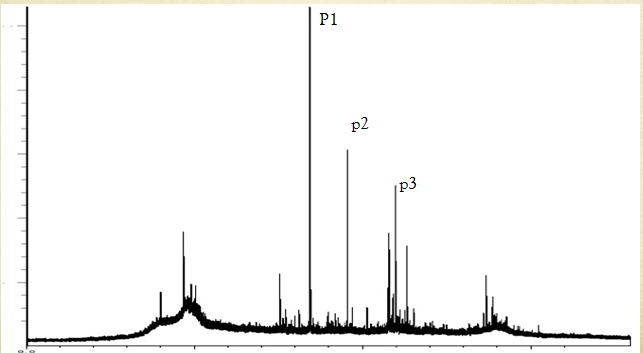
Suppose we wish to monitor Fridges, towards an internet of things

Problems with fridge types, storing infinite amount sof columsn, etc, better suited to schemaless.

**Mass Spec Example**



**Closer look at Mass Spec Data**



* Suppose we can say
* Select from MassSpec where height(p1) > (height (p2)-height(p3))
* We are now dealing with blob data as data we can run SQL against it’s properties
* This will return a new table

**But**

* This means we need a way of working out p1,p2 and p3 for any record on the fly
* Why ?
* Well suppose we have a new requirement to say:
* Select from MassSpec where sd(p1) > 0.5
* Where sd is the Standard deviation

**Problem**

* There are properties inside the MassSpec data that we do not know we need at the moment.
* So we can not simply add fields to the table for properties in the blob data.

**Solution - UDF**

* User Defined functions inside the dB

**EXAMPLES OF BIG DATA**

* Velocity -> sensors, Log files, Game User
* Volume -> DNA sequencing, Proteomic, Instagram, Social Media
* Variety -> Tweets, Sensors, Proteomic,

**WHAT IS IMMUTABLE DATA?**

Write once (or NEVER DELETE)

(Cassandra can be used to pull this off)

**Example** Police records (kept in the file for legal reasons but never erased)

Friend count

In a relational database friend count can answer a single question, but in an immutable database the records of when they are added e.g.

* 7 Nov ... added
* 8 Nov ... added
* 10 Nov ... removed

Answers who is added/removed, when and friend count at a specific time.

More data, more answers to more questions = Big Data

**HOW DO WE STRUCTURE IMMUTABLE DATA?**

* Denormalized
* Timestamps

E.g. Names table

Timestamp | Town ID | Name

Town table

Timestamp | Name ID | Town Name

If an ID changes then the record of it within the database still exists. Delete bad data to roll back to previous state.

**Big Data Strategies**

How can we handle Big Data?

How can we handle the volue and velocity?

**Properties of Big Data**

* You can answer more questions from raw data
* Processing data loses information
* Friend count example

**Big Data is not cleaned**

* Store the data as its entered
* Algorithms will improve over time to clean and make it semantic

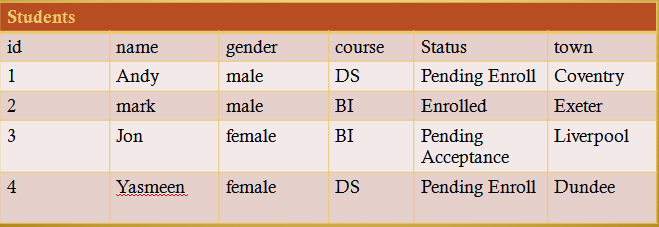
**Immutable Data**

**What is the cause of most problems with a Db ?**

Restoring data fom a back up – difficult and time consuming

**How easy is it to recover?**

**Traditionally stored like so - Unnormalised**



Use a timestamp and smaller tables



**Add and never delete**



**Never Delete**

* Your data sets are going to get bigger
* But you never lose a fact !
* You can construct the current state of affairs at any point
* Beware of regulation which may require data deletion.

Which brings us to:

**Data Vault**

***Definition***: The Data Vault is a detail oriented, historical tracking and uniquely linked set of normalized tables that support one or more functional areas of business. **It is a hybrid approach encompassing the best of breed between 3rd normal form (3NF) and star schema.** The design is flexible, scalable, consistent and adaptable to the needs of the enterprise. It is a data model that is architected specifically to meet the needs of today’s enterprise data warehouses.

* **Extensive possibilities for data attribution**.
* All data relationships are **key driven**.
* **Relationships can be dropped and created** on-the-fly.
* Data Mining can discover **new relationships between elements**
* **Artificial Intelligence can be utilized to rank the relevancy of the relationships to the user configured outcome**.

**End of Big DATA**

**-----------------------------------------------------------------------------**

**WHAT IS LAMBDA ARCHITECTURE**

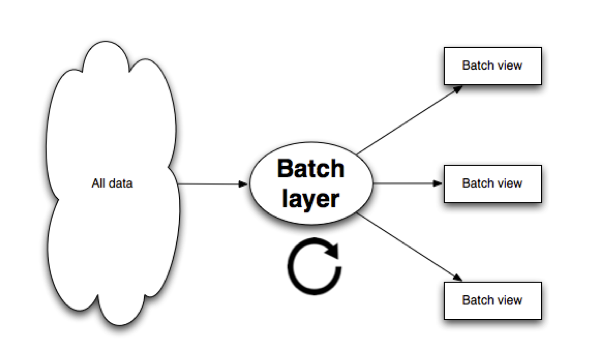
**Lambda Architecture**

**Three layers**

* Batch – friend Count example ( historical)
* Serving – Elephant DB ( Storage Engine)
* Speed – Storm(Up to date data)

**The Batch Layer**

* Stores master data immutable set
* Computes views – Continious operation using map reduce. New views are aggregated into the views when they are recomputered during the nxt map reduce
* Always out of date – each iteration could take hours



**Batch Technologies**

* Hadoop
* Cassandra ( Storage engine)
* HDFS – Hadoop distributed file system

**Serving Layer**

* Indexes the Batch’s output (which is likely to be flat files)
* Loads into dB for fast querying. But doesn’t need random writes which will make it faster !
* Flat files containing precoputer views. Responsible for indexing and exposing the views to they can be queried.
* As batch views are static, serving provides batch updates and random reads. Best used with Cloudera Impala to create table in hive metastore that points to HDFS.

**Serving Technologies**

ElephantDB – Merges the views to allow querying

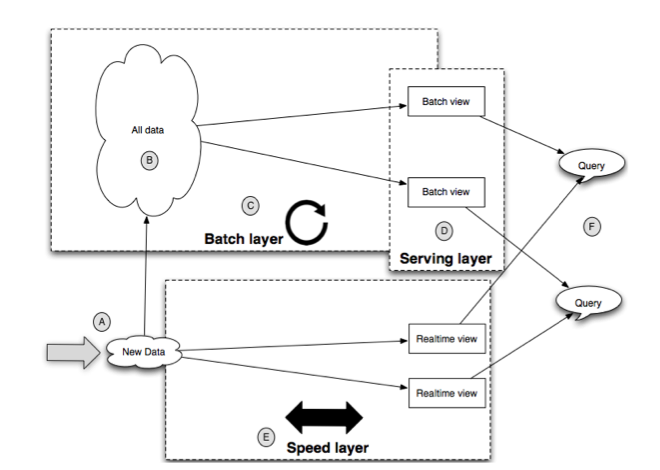
**Speed Layer**

* Batch layer is always out of date, perhaps by a few hours
* The Speed layer only works on new data **as it arrives**
* When the batch catches up throw it away
* Batch and realtime views are syncronised

