


Introduction to Hadoop/HBase

Shawn Hermans
Omaha Java Users Group
March 19th, 2013

About Me

- Mathematician/Physicist turned Consultant
- Graduate Student in CS at UNO
- Current Software Engineer at Sojern

Introduction to Hadoop

What is Hadoop?

Hue

Mahout

Zookeeper

Oozie

HBase

HDFS

Whirr

An Ecosystem of Open Source Tools for Handling Big Data Problems on Commodity Hardware

Hive

MapReduce

Avro

Pig

HCatalog

Sqoop

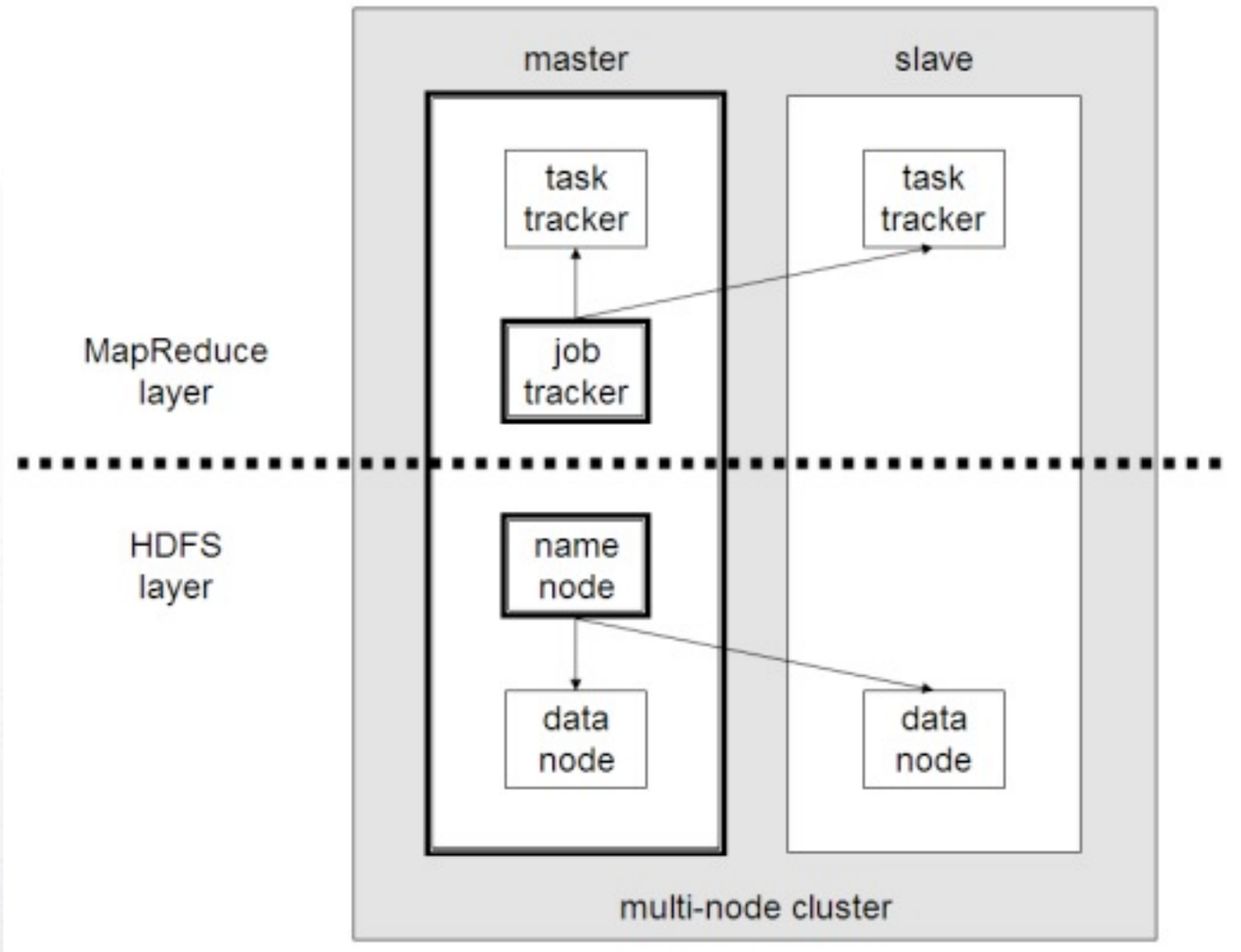
MapReduce YARN

Flume

First Impression

High Level

- HDFS - Distributed File System
- MapReduce - Scalable, Distributed Batch Processing
- HBase - Low Latency, Random Access to Large Amounts of Data



Retrieved from: http://en.wikipedia.org/wiki/Apache_Hadoop

Hadoop Use Cases

- Extract, Transform, Load
- Data Mining
- Data Warehousing
- Analytics

Hadoop is Well Suited for "Big Data" Use Cases

What is Big Data?

Data Source	Size
Wikipedia Database Dump	9GB
Open Street Map	19GB
Common Crawl	81TB
1000 Genomes	200TB
Large Hadron Collider	15PB annually

