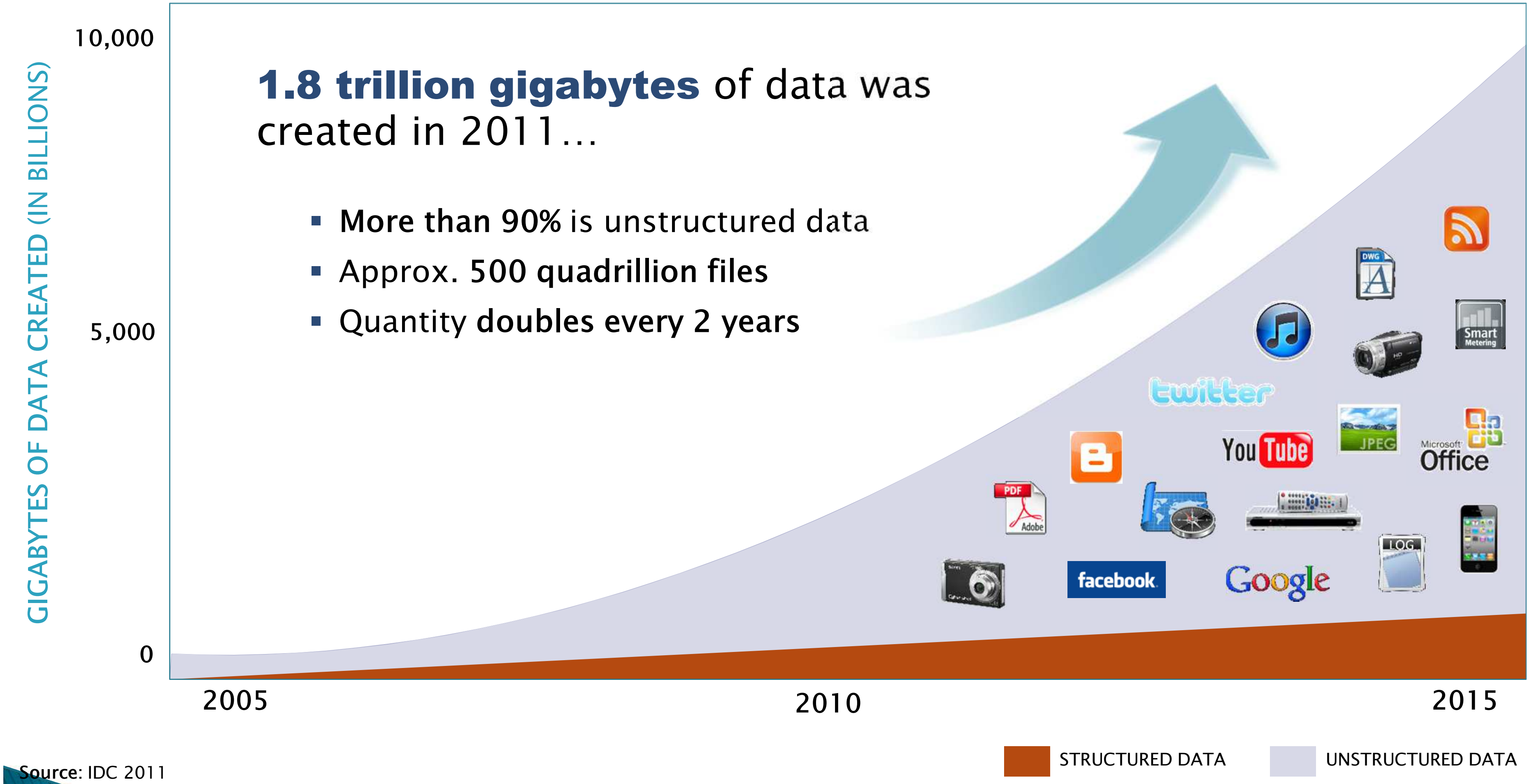
Intro to Big Data and Hadoop

@EvaAndreasson

Sr Product Manager, Cloudera

Big Data – Explosive Data Growth



1.8 trillion gigabytes of data was created in 2011…

- **More than 90%** is unstructured data
- Approx. **500 quadrillion files**
- Quantity **doubles every 2 years**

Source: IDC 2011

What is this Big Data thing?

