Enter the Elephant

Massively Parallel Computing With Hadoop

Toby DiPasquale
Chief Architect
Invite Media, Inc.

Philadelphia Emerging Technologies for the Enterprise
March 26, 2008

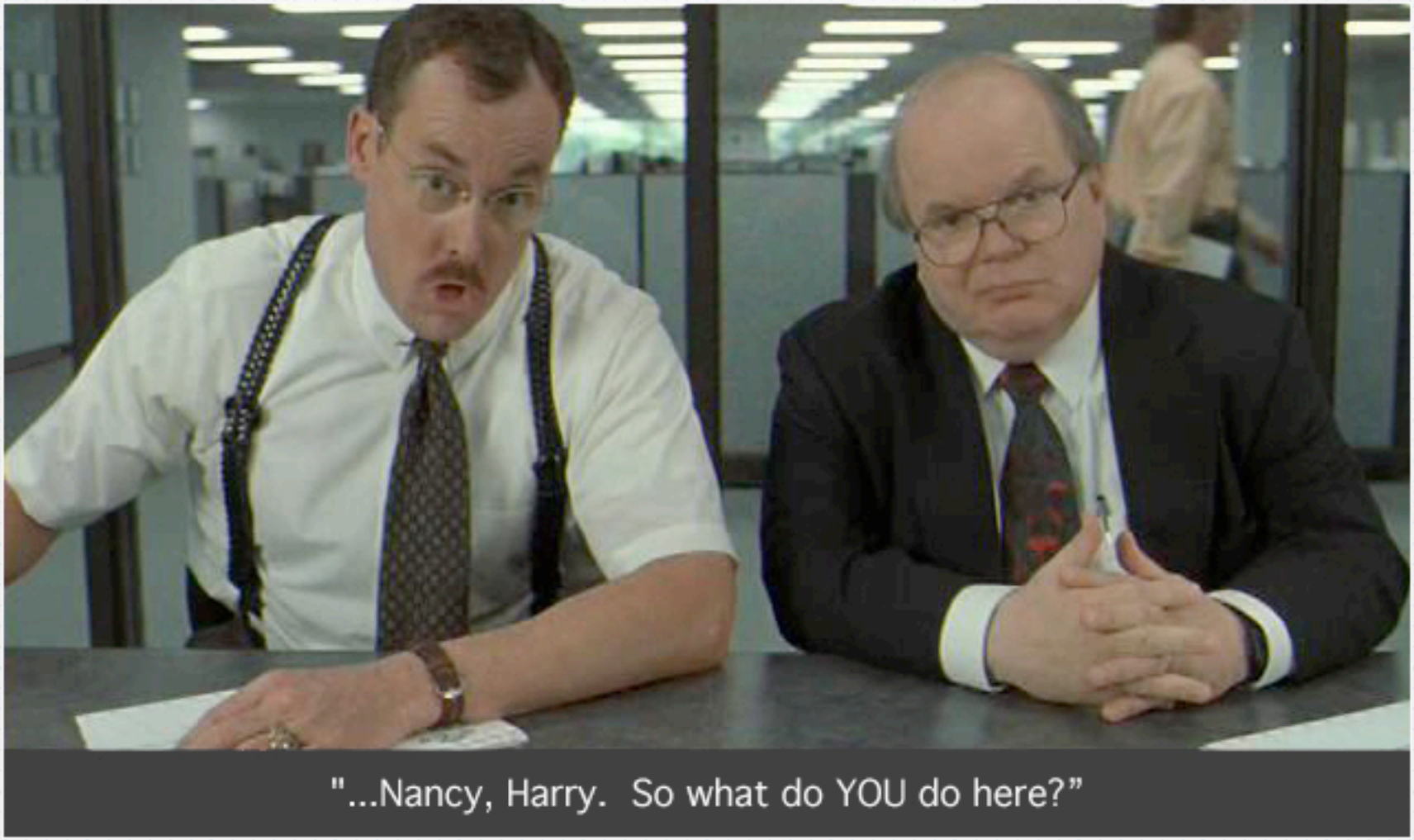

Image credit, http://www.depaulca.org/images/blog_1125071.jpg

How would you get counts of all the distinct words...

...in a file?

...in a directory?

...on the World Wide Web?

I got an idea...

Lets shove all the pages into Oracle!!!1!