**Speed Technologies**

* Storm
* Cassandra( Storage Engine)

**Overview of Lambda view**



**End of Lambda**

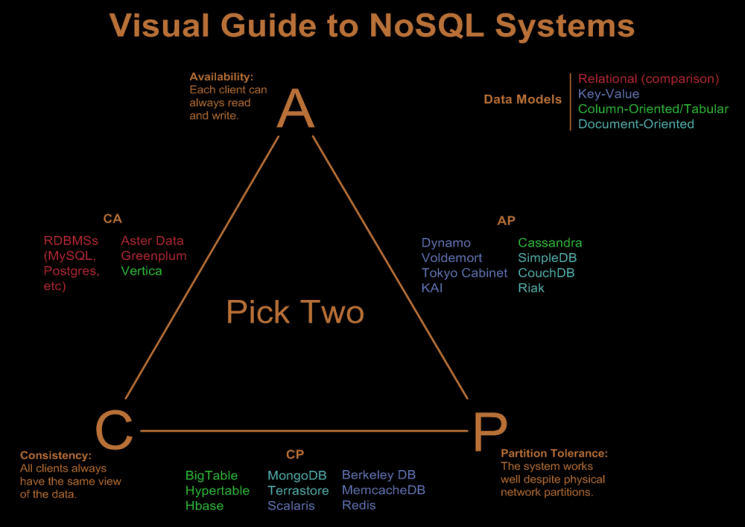
**What is CAP theorem ?**

CAP (or Brewers) theorem says:

It’s impossible for a web service to provide all of the following

* Consistency
* Availability
* Partition tolerance

So you have to pick two!



**Partitions**

* Essentially failing to achieve consistency within a set time causes a partition.
* You can sacrifice availability to ensure consistency
* Partitions are rare and if you have one server, almost never happen
* Partitions are caused by networks, failed nodes etc
* Important to note , partitions happen on failure
* So if you have a partition you must choose between availability and consistency
* Availablity
  + You choose not to answer requests until the partition goes away
* Consistency
  + You give the wrong answer until it goes away !

**What is availability?**

* Some noSql based systems feed into webpages
* Studies have shown that as little as 100ms can change the chances of a user purchasing from a web store.
* Availability is not just unavailable, but slow returns of data.

**What is consistency?**

* In ACID C= Consistency. The database will be consistent after a transaction. All rules such as unique keys will be obeyed.
* In CAP Consistency is across partitions. Will they give the same answer ?

**Data Replication**

1: Data sent to all replicas at the same time

* Each node might choose different order for updates. inconsistency across nodes

2: Data updates sent to an agreed upon location first

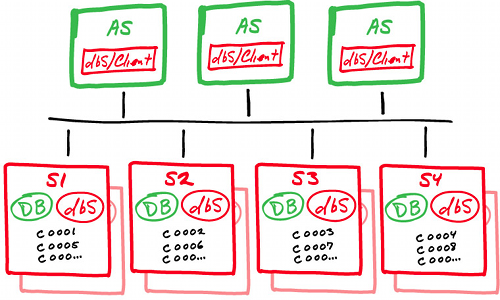
* Master node resolves update

3: Data updates sent to arbitrary location first.

* Again may go out of sync or introduce latency

**Sharding**

* Distribution of data across nodes
* Allows performance to be spread across multiple machines
* SQL databases can be sharded
* Not all NoSQL databases can be sharded



**Alternative Sharding architecture**

* Master node - Slave nodes
* Embed the Sharding logic in stored procedures

**But**

* Sharding is in the application
* Our alternative moves it from the app to the data layer
* But still hand coded by “application” writer

**Disadvantages of sharding**

* Sharding does not help redundancy
* If a node goes down, we still lose that data
* You may need to link sharding and replication together.

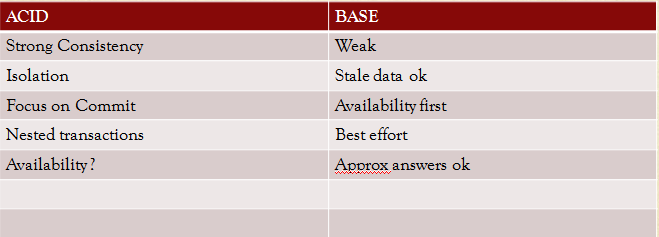
**What is ACID and BASE ?**

**ACID**: Atomicity, Consistency, Isolation, Durability

**BASE**:

* **B**asic **A**vailability: The system can be temporarily inconsistent to allow the information to be available
* **S**oft-state. Some data may not be persistent unless refreshed. The state of the system my change
* **E**ventual consistency. Eventually, if no new information is added the system is consistent

**Acid Vs Base**



**Soft-State**

* Soft state in BASE is very ill defined.
* It may be that data is set to time out when not needed, or is no longer accurate
* Example
  + We track a users phone
  + The user enters a car park and loses signal. Takes phone straight into a large building.
  + At some point we must assume the last location of the phone is wrong and time it out.
* Or the system will change state because of Eventual Consistency.

**Eventual Consistency**

* Eventually all nodes will tell the same story
* Isn’t this a mad idea ?
* Facebook (Actually not)
* The Internet is based on and Eventual Consistency dB
* DNS

**End of Principles of Big Data**

**-----------------------------------------------------------------------------**

**So what is NoSQL?**

* Throws away everything you know about Databases
* Is a family of different databases
* Lots of different “products”

BUT !

<http://nosql.mypopescu.com/post/1016320617/mongodb-is-web-scale>

They should only be used when it’s sensible, they are not magic sauce.

**NoSQL Types**

* Key-Value
* Column-family
* Document databases - Allow sharding across nodes
* Graph - Fast for graph like data and operations

**Examples of NoSQL Databases**

* CouchDb and Couchbase
* MongoDb
* Cassandra
* Riak
* Hbase
* Neo4j
* <http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis>

**Cassandra vs hbase**

* <http://bigdatanoob.blogspot.co.uk/2012/11/hbase-vs-cassandra.html>

**Google Big Table**

Father of many nosql databases including hbase

**Apache Dynamo**

Father of Cassandra and some others.

**Google Spanner**

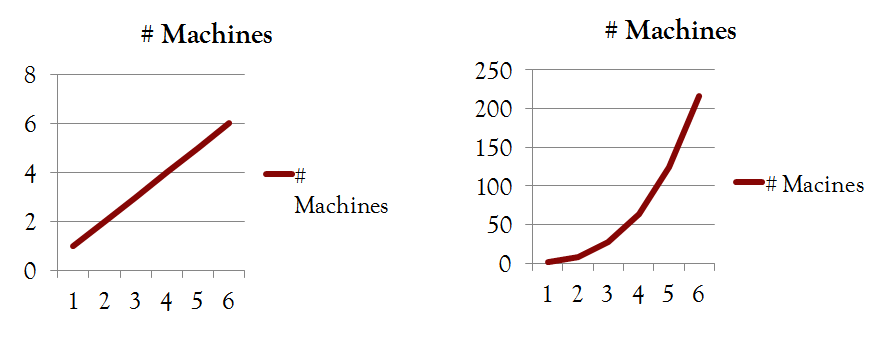
Spanner and F1

**Cassandra**

<http://www.datastax.com/documentation/articles/cassandra/cassandrathenandnow.html>