- Simple Definition:

  Data that exceeds processing capacity of conventional data management systems.

  The data is either too big, expands too fast, or doesn't easily fit the structures of current data models.

- Large industry leaders (and smart startups!) currently looking to find ways to:

Cost-efficiently make ALL your data work for you – enable better decisions and create new business growth!

10x = 10x

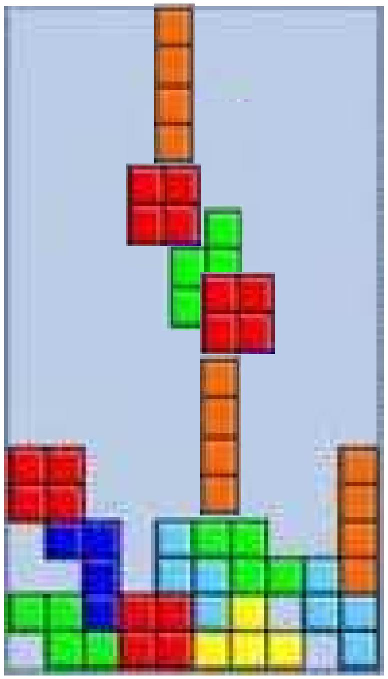
cloudera

Key Challenge #1: Volume

- “Return On Bytes”
  - How to cost efficiently query, manage, and store 100s TB or PB of data?
- Pre-mature data death
  - Off-disk and archived data difficult and costly to access

Key Challenge #2: Velocity

- Enough time to process raw data before you need it
  - Data ingest from sensors, cameras, feeds, streaming, logs, user interactions…
  - Raw data structuring for various ETL and DB models



Key Challenge #3: Variety

- Costly adaption to new data types
  - Saving account info, images, videos, url clicks, logs, and transactional data – together?
- Inflexible data models
  - Major surgery for future queries
    - Most data is modeled for questions we know will be asked…
    - Raw data value loss

Solution?

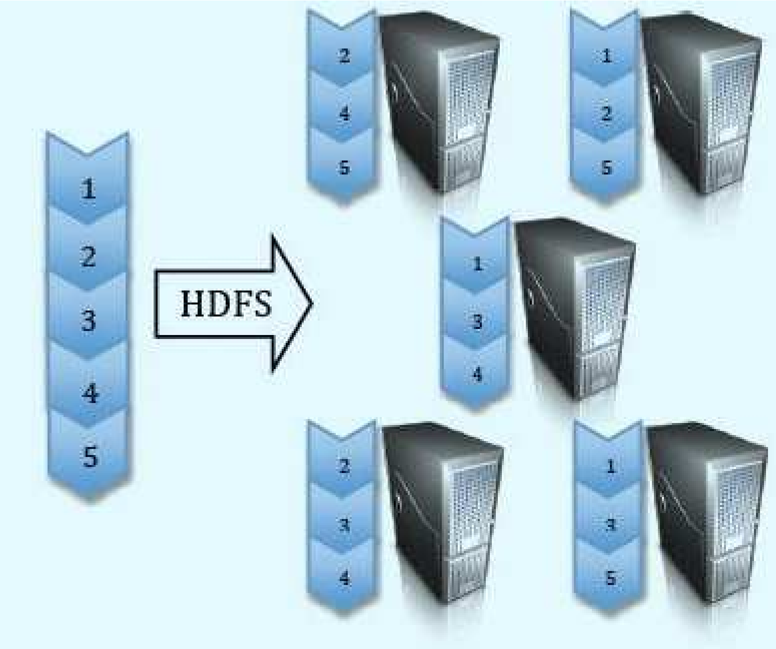
- A cost-effective, highly scalable, and flexible data processing framework
  - Distributed storage over cheap commodity servers
  - Linear distributed scale, bring processing closer to the data
  - Ability to manage and analyze unstructured data

Solution: Hadoop!