Image credit, http://www.uncov.com

We need a new paradigm

What does Google do?

Google's Infrastructure

* Distributed filesystem (GFS)
* Distributed execution framework (map/reduce)
* Query language (Sawzall)
* Distributed, column-oriented datastore (Bigtable)
* Machine learning (interns)

Wait...

...I don't work for Google.

Yahoo! to the rescue...

Enter Hadoop

* Distributed filesystem (HDFS)
* Distributed execution framework (MapReduce)
* Query language (Pig)
* Distributed, column-oriented datastore (HBase)
* Machine learning (Mahout)

Hadoop Distributed Filesystem

* Cluster filing system
* Designed for huge files (many GBs)
* Designed for lots of streaming reads and infrequent writes
* Not a POSIX filesystem, requires client help

Files on HDFS

* Files consist of an ordered series of blocks and some metadata
* Block data is distributed across DataNode machines
* NameNode maintains filesystem metadata and list of blocks

Blocks

- Blocks are replicated)=3 times (admin configurable)
- Blocks are regular files on DataNode machines*
- Default block size is 64MB

(*) Or Amazon S3 objects, if you'd prefer

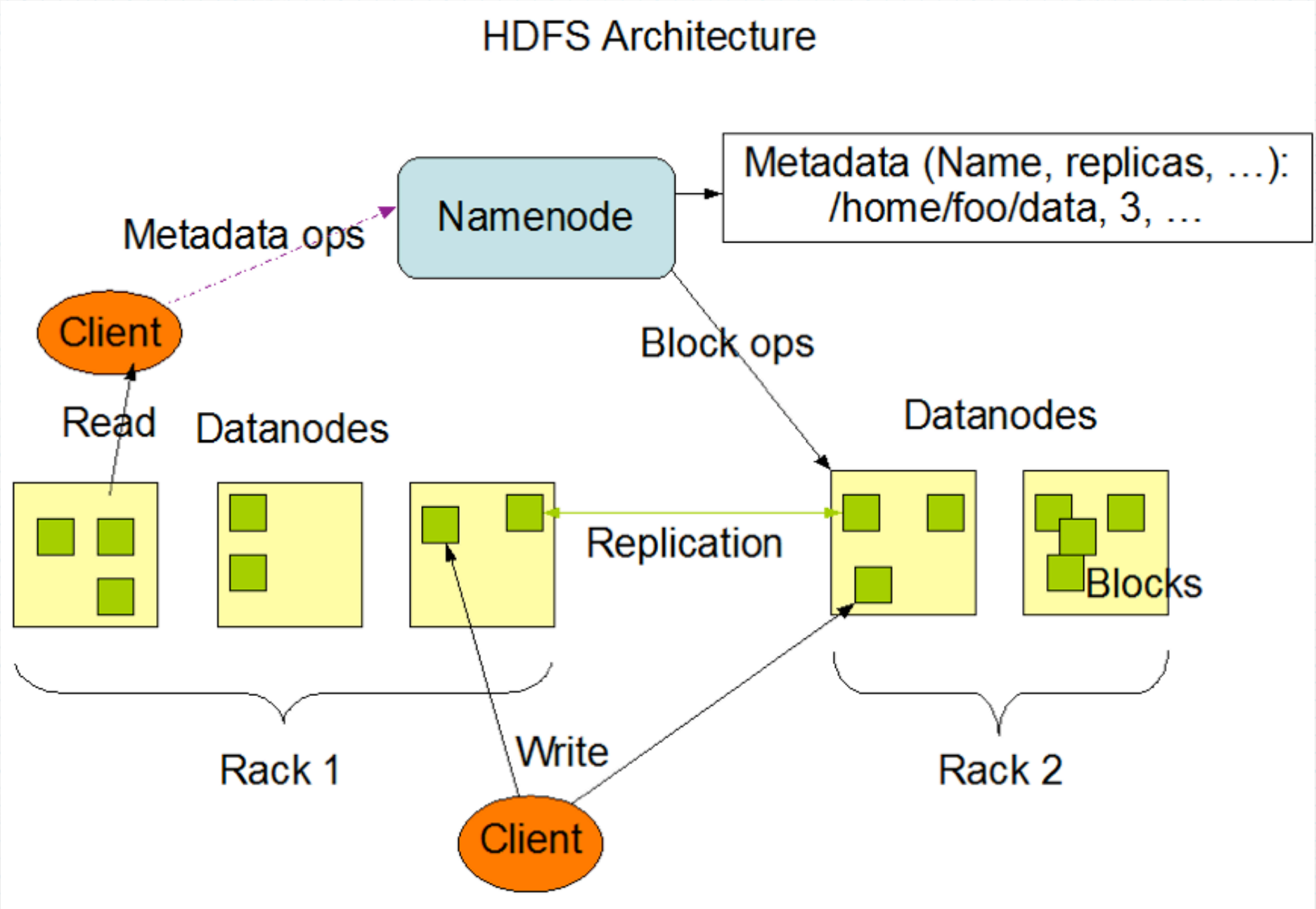

Image credit, http://hadoop.apache.org/core/docs/current/hdfs_design.html

Keep these blocks in mind...
they become important later.

Wait...

...what am I supposed to do
with all those huge files?

Image credit, http://icanhazcheezburger.com

MapReduce

* MapReduce serves that need
* Constrained programming model
* Data parallel
* Lifted from functional programming

Phases of MapReduce

* Initialization
* Map
* Shuffle
* Sort
* Reduce

Initialization

* Mappers and Reducers are allocated
* Code is shipped to nodes
* Mappers and Reducers are run on same machines as DataNodes

Map

* Mapper takes input key/value pair
* Does something to its input
* Emits intermediate key/value pair
* One call per input record
* Fully data-parallel

Shuffle

* Intermediate data from Mapper is copied to Reducer machines
* All data for a partition goes to same Reducer
* Triggered when a Mapper completes

Sort

* Sort intermediate data by key
* All records for key are then contiguous
* Can't start until all Mappers are finished

Reduce

* Input is all list of intermediate values for a given key
* Reducer aggregates list of intermediate values
* Yields a final key/value pair for output

Partitioning