Gigabytes -

Normal size for relational databases

Terabytes -

Relational databases may start to experience scaling issues

Petabytes -

Relational databases struggle to scale without a lot of fine tuning

RDBMS vs Hadoop

Organization	Storage
EBay	1.4PB
IRS	150TB
Yahoo	2PB

Large Postgres Installs

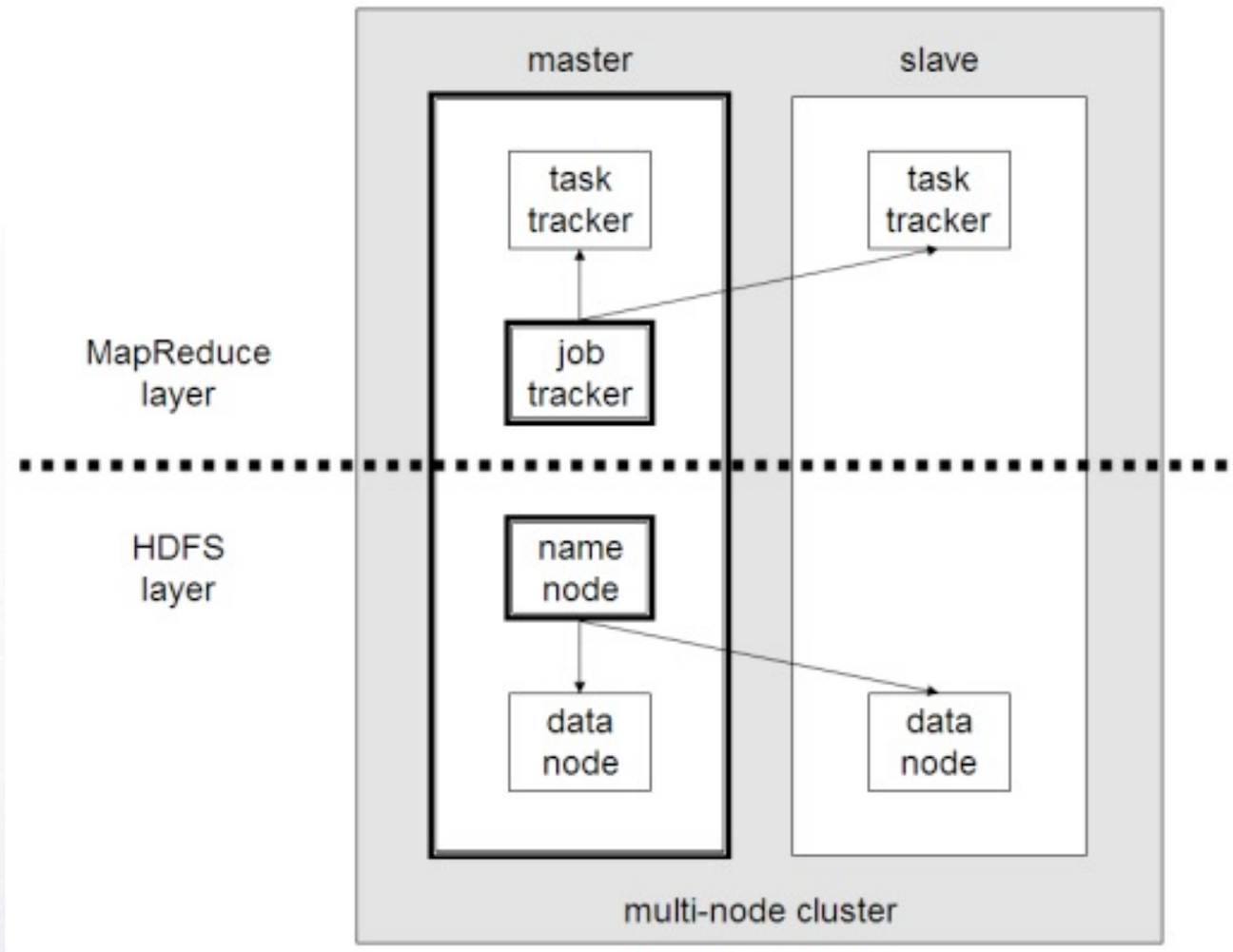
Large Hadoop Installs

Organization	Nodes	Storage
Quantcast	3,000 cores	3PB
Yahoo	4,000 nodes	70PB
Facebook	2,300+ nodes	100PB+

HDFS

HDFS

- Optimized for high throughput, not low latency
- Not POSIX compliant
- Not optimized for small files and random reads



HDFS Examples

```
hdfs dfs -mkdir hdfs://nn.example.com/tmp/foo

hdfs dfs -rmr hdfs://nn.example.com/tmp/foo

hdfs dfs -get hdfs://nn.example.com/tmp/foo /tmp/bar

 hdfs dfs -ls hdfs://nn.example.com/tmp/foo
-rw-r--r--   3 dev supergroup          0 2013-03-12 15:28 hdfs://../_SUCCESS
drwxr-xr-x   - dev supergroup          0 2013-03-12 15:27 hdfs://../_logs
-rw-r--r--   3 dev supergroup   46812001 2013-03-12 15:28 hdfs://../part-r-00000
```