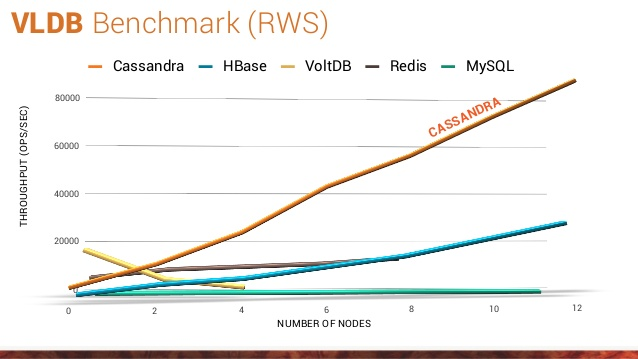
**Scalability**

* The ability to maintain performance under load by adding more resources.
* Linear vs none Linear Scalability



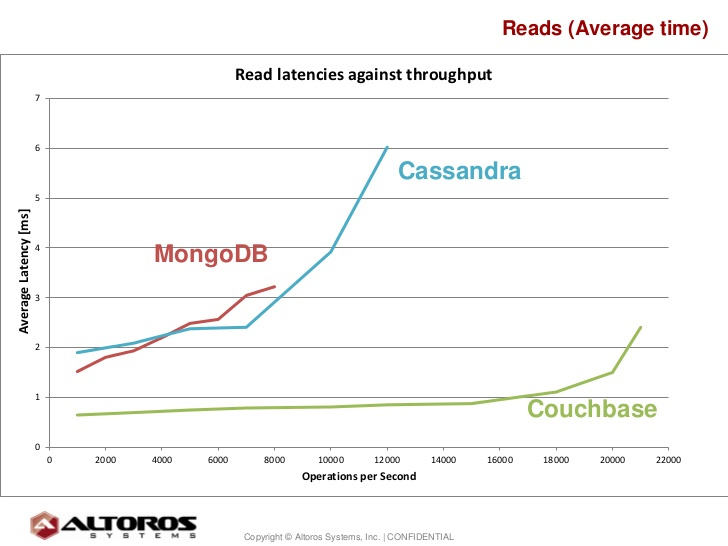
**What Scalability isn’t**

* It isn’t reducing latency
* If one db takes 100ms to return an answer
* Adding a second won’t reduce that number (it might increase it)
* BUT
* You may be able to do more operations !



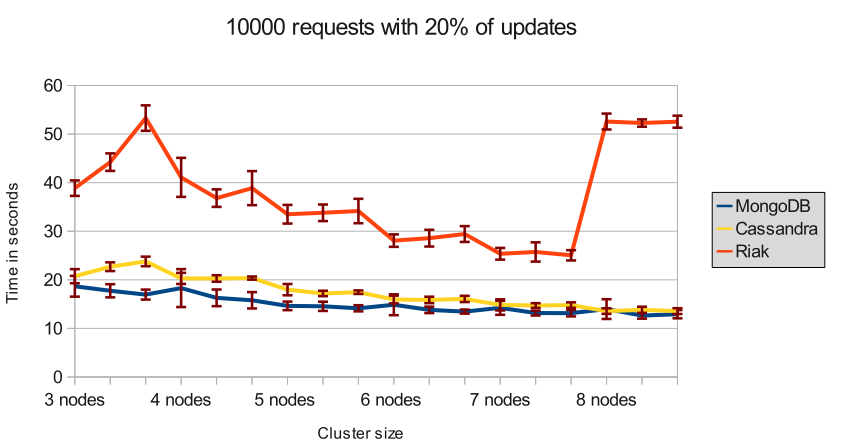
Benchmarks vary per database and every database seems to out perform the others depending who is doing the benchmarking.

**But**

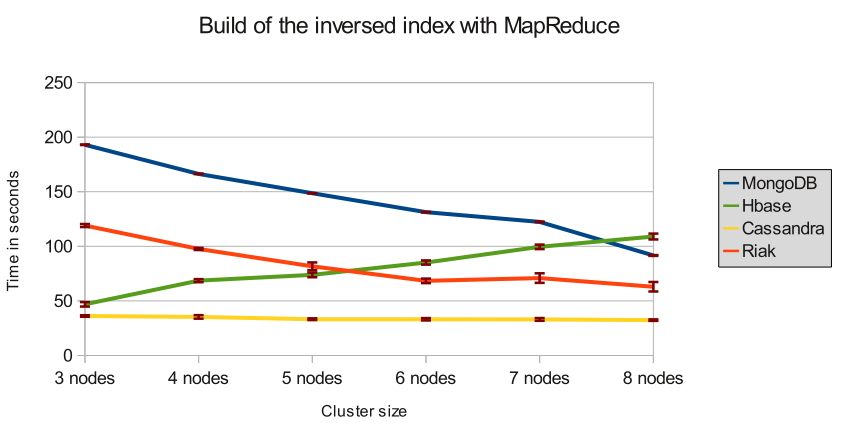


**Who to believe?**

* **Solving Big Data Challenges for Enterprise Application Performance Management**
* Also note latency vs throughput.

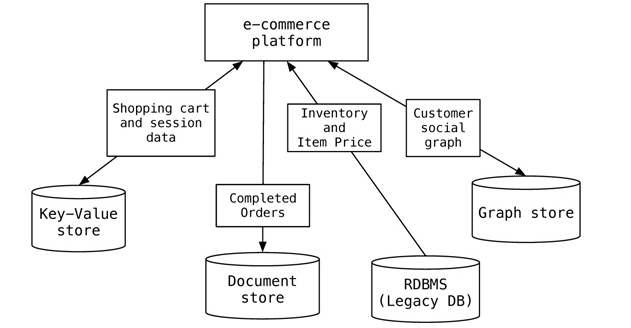


**Scaling**



**Polyglot Persistence**

We live in a world where storage is no longer SQL



**EXAMPLES OF NoSQL databases**

* Cassandra
* HBase
* memecached
* ElephantDB
* Riak
* Redis
* MongoDB
* Titan
* Neo4J

**End of NoSQL**

**-----------------------------------------------------------------------------**

**CASSANDRA**

A highly scalable distributed database.

* Distributed / Decentralized
* Column Orientated
* Key Value Store
* Fault Tolerant

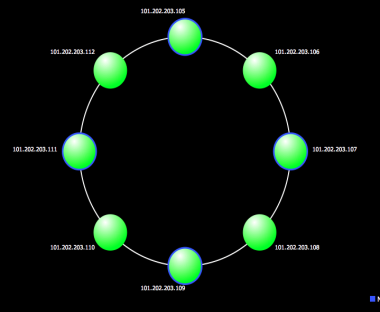
**Release Version**

* Written in java
* An Apache project (donated by Facebook)
* Currently at 1.2.10 and 2.0.1

**Network Topology of a Cassandra db**

* Multiple nodes
* Cassandra can be Rack Aware
* Keys can be replicated across nodes
* It’s essentially a DHT Distributed Hash Table
* Think BitTorrent

**Logical view of structure**



**Cassandra Partitoners**

* Based on the Partitioner type:
* **Murmur3Partitioner** (default):
  + uniformly distributes data across the cluster based on MurmurHash hash values.
* **RandomPartitioner**:
  + uniformly distributes data across the cluster based on MD5 hash values.
* **ByteOrderedPartitioner**:
  + keeps an ordered distribution of data lexically by key bytes

**Replication**

For any keyspace the Replication factor is set

* + How many copies ?
  + Where ?
  + Each node is the same, no master nodes.
  + How do we find a copy ?