* Intermediate keys are partitioned by user-defined function
* Used to route intermediate data to particular Reducer instances during Shuffle
* Clever workaround for non-data-parallel nature of reduce()

Mayhap an example...

WordCount Input

“Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum...”

WordCount Mapper

```
public static class Map extends MapReduceBase implements Mapper {
    public void map(WritableComparable key,
                    Writable value,
                    OutputCollector output,
                    Reporter reporter) throws IOException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);

        while (tokenizer.hasMoreTokens()) {
            Text word = new Text(tokenizer.nextToken());
            output.collect(word, new IntWritable(1));
        }
    }
}
```

WordCount Intermediate

```
(in, 1)
(in, 1)
(sunt, 1)
(in, 1)
(elit, 1)
(sed, 1)
(eiusmod, 1)
(dolore, 1)
(enim, 1)
(eu, 1)
(dolore, 1)
(et, 1)
(labore, 1)
[...]
(adipisicing, 1)
(incididunt, 1)
(reprehenderit, 1)
```

WordCount Reducer

```
public static class Reduce extends MapReduceBase implements Reducer {
    public void reduce(WritableComparable key,
                       Iterator values,
                       OutputCollector output,
                       Reporter reporter) throws IOException {
        int sum = 0;

        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```

WordCount Final

```
(ad, 1)
(irure, 1)
(in, 3)
(ea, 1)
(officia, 1)
(sunt, 1)
(elit, 1)
(sed, 1)
(eiusmod, 1)
(enim, 1)
(eu, 1)
[...]
(aute, 1)
(Duis, 1)
(dolore, 2)
(mollit, 1)
```

Applications of MapReduce

* Log processing
* Web search engines
* Data mining
* Machine learning
* Scientific retrieval and processing
* Lots more you can think of

Downsides to M/R

* Map and Reduce work best when side effect-free
* Requires programming to query datasets
* Will have to write multiple M/R jobs to do more complicated processing

Wait...

...that seems like an awful lot of coding.

Solution, DSL

Pig Latin

* DSL for data processing on Hadoop
* Compiles down to MapReduce jobs
* Somewhat similar to SQL (relational)
* Supports doing ad-hoc queries
* Doesn't require (as much) programmer time for analysis and processing

WordCount in Pig

```
A = LOAD ‘input’ USING TextLoader();
B = FOREACH A GENERATE FLATTEN(TOKENIZE
(*));
C = GROUP B by $0;
D = FOREACH C GENERATE group, COUNT(B);
```

GROUP output