- An integrated data storage and computation framework
  - Distributed over cheap commodity servers
  - Linear scale, bringing processing closer to data
  - Ability to manage and analyze unstructured data

Hadoop – Part One: HDFS

- HDFS – Hadoop Distributed File System
  - Splits data into equally sized blocks of bytes
  - Replicated across many machines
- Enables
  - Parallelism
  - Balanced execution time
  - Robustness



Hadoop – Part Two: MapReduce

- Distributed data processing framework
  - Process data where it’s stored – without schema!
- Three phased algorithm
  - Map
    - Find relevant data by key mapping
    - Gather the value of that data
    - Create output <key, value> pair file
  - Shuffle
    - Sort the <key, value> pair file
    - Get all the <key, value> pairs (or ranges thereof) to a reducer
  - Reduce
    - For all <key, value> pairs for a certain key, process all of the values
- Final Output
    - A sorted <key, processed value> pair file

Simple Example, Yet Difficult

- **Word count is challenging over massive amounts of data**
  - Single compute node too time-consuming
  - Distributed nodes require moving data
  - Number of unique words can easily exceed the RAM
  - Would need a hash table on disk
  - Would need to partition the results (sort and shuffle)
- **Fundamentals of statistics often are simple aggregate functions**
  - Most aggregation functions have distributive nature, e.g., max, min, sum, count
- **MapReduce breaks complex tasks down into smaller elements which can be executed in parallel**

Example: Word Count

- Count words across multiple files

```
$ cat file01
Hello World Bye World
$ cat file02
Hello Hadoop Goodbye Hadoop
```

→ For simplicity, assume each file's content ends up in a separate split

Finally Some Code!

```
[imports…]

public class WordCount {

public static class Map extends MapReduceBase implements Mapper<LongWritable,
Text, Text, IntWritable> {

private final static IntWritable one = new IntWritable(1);
private Text word = new Text();

public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable>
output, Reporter reporter) throws IOException {
                String line = value.toString();
                StringTokenizer tokenizer = new StringTokenizer(line);

                while (tokenizer.hasMoreTokens()) {
                                word.set(tokenizer.nextToken());
                                output.collect(word, one);
}}}…..
```

Finally Some Code!

```
[imports…]

public class WordCount {

public static class                                        able,
Text, Text, IntWrit

private final static
private Text word

public void map(L                                          le>
output, Reporter reporter) throws IOException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);

        while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                output.collect(word, one);
}}}.....
```

```
Map(input_key, input_value)
        foreach word w in input_value:
                emit(w, 1)
```

Results of the Map Phase

- Map task 1 emits:
  < Hello, 1>
  < World, 1>
  < Bye, 1>
  < World, 1>

- Map task 2 emits:
  < Hello, 1>
  < Hadoop, 1>
  < Goodbye, 1>
  < Hadoop, 1>

Finally Some Code!

```
…
public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text,
IntWritable> {
          public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable>
output, Reporter reporter) throws IOException {
          int sum = 0;
          while (values.hasNext()) {
                    sum += values.next().get();
          }
          output.collect(key, new IntWritable(sum));
}}

public static void main(String[] args) throws Exception {
          JobConf conf = new JobConf(WordCount.class);
          conf.setJobName("wordcount");
          conf.setOutputKeyClass(Text.class);
          conf.setOutputValueClass(IntWritable.class);
          conf.setMapperClass(Map.class);
          conf.setCombinerClass(Reduce.class);
          conf.setReducerClass(Reduce.class);
          conf.setInputFormat(TextInputFormat.class);
          conf.setOutputFormat(TextOutputFormat.class);
          FileInputFormat.setInputPaths(conf, new Path(args[0]));
          FileOutputFormat.setOutputPath(conf, new Path(args[1]));
          JobClient.runJob(conf);
}}
```