**Who has the latest copy?**

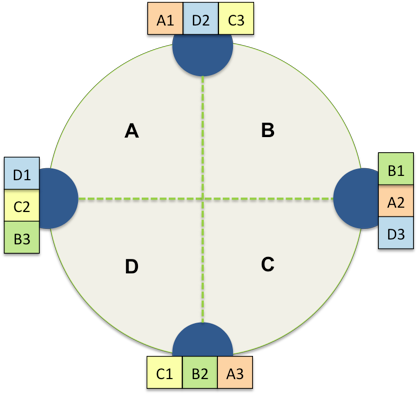
* So we have multiple copies of the data,
* Data is updated on one node,
* Takes time to update across all nodes
* So who has the latest data ?

**Replication Strategy**

* Simple Strategy Single Data Center
* Network Topology Multiple Data Center

**Simple Strategy**

* First node determined by partitioner
* Next replica goes on next node in ring

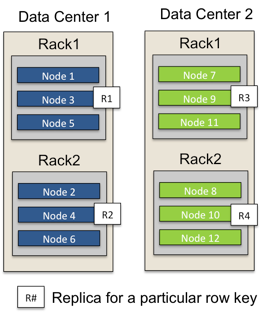
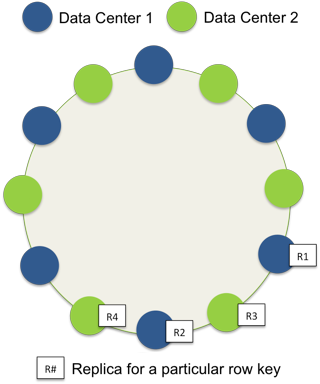


Network Topology

We have two problems

* + Cross Data Center latency
  + Failures (data center and network)

We can set the number of replicas per datacenter



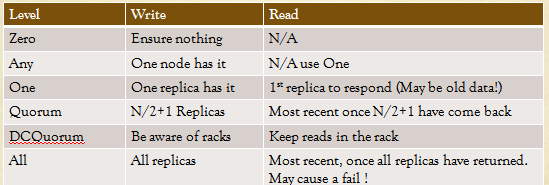
**Snitches**

To use network topology you must configure ”Snitches”.

* A snitch maps IPs to racks and data centers
* See:
  + <http://www.datastax.com/docs/1.1/cluster_architecture/replication>

**Consistency Levels**

* How many copies do we want to respond ?
* Can be set for column families
* Based on the replication factor not the number of nodes
* Separate values for read and write
* Can be set for individual operations



**Strong vs Weak Consistency**

Weak will read data and then “repair”it

* Zero, Any,, One

Strong will repair and then read it

* Quorum, All

**Write/Read speed / Mechanism**

* Writes are much faster than reads
* Writes go to commit log (Crash recovery, writes won’t return until its in here)
* then memtable
* then flushed to SSTables
* All writes are sequential
* BUT Reads are now faster in later versions

**Reads**

* Reads need to find the data, find the nodes and find if its in memory
* Remember we can read from any node.

**Adding Nodes is good**

* Adding nodes adds performance
* Adding nodes adds replicas of data (resilience)
* BUT
* Make sure your ring is balanced, Cassandra doesn’t like to be unbalanced.

**Stress test commands**

* ./stress -d 192.168.1.10,192.168.1.11,192.168.1.12 -o insert -I DeflateCompressor
* Note: nodes to use
* You will get different performance if you insert to less nodes than you have in your ring

**Adding Nodes Procedure**

**Adding a node is not easy**

* + Must seed form a known node
  + Use a program to calculate new keys
  + Bring up new node with the correct key in cassandra.yaml
  + Use node tool to move other nodes

**Calculating Keys ( Random Partitioner)**

Calculate keys manually or use

* http://www.datastax.com/docs/1.1/initialize/token\_generation

**End of Cassandra Overview**

**-----------------------------------------------------------------------------**

**Calculating Keys ( Murmur Partitoner)**

For example, to generate tokens for 6 nodes:

* python -c 'print [str(((2\*\*64 / 6) \* i) - 2\*\*63) for i in range(6)]'

**Moving existing nodes**

* Use nodetool
* sudo ./nodetool -h 192.168.1.10 move 42535295865117307932921825928971026432
* And cleanup
  + ./nodetool -h 192.168.1.10 cleanup

**What happens if a node goes down?**

**Cassandra Internals**

VNodes

* Previously nodes hold a contiguous range of keys
* With vnodes it’s distributed

**Whats the importance of the ring and tokens**

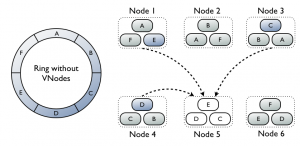
Key value stored

What about vnodes?

Rather than a key on a single node, nodes contain multiple keys to backup other nodes

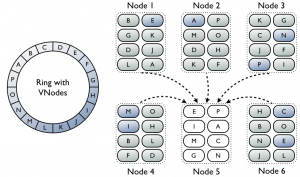
**Effects of Vnodes**

Node repair without



**Repair with Vnodes**

Node repair with VNodes



**Problem with compression**

* Cassandra uses compression for performance
* Started in vesion 1.0
  + 2x-4x reduction in data size
  + 25-35% performance improvement on reads
  + 5-10% performance improvement on writes

**Reads are nearly as fast as writes with compression.**

**Compression Types – Used between nodes**

* Two types:
  + **Google Snappy Compressor** (Faster read/writes)
  + **DeflateCompressor** (Java zip slower,better compression)
  + **LZ4Compression** (1.2.2 or later)
* **Compression is per table.**

**Cassandra Data model : Columns**

* Key
  + Name: Value

**Example:**

* Authors
  + Andy
    - Name: Andy
    - Tel: 01382 555444
    - Building: QMB
  + Janet
    - Name: Janet Borlands
    - Twitter: jantwit
    - Email: Janet@abc.com

**Things to note** : Name Value columns can be different for different keys

* Names and Values are stored as Byte Arrays
* Keys need not be strings
* This forms a column Family.

**Columns Ordering**

* ASCII,
* UTF-8,
* Long,
* UUID (time)

**Serialisable**?

**Lightweight Transactions**

* Suppose 2 people try to create an account with the same name at the same time,
* They will over write each other without a transaction, so:
* INSERT INTO customer\_account (customerID, customer\_email) VALUES (‘LauraS’, ‘lauras@gmail.com’) IF NOT EXISTS;

**4x Slower implementing but sometimes necessary.**

**Serial Consistency**

* A write must be written to the commit log and memory table on a quorum of replica nodes.
* Reading the current (and possibly uncommitted) state of data without proposing a new addition or update. If a SERIAL read finds an uncommitted transaction in progress, it will commit it as part of the read.

**Problems with open source software**

* Versioning !
* Things Change !
* Documentation is wrong !
  + <http://prettyprint.me/>
* End up reading unit tests to actually program.

**Design Anti-Patterns**

* 1: Don’t run C\* on a SAN
  + Cassandra likes a real disk real close (but it does know about SSDs)
* 2: Commit log and Data on same disk
  + Not SSD or EC2 though
* 3: Careful with the heap size
* 4: Low file handle limit
* 5: You don’t need a load balancer
* 6: Don’t read or write to a single node
* 7: Don’t let your ring get unbalanced
* 8: Watch your partitioner. Don’t use Byte ordered
* 9: Watch the batch insert size, must fit in memory !
* 10: use a 64 bit JVM
* 11: use the sun JVM
* 12: try not to use supercolumns (composites are coming!)
* 13: Use Datastax Operations center !