Common File Formats

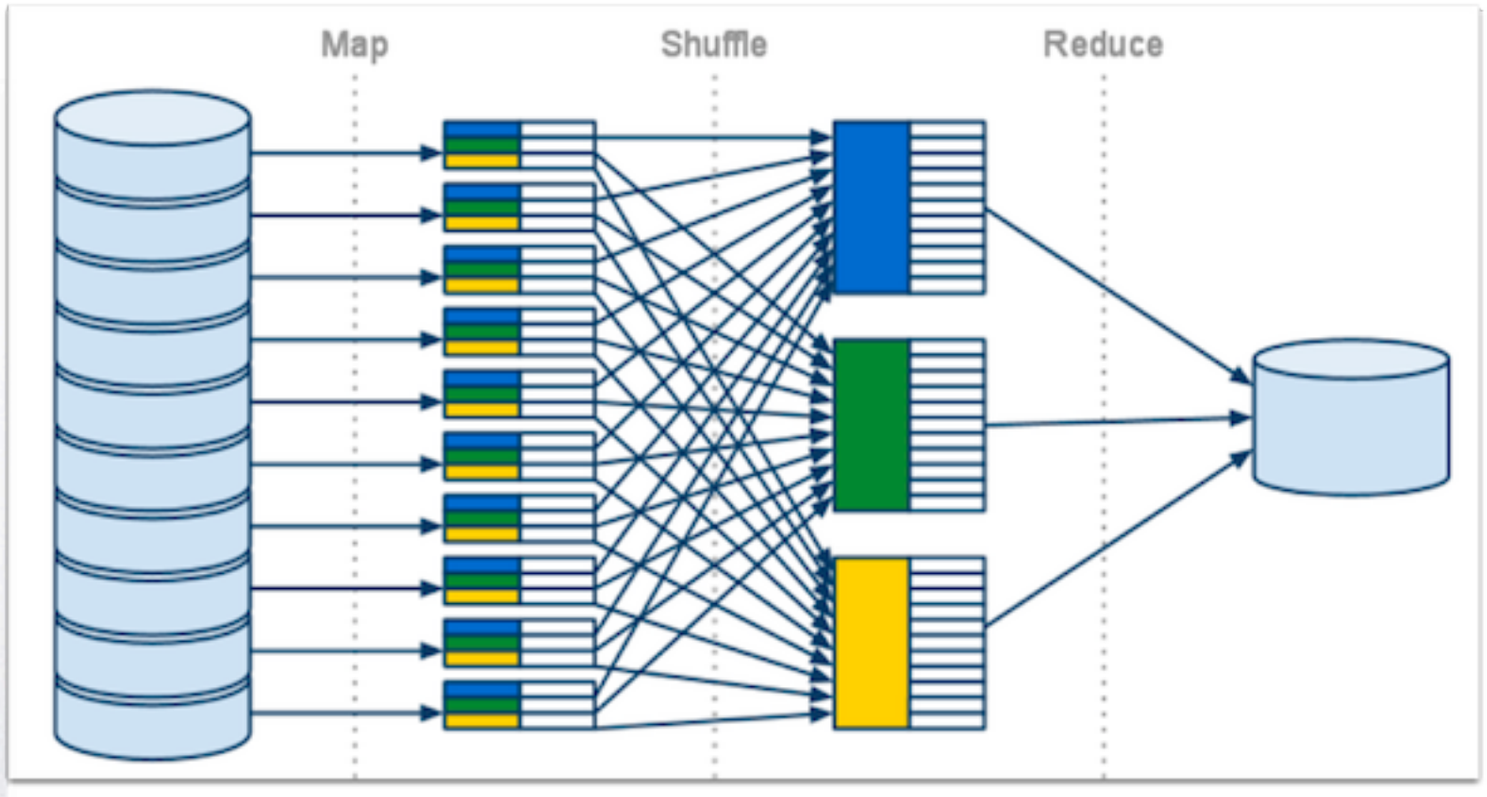
- Avro
- SequenceFile
- RCFile
- Protocol Buffers
- Thrift
- CSV/TSV

Common Compression

- Snappy
- LZO
- Gzip
- Bzip

MapReduce

MapReduce



- Map - Emit key/ value pairs from data
- Reduce - Collect data with common keys
- Tries to minimize moving data between nodes

Image taken from: https://developers.google.com/appengine/docs/python/dataprocessing/overview

Word Count

- Given a corpus of documents of documents, count the number of times each word occurs
- The "Hello World" of MapReduce


Mapper

```
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
  18      private final static IntWritable one = new IntWritable(1);
  19      private Text word = new Text();
  20
  21      public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
  22          String line = value.toString();
  23          StringTokenizer tokenizer = new StringTokenizer(line);
  24          while (tokenizer.hasMoreTokens()) {
  25              word.set(tokenizer.nextToken());
  26              context.write(word, one);
  27          }
  28      }
  29  }
```

Taken from http://wiki.apache.org/hadoop/WordCount

Reducer

```
 31  public static class Reduce extends Reducer<Text, IntWritable,
Text, IntWritable> {
 32
 33      public void reduce(Text key, Iterable<IntWritable> values,
Context context)
 34          throws IOException, InterruptedException {
 35          int sum = 0;
 36          for (IntWritable val : values) {
 37              sum += val.get();
 38          }
 39          context.write(key, new IntWritable(sum));
 40      }
 41  }
```