Finally Some Code!

```
…
public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text,
IntWritable> {
                public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable>
output, Reporter reporter) throws IOException {
                int
                wh

                }
                ou
}}

public static vo
                Job
                co
                co
                co
                co
                co
                conf.setReducerClass(Reduce.class);
                conf.setInputFormat(TextInputFormat.class);
                conf.setOutputFormat(TextOutputFormat.class);
                FileInputFormat.setInputPaths(conf, new Path(args[0]));
                FileOutputFormat.setOutputPath(conf, new Path(args[1]));
                JobClient.runJob(conf);
}}
```

```
reduce(output_key, intermediate_vals)
        set count = 0
        foreach v in intermediate_vals:
                count += v
                emit(output_key, count)
```

Results of the Reducer

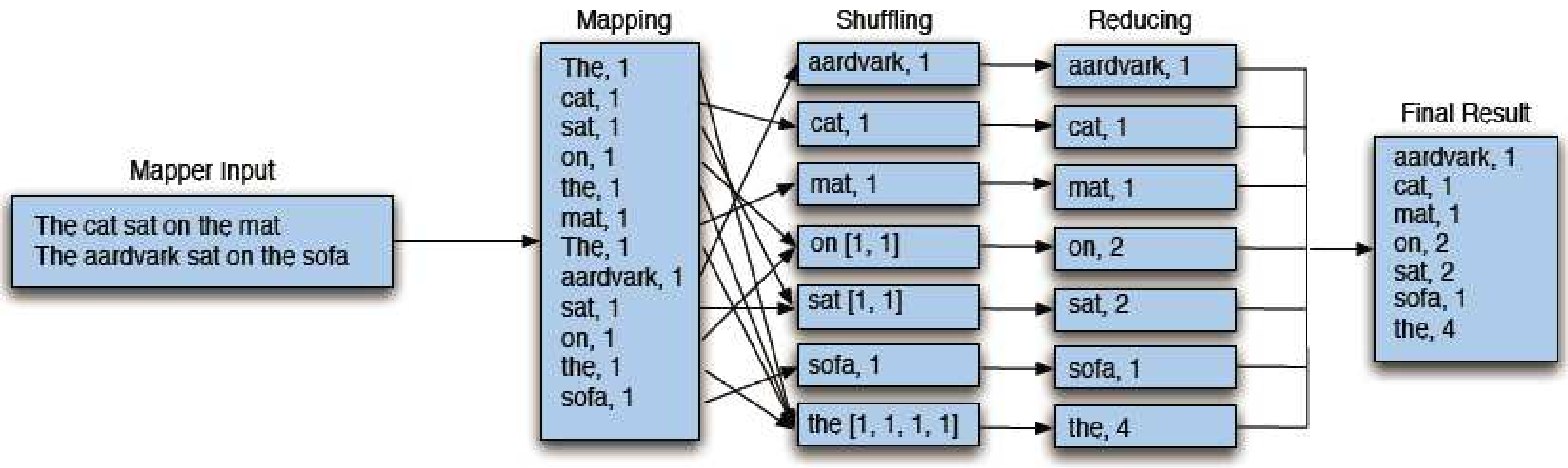
- Map task 1 emits:
  < Hello, 1>
  < World, 1>
  < Bye, 1>
  < World, 1>

- Map task 2 emits:
  < Hello, 1>
  < Hadoop, 1>
  < Goodbye, 1>
  < Hadoop, 1>

- Reducer emits:
  < Bye, 1>
  < Goodbye, 1>
  < Hadoop, 2>
  < Hello, 2>
  < World, 2>

Putting it all Together

The overall word count process



Mapper Input

The cat sat on the mat
The aardvark sat on the sofa

Mapping

The, 1
cat, 1
sat, 1
on, 1
the, 1
mat, 1
The, 1
aardvark, 1
sat, 1
on, 1
the, 1
sofa, 1

Shuffling

aardvark, 1
cat, 1
mat, 1
on [1, 1]
sat [1, 1]
sofa, 1
the [1, 1, 1, 1]