**End of Principles of Cassandra Data odel**

**-----------------------------------------------------------------------------**

**Interfacing with Cassandra**

**Changes very often**

* **Based on Thrift**
  + **http://thrift.apache.org/**
* **Large number of Languages supported**

**What about CQL ?**

Like SQL but

No joins

Funny indexing

No random selection of data (only access stuff that has been previously specified as indexes)

No name value pairs

**Cassandra Query Language**

* **Latest development is CQL**
* **Looks like SQL but isn’t !**
* **No joins !**

**but**

* **Cassandra 1.2 introduces CQL3**
* **This is different !**

**Cassandra-cli**

* **Allows us to query Cassandra at a lower level**

**Examples of Cassandra-CLI**

**Create a keyspace:**

* create keyspace Keyspace2 WITH replication = {'class':'SimpleStrategy', 'replication\_factor':1};

**Create Table**

CREATE TABLE Users (

id uuid Primary Key,

name text,

email\_addresses set<text>) ;

**Insert Data**

insert into Users (id,name)

values (88b8fd18-b1ed-4e96-bf79-4280797cba80,'andy');

**Update Data**

update Users set email\_addresses = email\_addresses + {'andy@r2-dvd.org'} where id =88b8fd18-b1ed-4e96-bf79-4280797cba80;

Select and then another update

update Users set email\_addresses = email\_addresses + {'acobley@computing.dundee.ac.uk'} where id =88b8fd18-b1ed-4e96-bf79-4280797cba80;

**Select User**

**Try and select based on user**

* **select \* from timeline where name=’andy';**

**Composite Primary Keys**

CREATE TABLE timeline ( user\_id varchar, tweet\_id uuid, author varchar, body varchar, PRIMARY KEY (user\_id, tweet\_id));

* **Note must have one column not in the key**
* **insert into timeline (user\_id, tweet\_id, author, body) values ('abcde', 88b8fd18-b1ed-4e96-bf79-4280797cba80 , 'andy' , 'this is a tweet');**

**Indexing**

select \* from timeline where author='andy';

* Create an index on the author:

create index authors on timeline(author);

**Disadvantages of Indexing**

* **Indexes can be very inefficient:**

**Maps, Sets, Lists – Types of Colections**

* Sets unordered items
* Map name value pairs
* List ordered items

**Sets**

CREATE TABLE users (

user\_id text PRIMARY KEY,

first\_name text,

last\_name text,

emails set<text>

);

INSERT INTO users (user\_id, first\_name, last\_name, emails)

VALUES('frodo', 'Frodo', 'Baggins', {'f@baggins.com', 'baggins@gmail.com'});

**List**

ALTER TABLE users ADD top\_places list<text>;

UPDATE users

SET top\_places = [ 'rivendell', 'rohan' ] WHERE user\_id = 'frodo';

**Maps**

ALTER TABLE users ADD todo map<timestamp, text>

**UPDATE users**

SET todo =

{ '2012-9-24' : 'enter mordor',

'2012-10-2 12:00' : 'throw ring into mount doom' }

WHERE user\_id = 'frodo';

**Programming Cassandra Applications**

* Hector
* Astyanax
* JDBC
* Datastax Java Driver

**End of Principles of Interfacing with Cassandra**

**-----------------------------------------------------------------------------**

**Cassandra Design Patterns**

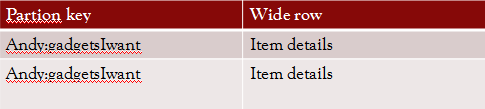
* **Cassandra is not for all use cases !**
* **RDBMS do still have a very big place**
* **Other NoSQL db’s may be more appropriate**

**No joins**

* **Don’t even think about trying to program joins**
* **You could do it at the application layer**

**Shopping card Example**

* In this example each user can have more than one cart
* Each column entry is a cart item
* Extra details are stored in the cart



CREATE *TABLE shopping\_cart (*

username *varchar,*

cart\_name *text,*

item\_id *int,*

item\_name *varchar, description varchar,*

price *float,*

item\_detail *map<varchar,varchar>,*

*PRIMARY KEY ((username,cart\_name),item\_id) );*

**Whats new here ? - MAP**

**User Activity for websites**

* We want to record user actions.
* We are only interested in the most recent ones
* We are looking for specific actions that might trigger our website to do something
* We will use Clustering
* This orders the columns of a table

**CREATE TABLE** user\_activity (

username varchar,

interaction\_time timeuuid,

activity\_code varchar,

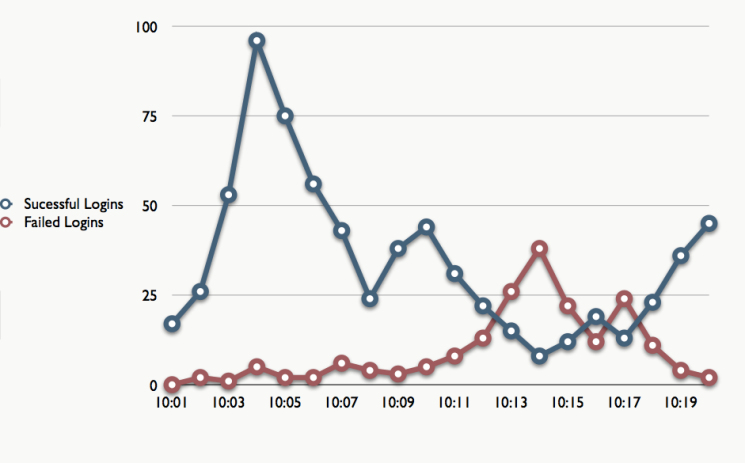
detail varchar,

**PRIMARY KEY** (username, interaction\_time)

) **WITH CLUSTERING ORDER BY** (interaction\_time DESC);

**Log Files**

* Machine Log files,
* So lets assume we have large labs of machines constantly in use.
* Or a web service that is in constant use
* We want to monitor in real time
* Lets record logon and logoff events



**What can log files show us?**

Hackers – other data

**Clustering – Order By Problem**

**CREATE TABLE** login\_success (

source varchar,

date\_to\_minute varchar,

successful\_logins counter,

PRIMARY KEY (date\_to\_minute)

) **WITH CLUSTERING ORDER BY** (date\_to\_minute **DESC**);

**CREATE TABLE** login\_failure ( ! source varchar,

date\_to\_minute varchar,

failed\_logins counter,

PRIMARY KEY (date\_to\_minute)

) **WITH CLUSTERING ORDER BY** (date\_to\_minute **DESC**);

**What problem does this schema create?**

Large amounts of logins or failures will cause one machine to be hammered.

**How the problem is solved?**

**CREATE TABLE** login\_success (

source varchar,

date\_to\_minute varchar,

successful\_logins counter,

PRIMARY KEY (source,date\_to\_minute)

) **WITH CLUSTERING ORDER BY** (date\_to\_minute **DESC**);

**CREATE TABLE** login\_failure (

source varchar,

date\_to\_minute varchar,

failed\_logins counter,

PRIMARY KEY (source,date\_to\_minute)

) **WITH CLUSTERING ORDER BY** (date\_to\_minute **DESC**);

**What about raw files?**

**PRIMARY KEY** ((source,date\_to\_minute),timestamp) );

**End of Principles of Design Pattersn**

**-----------------------------------------------------------------------------**

**WHAT IS STORM ?**

Spouts + Bolts

Connect the data spouts to the bolts

D3.js

Used in Speed layer

**MAP REDUCE**

**What is Map reduce**

A paradigm for parallel processing data by mapping it to key value pairs and then reducing them down.

If you have a big set of data, split it into smaller file and sort them out then bring them all together until you get the output file.