Taken from http://wiki.apache.org/hadoop/WordCount


Program

```
43  public static void main(String[] args) throws Exception {
44      Configuration conf = new Configuration();
45
46          Job job = new Job(conf, "wordcount");
47
48      job.setOutputKeyClass(Text.class);
49      job.setOutputValueClass(IntWritable.class);
50
51      job.setMapperClass(Map.class);
52      job.setReducerClass(Reduce.class);
53
54      job.setInputFormatClass(TextInputFormat.class);
55      job.setOutputFormatClass(TextOutputFormat.class);
56
57      FileInputFormat.addInputPath(job, new Path(args[0]));
58      FileOutputFormat.setOutputPath(job, new Path(args[1]));
59
60      job.waitForCompletion(true);
61  }
```

Taken from http://wiki.apache.org/hadoop/WordCount

Problem With MR

- Cumbersome API
- Too Low Level for Most Programs
- Requires Writing Java (Unfamiliar to many data analysts)

MapReduce Alternatives

- Hive - SQL-like query language for Hadoop
- Pig - Procedural, high-level language for creating Hadoop workflows
- Cascading - Java API for creating creating complex MapReduce workflows
- Cascalog - Clojure-based query language

Hive

```
SELECT word, COUNT(*) FROM input LATERAL VIEW
explode(split(text, ' ')) lTable as word GROUP
BY word;
```

Taken from: http://stackoverflow.com/questions/10039949/word-count-program-in-hive


Pig

```
input_lines = LOAD '/tmp/the-internet' AS (line:chararray);
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
word_groups = GROUP words BY word;
word_count = FOREACH word_groups group, GENERATE COUNT(words);
STORE ordered_word_count INTO '/tmp/words-counted';
```

Based on: http://en.wikipedia.org/wiki/Pig_(programming_tool)

Cascading

```
Scheme sourceScheme = new TextLine( new Fields( "line" ) );
Tap source = new Hfs( sourceScheme, inputPath );

Scheme sinkScheme = new TextLine( new Fields( "word", "count" ) );
Tap sink = new Hfs( sinkScheme, outputPath, SinkMode.REPLACE );
Pipe assembly = new Pipe( "wordcount" );
String regex = "(?<!\\pL)(?=\\pL)[^ ]*(?<=\\pL)(?!\\pL)";
Function function = new RegexGenerator( new Fields( "word" ), regex );
assembly = new Each( assembly, new Fields( "line" ), function );
assembly = new GroupBy( assembly, new Fields( "word" ) );
Aggregator count = new Count( new Fields( "count" ) );
assembly = new Every( assembly, count );

Properties properties = new Properties();
FlowConnector.setApplicationJarClass( properties, Main.class );
FlowConnector flowConnector = new FlowConnector( properties );
Flow flow = flowConnector.connect( "word-count", source, sink,
assembly );
flow.complete();
```

Taken from: http://docs.cascading.org/cascading/1.2/userguide/html/ch02.html

PyCascading

```
from pycascading.helpers import *

@udf_map(produces=['word'])
def split_words(tuple):
    for word in tuple.get(1).split():
        yield [word]

def main():
    flow = Flow()

    input = flow.source(Hfs(TextLine(), 'pycascading_data/town.txt'))
    output = flow.tsv_sink('pycascading_data/out')

    input | split_words | group_by('word', native.count()) | output

    flow.run(num_reducers=2)
```

From: https://github.com/twitter/pycascading/blob/master/examples/word_count.py

Cascalog

```
(defn lowercase [w] (.toLowerCase w))

(?<- (stdout) [?word ?count]
  (sentence ?s) (split ?s :> ?word1)
  (lowercase ?word1 :> ?word) (c/count ?count))
```

Taken from: http://nathanmarz.com/blog/introducing-cascalog-a-clojure-based-query-language-for-haado.html

HBase

HBase

- NoSQL
- Distributed
- Versioned
- Timestamped
- Columnar

HBase Use Cases

- Web Crawl Table
- Large Scale Web Messaging
- Sensor Data Collection
- Analytics

HBase Architecture

- Rows are split across regions
- Each region is hosted by a single RegionServer
- Reads and writes to a single row are atomic
- Row is a key/value pair container

HBase Data Model

`(rowkey, column family, column, timestamp) -> value`

- Column Families Control Physical Storage
  - TTL
  - Versions
  - Compression
  - Bloom Filters