Reducing

aardvark, 1
cat, 1
mat, 1
on, 2
sat, 2
sofa, 1
the, 4

Final Result

aardvark, 1
cat, 1
mat, 1
on, 2
sat, 2
sofa, 1
the, 4

Real World Use Cases

- Data processing
  - Magnitudes faster file parsing
  - How many trades over the last 24 months finished within 5 seconds
  - What users have been less active this month
- Data analytics and Pattern matching / Machine learning
  - Match chemical compounds against huge research data to find dangerous combinations of multiple medicines
  - Pattern detection over huge data sets to find previously identified “unrelated events” that with enough input can be identified as malicious
  - Compared to my neighborhood, how much power is a specific user using
  - Based on real time images, analyzing best rescue paths in hazard zones
  - Based on real time images, analyze what days how many cars were parked outside a certain store
- Personalization, user experience optimization
  - Replicate successful sales experiences, optimize based on patterns
  - What items are most popular during what time of the day?
  - More optimized recommendations and ad-placement
- Phone home
  - Predict when HW components and mechanic components will need to be replaced, improve customer service
- ….and many more!!!!

cloudera

Interested in More?

- Learn
  - Videos, books, training: http://university.cloudera.com/
  - Blog: http://www.cloudera.com/blog/
- Download'n'play
  - www.cloudera.com/download
- Expertise input
  - Join cdh-user@cloudera.org
- Contribute to the community
  - hadoop.apache.org
- Contact me:
  - @EvaAndreasson
  - eva@cloudera.com

Thank you!!

TO THINK ABOUT:

Does querying
huge raw data sets
win over
advanced algorithms
applied to limited data?

Extra Slides

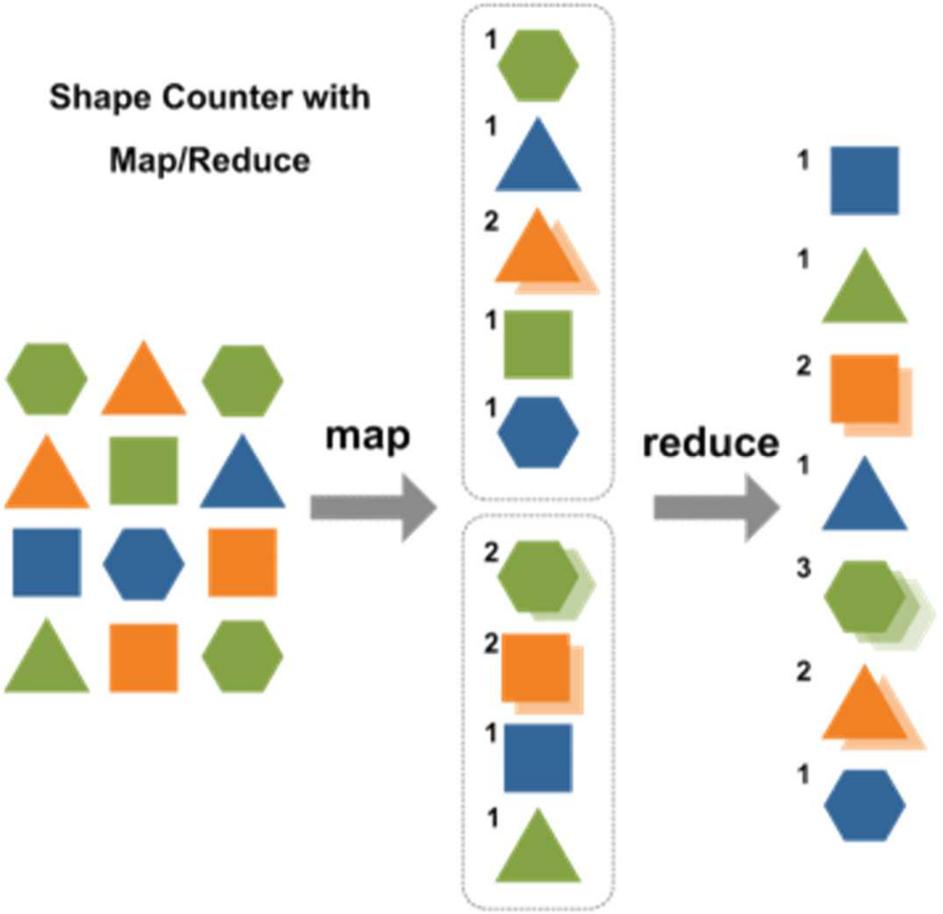
- 1) Command line view – how to start Hadoop
- 2) Visual example of MapReduce
- 3) Combiner example
- 4) More than just a framework