**WHAT IS HADOOP**

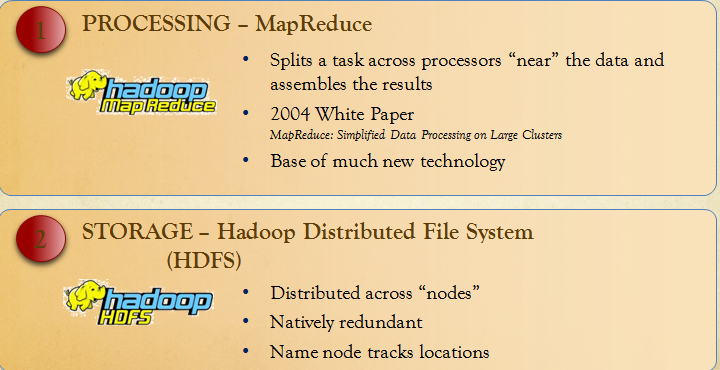
**History and Background**

* **Google 2004 - MapReduce: Simpliﬁed Data Processing on Large Clusters**
* [**http://static.googleusercontent.com/external\_content/untrusted\_dlcp/research.google.com/en/us/archive/mapreduce-osdi04.pdf**](http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/archive/mapreduce-osdi04.pdf)
* **A bigger horse?**
* **a screwdriver or a hammer?**

**What is Hadoop?**

Can run on any machine

Makes use of its disk drive for HDFS

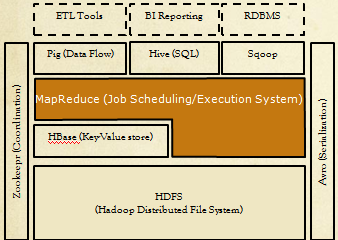
****

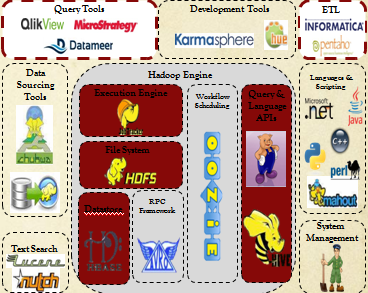
**Extensions to Hadoop**

****

**Vendors**

**Apache, Hortonworks, Cloudera, MapR**

****

****

**Use Cases**

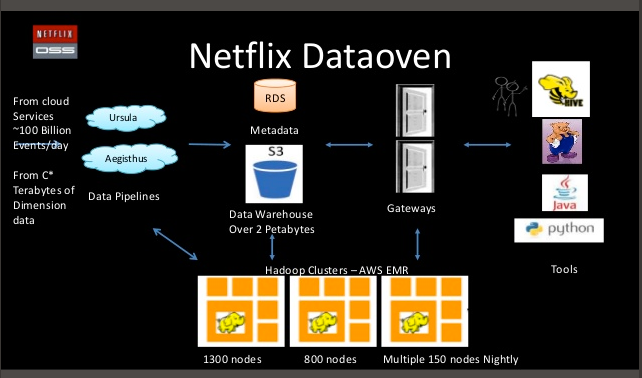
* **Web Log Processing**
* **Text Search**
* **Social Network Analysis**
* **Batch Pre-Processing**
* **Search Engine Ad-word optimisation**
* **Last.fm – analytics**
* **eCircle – automated email recommendations**
* [**http://wiki.apache.org/hadoop/PoweredBy**](http://wiki.apache.org/hadoop/PoweredBy)

**Klout**

* [**http://www.microsoft.com/casestudies/Microsoft-SQL-Server-2012-Enterprise/Klout/Data-Services-Firm-Uses-Microsoft-BI-and-Hadoop-to-Boost-Insight-into-Big-Data/710000000129**](http://www.microsoft.com/casestudies/Microsoft-SQL-Server-2012-Enterprise/Klout/Data-Services-Firm-Uses-Microsoft-BI-and-Hadoop-to-Boost-Insight-into-Big-Data/710000000129)

**Netflix**

* [**http://www.youtube.com/watch?feature=player\_detailpage&v=oGcZ7WVx6EI#t=614**](http://www.youtube.com/watch?feature=player_detailpage&v=oGcZ7WVx6EI)

****

**HDFS – Hadoop Distributed File System**

**Is scaleable -** Add more machines, more space and throughput

**Controllable -** You choose how the data is stored

**It is a file system so you can partition the data uisng directories**

**Versions**

**1.x**

* **Most stable version**
* **Classic Map Reduce**

**0.2**

* **1.x is based on this**
* **Classic Map Reduce**

**2.0**

* **Newest version**
* **New Map Reduce (YARN)**
* **Federated filesystem**

**Releases are parallel and confusing.**

**The Hadoop Approach**

* **Scale up or scale out**
* **Issues between nodes in a cluster**
* **Issues with Disk access** 
  + **Scan or seek**
* **data to the analysis vs. analysis to the data**

**What is Hadoop – Map Reduce ?**

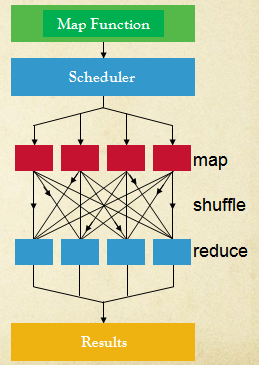
**A parallel programming framework**

* **C++, Java, Python, etc.**
* **Harness 1000s of CPUs**

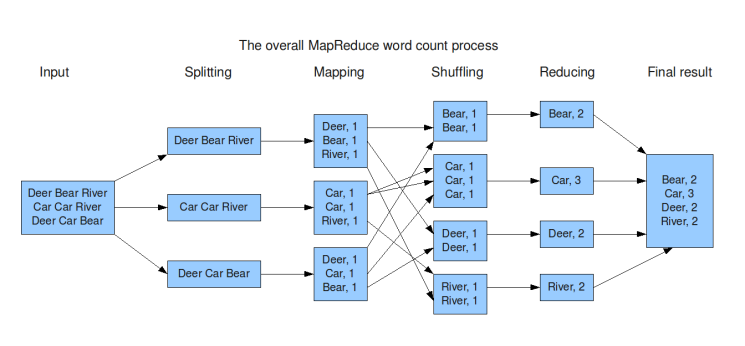
**MapReduce provides**

* Automatic parallelization
* Fault tolerance
* Monitoring & status updates

***“MapReduce allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.”***

****

**Example of a map reduce job**

****

**Classic example of Word Sort / Count**

**What is Hadoop – HDFS?**

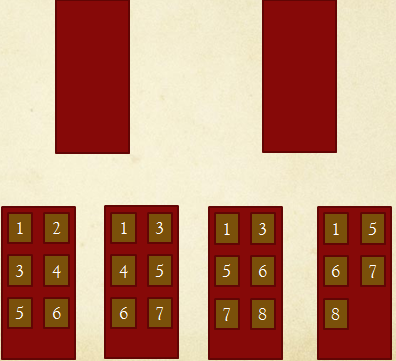
* **Hadoop Distributed File System**
* **Default replication factor of 3**
* **Files split into 64Mb chunks and copied around Data Nodes**

**What is hadoop architecture?**

**Primary NameNode & JobTracker**

**Secondary** NameNode

**Data Nodes – Task Trackers**

****

**Name Node –** In version 2 there is more than one name node – This adds resilience

**Sizing a hadoop cluster – By Data**

**By Data**

* **80Tb of data**
* **Hadoop default replication = 3, need 240Tb storage**
* **1\*2Tb disk per CPU core**
* **4Gb Ram per CPU core**
* **Example servers with 12 cores**
* **10 servers for data with 24Tb Disk and 48Gb Ram**
* **Plus 2 servers for namenode and secondary namenode**