HBase vs RDBMS

- Horizontal Scaling
- Atomic Across Single Row Only
- No JOINs
- No Indexes
- No Data Types
- Normalization = Bad in HBase

HBase vs HDFS

Batch Processing	HBase is 4 to 5 times slower than HDFS
Random Access	HBase uses techniques to reduce read/write latency
Small Records	Normal HDFS Block size is 128MB or larger
# of Records	# of HDFS files limited by NameNode RAM

Create Table

```
hbase(main):001:0>
hbase(main):001:0> create ‘table1’, {NAME => ‘f1’, VERSIONS => 1, TTL => 2592000},{NAME => ‘f2’}, {NAME => ‘f3’}
0 row(s) in 1.5700 seconds
hbase(main):002:0> describe ‘table1’ DESCRIPTION ENABLED
{NAME => ‘table1’, FAMILIES => [{NAME => ‘f1’, BLOOMFILTER => ‘NONE’, REPLICATION_SCOPE => ‘0’, VERSIONS => ‘1’, COMPRESSION => ‘N true ONE’, MIN_VERSIONS => ‘0’, TTL => ‘2592000’, BLOCKSIZE => ‘65536’, IN_MEMORY => ‘false’, BLOCKCACHE => ‘true’}, {NAME => ‘f2’, BLOOMFILTER => ‘NONE’, REPLICATION_SCOPE => ‘0’, VERSIONS => ‘3’, COMPRESSION => ‘NONE’, MIN_VERSIONS => ‘0’, TTL => ‘2147483647’, BLOCKSIZE => ‘65536’, IN_MEMORY => ‘false’, BLOCKCACHE => ‘true’}, {NAME => ‘f3’, BLOOMFILTER => ‘NONE’, REPLICATION_SCOPE => ‘0’, VERSIONS => ‘3’, COMPRESSION => ‘NONE’, MIN_VERSIONS => ‘0’, TTL => ‘2147483647’, BLOCKSIZE => ‘65536’, IN_MEMORY => ‘false’, BLOCK CACHE => ‘true’}]}
1 row(s) in 0.0170 seconds
hbase(main):003:0>
```

Get

```
Configuration conf = HBaseConfiguration.create();
HTable table = new HTable(conf, "testtable");
Get get = new Get(Bytes.toBytes("row1"));
get.addColumn(Bytes.toBytes("colfam1"),
Bytes.toBytes("qual1"));
Result result = table.get(get);
byte[] val = result.getValue(Bytes.toBytes("colfam1"),
Bytes.toBytes("qual1")); System.out.println("Value: " +
Bytes.toString(val));
```

Example from HBase: The Definitive Guide

Put

```
Configuration conf = HBaseConfiguration.create();
HTable table = new HTable(conf, "testtable");
Put put = new Put(Bytes.toBytes("row1"));
put.add(Bytes.toBytes("colfam1"), Bytes.toBytes("qual1"), Bytes.toBytes("val1"));
put.add(Bytes.toBytes("colfam1"), Bytes.toBytes("qual2"), Bytes.toBytes("val2"));
table.put(put);
```

Example from HBase: The Definitive Guide

Scan

```
Scan scan = new Scan();
scan.addColumn(Bytes.toBytes("colfam1"),
Bytes.toBytes("col-5")).
  addColumn(Bytes.toBytes("colfam2"),
Bytes.toBytes("col-33")).
  setStartRow(Bytes.toBytes("row-10")).
  setStopRow(Bytes.toBytes("row-20"));
ResultScanner scanner = table.getScanner(scan);
for (Result res : scanner) {
  System.out.println(res);
}
scanner.close();
```

Example from HBase: The Definitive Guide

HBase Schemas

- Column Families control physical storage
- Prefer short names for columns
- Use "smart" row keys
- Avoid incremental keys
- Use "smart" column names for nested relationships

Hadoop in Practice

Issues

- Still in infancy
- Complex configuration options
- Confusing versioning
- Distributed systems are inherently complex