Run WordCount with Hadoop

```
//Run the word count application:
        $ bin/hadoop jar /usr/joe/wordcount.jar
        org.myorg.WordCount /usr/joe/wordcount/input
        /usr/joe/wordcount/output

//Results:
        $ bin/hadoop dfs -cat /usr/joe/wordcount/output/part-
        00000
        Bye 1
        Goodbye 1
        Hadoop 2
        Hello 2
        World 2
```

MapReduce – Visual Example



If Using a Combiner

- Map task 1 emits:
  < Hello, 1>
  < World, 1>
  < Bye, 1>
  < World, 1>

- Map task 2 emits:
  < Hello, 1>
  < Hadoop, 1>
  < Goodbye, 1>
  < Hadoop, 1>

- Combiner 1 emits:
  < Bye, 1>
  < Hello, 1>
  < World, 2>

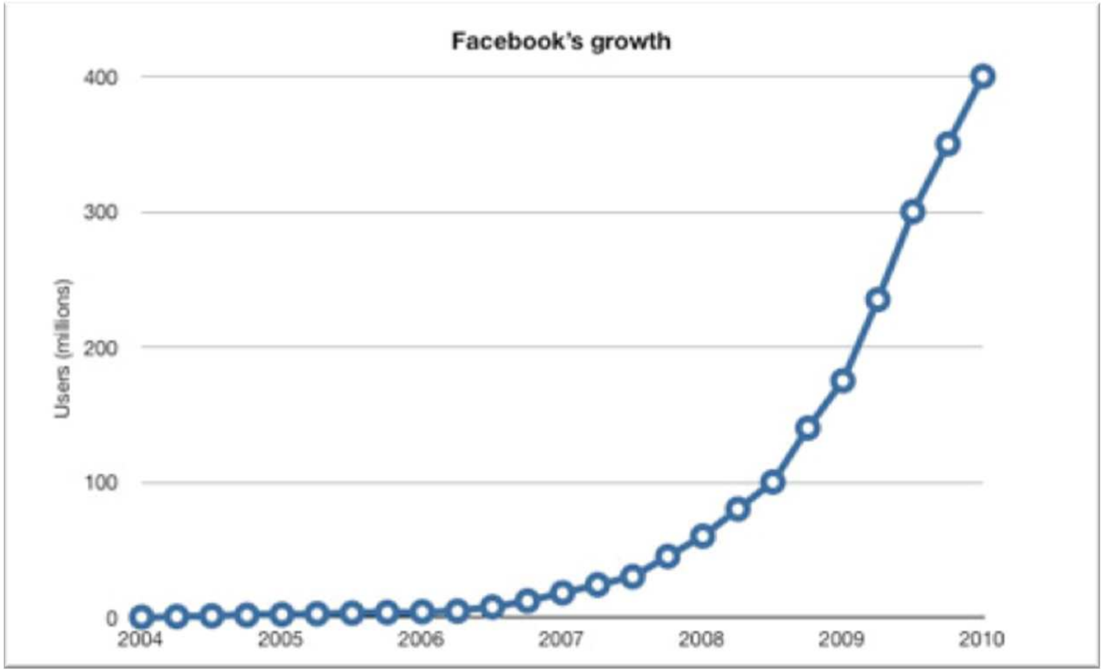
- Combiner 2 emits:
  < Goodbye, 1>
  < Hadoop, 2>
  < Hello, 1>

- Reducer emits:
  < Bye, 1>
  < Goodbye, 1>
  < Hadoop, 2>
  < Hello, 2>
  < World, 2>