**Key requirement of hadoop** is disk local to CPU so one disk per core

**Sizing a Hadoop Cluster – By Preformance**

* Sample workload and code, run on Amazon EC2 instance
* Hadoop scales linearly, use test results to calculate required cluster size

**Running Hadoop**

* Use the HDFS commands to manipulate files
* Be able to run MapReduce Jobs
* Understand what is possible with the Hadoop command line
* Monitor HDFS file system and MapReduce tasks

**HDFS Commands**

* $ hadoop <command> <arguments>
* $ hadoop fs –ls <directory>
* $ hadoop fs –cat <filename>
* $ hadoop fs –mkdir <directory>
* $ hadoop fs –rmr <directory>
* -chgrp, -chmod, -put, -get, -stat

Running MapReduce Jobs

* $ hadoop jar <jarfile> <input> <output> <arguments>
* Input: file or directory
* Output: new directory, error if it exists
* Results will be in one file per reducer
  + Part-r-00000, part-r-00001
* Logs in output directory

**Cluster Maintenance**

* **Web Interfaces**
  + [**http://namenode:50070**](http://namenode:50070/) **- HDFS**
  + [**http://namenode:50030**](http://namenode:50030/) **- MapReduce**
  + [**http://localhost:8088**](http://localhost:8088) **- resource manager (v2)**
* **$ hadoop namenode –format**
* **hadoop dfsadmin -report, -safemode**

**Cluster Maintencen - Configuration files**

* **/usr/lib/hadoop/conf** 
  + **hadoop-env.sh**
  + **core-site.xml**
  + **mapred-site.xml**
  + **hdfs-site.xml**
  + **masters**
  + **slaves**

**What makes it good for map reduce**

**What elements do we have in hadoop**

* Data Node - Data blocks
* Name Node - Job trace, Data tracker
* V1 = 1, V2 = 2
* V1 single point of failure while V2 you don't (more robust).

**What is HDFS**

Hadoop Distributed File Store - Replace it with Cassandra or HBASE and probably other things as well.

**End of Principles of Hadoop**

**--------------------------------------------------------------------------**

**What are PIG and HIVE**

PIG is a simplified

HIVE is a simplified query language that sits on top of PIG

Looks like SQL but doesn't give the performance of SQL

**Pig Overview**

* High level query language built on top of MapReduce
* 2 components
  + Language: Pig Latin
  + Compiler: Translates Pig Latin to MR jobs
* Created by Yahoo to simplify data queries
* The language that can “Eat any data”

**Example of Execution Steps**

* A = LOAD 'm100\_5.txt';
* B = LIMIT A 1;
* C = FOREACH B GENERATE $0, $1;
* DUMP C;

Pig does not compile or execute until results need to be returned e.g. DUMP or STORE used.

**Grunt Shell**

* **Pig**

1. --**Load file into Pig**

grunt> A = LOAD 'email/m100\_5.txt' USING PigStorage ('|') as (id, from, datetime, servid, subject, folder);

1. **--Limit to first line of file**

grunt> B = LIMIT eg 1;

1. **--Choose fields id, from and datetime**

grunt> C = FOREACH eg1 GENERATE id, from, datetime;

1. --**Look at contents of query**

grunt> DUMP C;

**Pig Latin**

* LOAD, DUMP, STORE
* FILTER, DISTINCT, FOREACH, SAMPLE
* JOIN, GROUP
* ORDER, LIMIT
* UNION, SPLIT
* DESCRIBE, EXPLAIN, ILLUSTRATE

**File PigExamples.txt**

* Basic Load
* Count number of mails per user
* Find users talking about fraud

**Hive Overview**

* High level query language built on top of the MapReduce framework
* Allows a schema to be defined on the data in HDFS files
* This schema can then be queried with a SQL-like language
* Commands are compiled into MapReduce jobs and executed on the cluster
* Often used in conjunction with Pig Scripts

**Features of Hive**

* Hive is not a relational database
* Hive does not include transactions or any notion of transactional processing
* Not designed for real-time query response
* A database is used in the back-end to store metadata for schemas, all user data is stored in HDFS files.

**Looks remarkably like SQL**

**Data Management**

* **Data in Hive Store (/user/hive/warehouse)**
  + LOAD DATA LOCAL INPATH ‘<path>’ INTO <table>
* **Table created on a HDFS directory**
  + Implications for updates, deletes, inserts
* **HQL Insert command** 
  + What does this do?
  + Overwrite option

**Examples of Hive**

SELECT \* FROM tweets LIMIT 1;

SELECT tweetDate, count(\*) FROM tweets

GROUP BY tweetDate;

SELECT tweetID , COUNT(\*) as tweetCount

FROM tweets GROUP BY tweetID

ORDER BY tweetCount DESC LIMIT 10;

**Execution of Hive**

* Apache Weblog in the examples dir
* Upload to HDFS
* Create table in HIVE using SERDE formatter

**Does not return instantly, does a batch job!**

**Hive connectors**

* ODBC, JDBC allow “BI” tools to access Hive Data
* E.g. Excel ODBC driver from Microsoft

**HCatalog – Scheme First or Scheme Last Dilema**

* Schema first or schema last
* What are the advantages/disadvantages of each?

**HCatalog - Overview**

* Hcatalog is a repository for schema definitions
* The definitions can be re-used by parts of the eco-system - **Pig, Hive, MapReduce code**
* Provides consistency, allows for partioning for performance

**Syntax – Similar to Hive**

CREATE TABLE massspec (

fileID int,

scanLvl int,

scanNum int,

mz double,

intensity double)

PARTITIONED BY fileID

**If Above definition stored in file masspec.hcat execute it as follows**

$ Hcat –f masspec.hcat

**Implications**

* Other applications outside of the Hadoop eco-system can use the HCatalog schemas to query data in HDFS
* E.g. SQL-H in Teradata Aster can query HDFS files using standard ANSI-SQL
* **End of Principles of Pig / Hive**
* **---------------------------------**

**FUNCTIONAL PROGRAMMING**

**How is it different from procedural/object oriented**

- Variables are immutable

**How do we process lists**

[H|T] = [Some, list, more, values]

function([H|T]). % start

H = Some.

T = [list, more, values].

function(T). % progress through list

fact(4) = 4 x 3 x 2 x 1

fact(1) ->

fact(N) -> N x fact(N - 1).

Does

fact(4) = 4 x fact(3)

fact(3) = 3 x fact(2) ...

**Erlang**

**What is Erlang?**

* **A functional language**
  + Evaluation of functions rather than iterating through code
  + Avoids state and mutable data with their side effects.
* **Lightweight concurrency model** with no shared data.
* **Highly fault tolerant**. Let it fail paradigm.

**History**

Developed in the 1980’s by Ericsson for telecomms apps

* + These need to be very available
* OTP library in 1996
* 1998 Erlang goes open source

Current Version is R168

**Assigning Variables**

* **Atoms and variables**
* **Comparison**
* **Assign once**

**Atoms – Constant Literal**

* Lower case
* Can be considered as an enumeration !
* In quotes if you want
* Examples
  + one
  + ‘one’
  + blue
  + ‘one two three’

**But ‘one’ is not a string**

* + one==‘one’ = true
  + one==“one” = false

**Recursion- Is the basic tool in Erlang**

* Call a function from within a function repeatedly until some stop condition
* **Pattern Matching** allows it to happen

**Pattern Matching**

* Can be used to bind to a new variable
* Can check if both sides the same only if the same shape.
  + Sum=1+2.
  + Sum=Sum+1.
  + Sum=3.