Distributions

- Cloudera's Distribution Including Hadoop (CDH)
- MapR
- Hortonworks Data Platform (HDP)
- Amazon Elastic MapReduce (EMR)

General Tips

- Use a distribution
- Monitor your cluster
- Understand how HDFS, MapReduce, HBase, etc... works at a fundamental level

Conclusion

- Hadoop is a great tool for working with big data on "cheap" hardware
- Power comes at the cost of complexity

Twitter: @shawnhermans
Email: shawnhermans@gmail.com

Backups

facebook

Storage Infrastructure Behind Facebook Messages

HBase/HDFS/ZK/MapReduce

The New Facebook Messages

- Brings all your messages (text/chat/email) together in one place
- Full conversation history – see everything you have discussed with each friend as a single conversation
- Focuses on messages from friends
- Lightweight/less formal/no subject lines!

Why we chose HBase

- High write throughput
- Good random read performance
- Horizontal scalability
- Automatic Failover
- Strong consistency
- Benefits of HDFS
  - Fault tolerant, scalable, checksums, MapReduce
  - Internal dev & ops expertise

What do we store in HBase

- HBase
  - Small messages
  - Message metadata (thread/message indices)
  - Search index
- Haystack (our photo store)
  - Attachments
  - Large messages

Facebook Messages: Quick Stats

- 8B+ messages/day
- Traffic to HBase
  - 75+ Billion R+W ops/day
  - At peak: 1.5M ops/sec
  - ~ 55% Read vs. 45% Write ops
  - Avg write op inserts ~16 records across multiple column families.
- 2PB+ of online data in HBase (6PB+ with replication; excludes backups)
  - Message data, metadata, search index
- All data LZO compressed
- Growing at 250TB/month

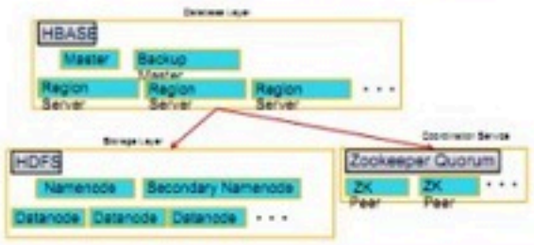
Timeline:

- Started in Dec 2009
- Roll out started in Nov 2010
- Fully rolled out by July 2011 (migrated 1B+ accounts from legacy messages!)

While in production:

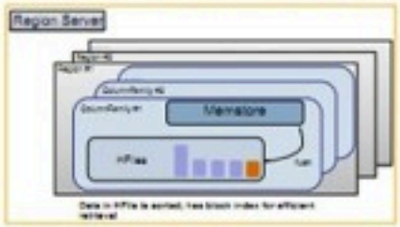
- Schema changes: not once, but twice!
- Implemented & rolled out HFile V2 and numerous other optimizations in an upward compatible manner!

HBase-HDFS System Overview



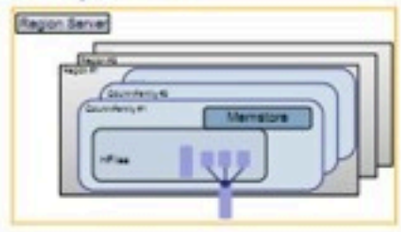
Write Path Overview

Flushes: Memstore → HFile



Read Path Overview

Compactions



Future Work

- Reliability, Availability, Scalability!
- Lot of new use cases on top of HBase in the works.
  - HDFS Namenode HA
  - Recovering gracefully from transient issues
  - Fast hot-backups
- Delta-encoding in block cache
- Replication
- Performance (HBase and HDFS)
- HBase as a service Multi-tenancy
- Features- coprocessors, secondary indices

Key Papers

Paper	Technology
The Google Filesystem	HDFS
MapReduce: Simplified Data Processing on Large Clusters	MapReduce
BigTable: A Distributed Storage System for Structured Data	HBase
Pig Latin: A Not-So-Foreign Language for Data Processing	Pig
Dremel: Interactive Analysis of Web-Scale Datasets	Impala

Hadoop Versioning