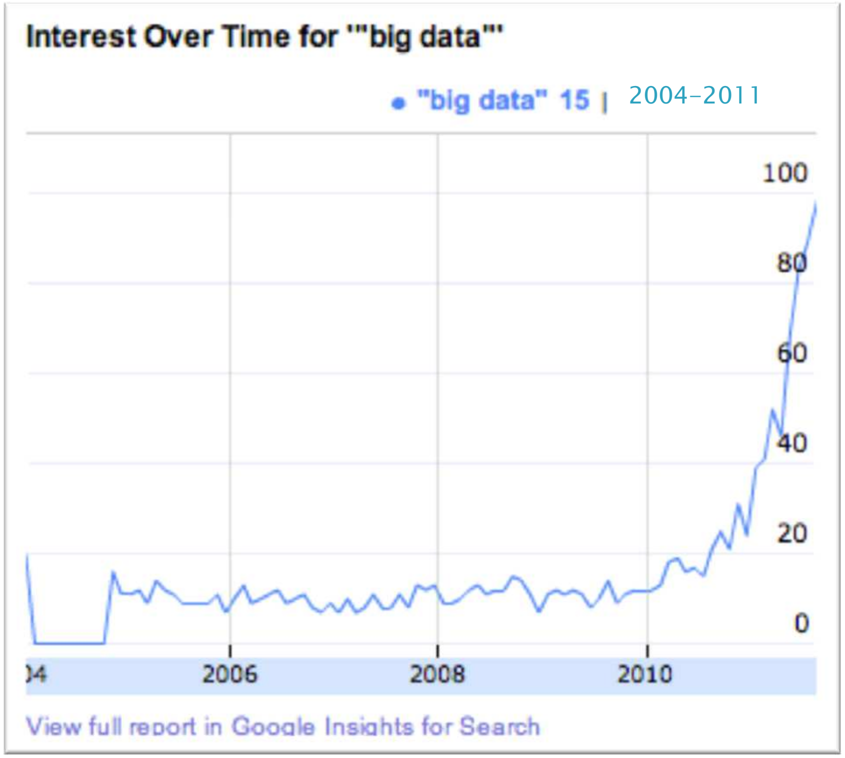
More than just a framework…

- Hadoop includes many sub-projects:
  - Column oriented database (Hbase)
  - Workflow scheduling (Oozie)
  - Import/export of data (Flume, Sqoop)
  - Task creation and tracking (Hue)
  - SQL interfaces (Hive, Pig)
  - Service and configuration management (Zookeeper)
  - Extra functionality (Whirr, Mahout, Avro)