```
...
(do, {(do)})
(dolor, {(dolor), (dolor)})
(dolore, {(dolore), (dolore)})
(ea, {(ea)})
(eiusmod, {(eiusmod)})
(elit, {(elit)})
(enim, {(enim)})
(esse, {(esse)})
...
```

Output

```
(ad, 1)
(irure, 1)
(in, 3)
(ea, 1)
(officia, 1)
(sunt, 1)
(elit, 1)
(sed, 1)
(eiusmod, 1)
(enim, 1)
(eu, 1)
[...]
(aute, 1)
(Duis, 1)
(dolore, 2)
(mollit, 1)
```

Pig Features

* Supports equi-join and inner join
* Operators, FILTER, FOREACH, GROUP
* Binary operators, COGROUP, CROSS, UNION
* Loads TSV and plain text

Downsides to Pig

- Pretty young
  - Still in incubator stage as Apache project
  - Subject to big changes
- Light on built-in functionality

Wait...

...I still need to query stuff fast sometimes.

Geez, you guys want everything

HBase

* Built on top of HDFS
* No schemas (yay!)
* Column-oriented datastore
  * Allows arbitrary columns per row
  * No space penalty for NULL columns

Conceptual View

Row Key	Time Stamp	Column *"contents:"*	Column *"anchor:"*		Column *"mime:"*
"com.cnn.www"	t9		"anchor:cnnsi.com"	"CNN"	
	t8		"anchor:my.look.ca"	"CNN.com"	
	t6	"<html>..."			"text/html"
	t5	"<html>..."			
	t3	"<html>..."			

Image credit, http://wiki.apache.org/hadoop/Hbase/HbaseArchitecture

Physical View

Row Key	Time Stamp	Column *"contents:"*
"com.cnn.www"	t6	"<html>..."
	t5	"<html>..."
	t3	"<html>..."

Row Key	Time Stamp	Column *"anchor:"*	
"com.cnn.www"	t9	"anchor:cnnsi.com"	"CNN"
	t8	"anchor:my.look.ca"	"CNN.com"

Row Key	Time Stamp	Column *"mime:"*
"com.cnn.www"	t6	"text/html"

Image credit, http://wiki.apache.org/hadoop/Hbase/HbaseArchitecture

HBase API

* Use HBaseAdmin to manipulate tables
* Use HTable to manipulate table data
* Use HScannerInterface to scan a table

Downsides of HBase

* Not as feature-packed as traditional RDBMS
* Known to have been unstable as of yet
  * Powerset is working hard on this one
* Michael Stonebraker doesn't want you to use it

Wait...

...I don't have the skIIIz to code that fancy machine learning stuff.


Cheez
Yor doin it wrong.

Mahout

* Project to implement machine learning for MapReduce
* Statistical and machine learning tools
* Powered by grad students on the Intertron and some Yahoo! people

Mahout Goals

* High-performance, distributed matrix (both sparse and dense)
* Clustering (Canopy, K-Means, Mean Shift, etc) with distancing (Manhattan, Pearson, Tanimoto, etc)
* Naive Bayes classification and Bayesian network
* Backpropogation (Neural Network)
* Expectation Maximization (e.g. Probabilistic Latent Semantic Indexing)
* Locally-Weighted Linear Regression (LWLR) and logistic regression
* Support Vector Machine
* Gaussian Discriminant Analysis
* Singular Value Decomposition, Principal Components Analysis, Independent Component Analysis

Downsides of Mahout

* Super-ultra-mega new
  * Only some clustering algorithms even implemented at this time
* Still some confusion as to who's doing what

Hadoop Cons

* Hadoop doesn't play well with existing tools
* Latency is high; real-time needs beware
* Your DBAs will suck at it in the beginning

Hadoop Pros

* Process large data very efficiently
* Very flexible
* Shared-nothing scalability
* Simple API and model
* Hadoop and Amazon EC2 fit together like chocolate and peanut butter

Bottom Line,

If you have large data, you should be looking hard at Hadoop.

Image credit, http://icanhazcheezburger.com

Links

* http://hadoop.apache.org/core/
* http://wiki.apache.org/pig/
* http://hadoop.apache.org/hbase/
* http://lucene.apache.org/mahout/