**Functions**

**Consist of a head and a body**

* Lines in the body separated by commas
* Clauses separated by semicolons

**Factorial Example**

Factorial(0) ->

factorial(N) ->

(N-1) \*factorial(N-2).

**To use functions we need to compile**

* Put the following in a text file
* -module (basic).
* -export ([mirror/1]).
* mirror(Anything) -> Anything.
* Save it to basic.erl
* Open erl. C (basic).
* basic:mirror("hello world").

**Modules**

* Groups functions together
* You must export a function if it will be used outside the module –export([function/0]).

**Incrementing and Printing**

count(N)

-> io:fwrite("~3.3.0w " , [N]),

count(N+1).

**Guards**

Guards can be used to “guard against” illegal arguments

* + factorial(0) -> 1;
  + factorial(N) when N >0
  + -> N \* factorial(N-1).

E.g. running this with -1

**Catch**

catch fact:factorial(-1).

**Type Matching**

Try: fact:factorial("one"). What do you get ?

factorial(N) when N >0 and is\_integer(N)

**Tuples**

Data type to store a collection of patterns

* {“one”, one, 2 ,3}
* String
* Atom
* Integers.

Can be tested for matching try the following:

* {1,2,3}=={1,2,3} = true
* {one,2,3} == {1,2,3}. = false
* {one,2,3} == {one,2,3}. = true
* {one,2,3} == {2,one,3}. = false

**Can be used to bind Variables**

* {A,B,C} = {one,2,3}.
* What values do A,B and C have ?

**What about**

* {D,D,E} = {1,1,3}. = true
* {F,F,G}= {1,2,3}. = false

**We can also not care about some tuple elements**

* {A, \_, B} ={1,2,3}
* {C,\_,\_,\_}={1}

**Can be used in functions**

**Tuples can contain tuples**

* {1,{one,2,3},2,{two,3,4}}
* {A,B,C,D}={1,{one,2,3},2,{two,3,4}}.

**Lists**

Look like tuples but aren’t !

* [1,2,3,4]
* [“one”,2,3,4]
* [{circle,3},{rect,2,3},{square,10}].

**Just like Tuples**

[A,B,C]= [{circle,3},{rect,2,3},{square,10}].

**But what about:**

* [A,B,C]= [{circle,3},{rect,2,3},{square,10},{square,10}].
* False
* [A,B,C,\_]= [{circle,3},{rect,2,3},{square,10},{square,10}].
* true

**Heads and Tails**

* We can split a list with
* [H|T]=[1,2,3,4].
  + Try it a few times.
  + There is nothing special about H and T
* We can go through a list

each([H|T]) ->

io:fwrite("~w ",[H]),

each (T);

each([])->[].

Example of H&T recursion is sending a list of tuples to a function to calculate different types of share areas.

**List Comprehensions**

**Runs through a list and applies a function.**

* Given A=[1,2,3]
* [X+1 || X <-A]

**Can also filter**

* [X+1 || X <-A, X>2].

**Example Quicksort**

sort([Pivot|T]) ->

sort([ X || X <- T, X < Pivot]) ++

[Pivot] ++

sort([ X || X <- T, X >= Pivot]);

sort([],D) -> [].

**For any list element,** To the left is a list of items < To the right is a list of items >

**End of Principles of Syntax**

**-----------------------------------------------------------------------------**

**Erlang Concurrency**

Erlang has **built in concurrency** + Can be **across networks**

**Advantages of Concurrency**

* Can speed up processing
  + Especially in today’s multiprocessor world
* Because there is no shared memory, corruption is unlikely
  + Implementation is simplified
* Memory leaks can be avoided
* Message passing is asynchronous
* Processes are lightweight (not managed by the OS)

**Spawning a new process**

* spawn(modulename, function, [])

**Example**

**nowtime()-> [date(),time()].**

This is simple function that returns date and time

**spawn(datetime, nowtime, []).**

Returns process ID of spawn

<0.32.0> - self().

**What happens when you spawn a process?**

* When you start a process, it will start
* And die without telling you
* Now will it return anything unless you make it !

**Process information**

You can get process information using:

**Processes(). & i().**

**Messaging Passing**

* Messages can be passed between any process
* They can be of any type, including long lists
* They are copied to avoid memory corruption
* They can be used to return values to the main process

**Example of Messaging passing**

* Pid ! Message
* self() ! "hello world".

**Flush(). Gets the message of the queue.**

**Sending a result - Stages**

* **Send Process ID to function**
* **Pid ! Message**
* **Sitting in buffer**
* **Flush the buffer**

**Delay**

**Timer:sleep(5000).**

**Receiving a message**

* Receiving is a blocking operation
* Messages in the incoming queue are matched against the clause in the receive.

**Example**

receive

{From,Msg}-> io:fwrite("~w",[Msg]);

\_Other -> {error, unknown}

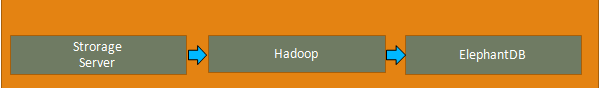
end.

**ElephantDB - Overview**

* Created by Nathan Marz - BackTrack
* ElephantDB was part of the solution to the challenges they were facing
* Process lots of data (>30TB) in real time.
* Complex analysis of lots of data

**Use Case**

* **ElephantDB is a serving layer technology**
* **In order for the batch layer views to be useful, they need to be queried with low latency**
* **Serving layer achieves this**

****

**Programming Language and Commercial Status**

* Written in the Clojure Programming Language
* BackType was acquired by Twitter
* Unclear but it can be assumed ElephantDB is being used presently by Twitter
* Used by other companies(YeildBot)
* Currently an open source project

**Strengths**

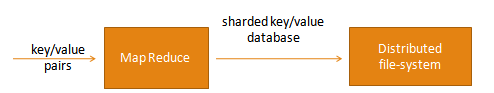
* Simple to configure
* Robust and Fault tolerant
* Small code-base
* Trivial to Scale
* Easy to maintain in terms of developer time

**Weaknesses**

* Lack of documentation and experienced developers
* Due to the fact that ElephantDB uses data from the batch layer (Hadoop), it is always behind with the latest changes to the data.
* The Lambda Architecture solves this with the Speed Layer.

**Data Model**

* Key/value serving layer database
* Specialise in exporting key/value pairs from Hadoop
* Where it is in the CAP Theorem
* Always available and tolerant to partitions
* View Creation and View Serving
* Creating an ElephantDB index is disassociated from serving an index

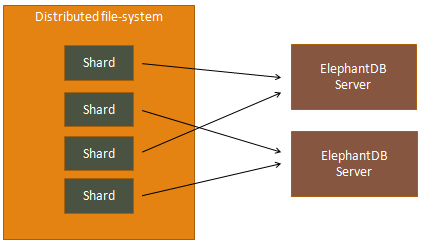
****

**View Creation**

* Occurs as a MapReduce job at the end of the batch layer
* Generated partitions/shards are stored on a DFS (HDFS Hadoop)
* Input :- JCascalog reads Hadoop batch output files and generates key/value pairs
* Number of reducers equal to the number of ElephantDB shards
* Keys are partitioned to the reducers using a sharding scheme
* In the MapReduce Job every reducer is reasonable for producing exactly one shard

**View Serving**

* Shards are served by ElephantDB servers
* Each server serves a subset of the shards
* An ElephantDB cluster is all the servers serving shards
* ElephantDB supports replication
* Replication factor of 3 with 64 shards and 8 servers equals to 24 shards per server
  + Each shards will replicate across 3 servers

****

**Redis – Overview**

**Remote dictionary Server**