Other Indications that this is Real

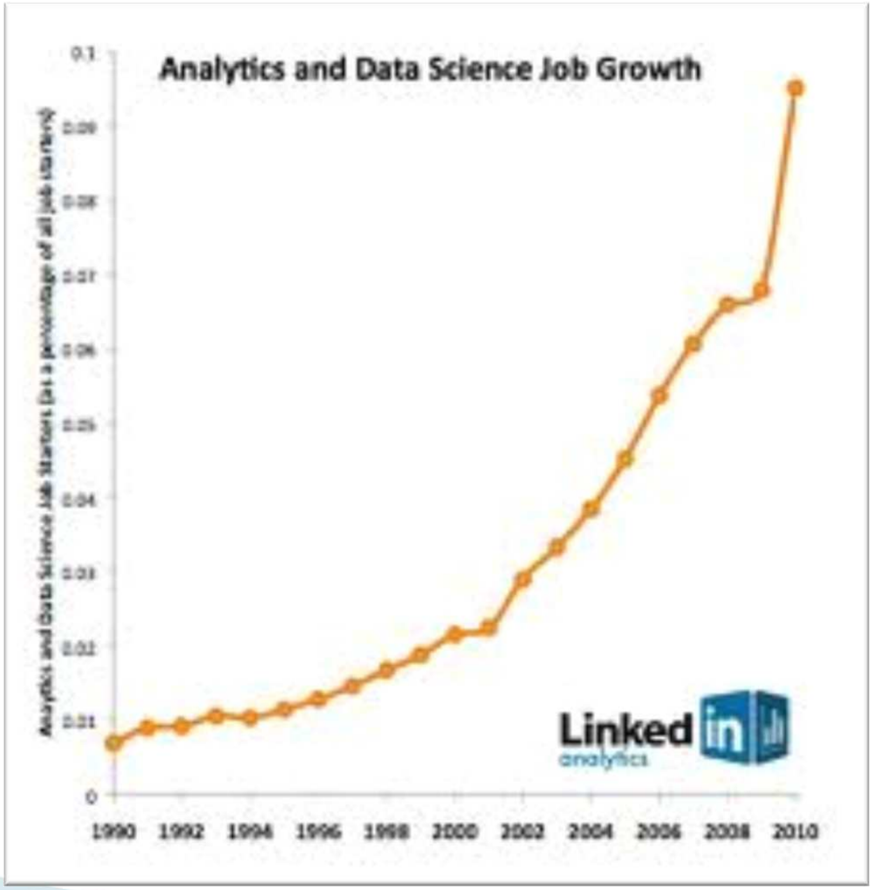


Other Indications that this is Real

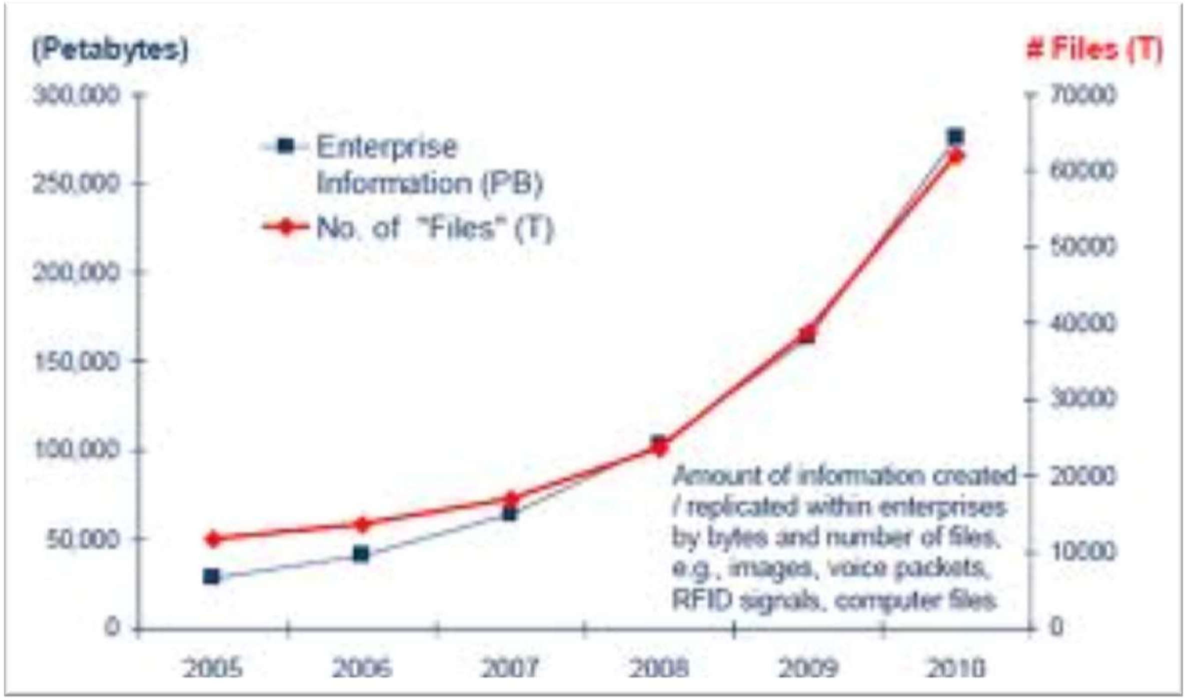


2004-2011

Other Indications that this is Real



Other Indications that this is Real



Enterprise Content Trends: smart content and big dataContent Geeks
*www.contentgeeks.net/2011/10/16/enterprise-content-trends-2/*Cached
You +1'd this publicly. Undo
Oct 16, 2011 – Two main trends in Enterprise Content Management: Content is getting smarter & bigger!