

cloudera

Introduction to Apache Hadoop



Introduction

- **Brock Noland**
- brock@cloudera.com
- **Twitter: @brocknoland**
- **Instructor at Cloudera**
- **Formerly a Lead Software Engineer at Thomson Reuters**
- **I love log files**
- **Implemented log processing using Hadoop at TR**
- <http://tch.ug>

Introduction to Apache Hadoop



What Is Hadoop

How MapReduce works

A Useful MapReduce Solution

An Easier Solution: Hive/Pig

Conclusion

What is Hadoop?

- **Scalable, fault-tolerant, distributed data storage and processing system**
- **Collection of software packages for large scale data processing**
 - MapReduce
 - Hadoop Distributed File System (HDFS)
 - HBase
 - Hive
 - Pig
 - Sqoop
 - ...

When is Hadoop appropriate?

- **Defining database schema would result in losing data**
 - Semi-structured
 - Non-homogenous but fully structured
- **Expense of traditional storage causes data loss**
- **Your organization has a “big data” problem**
- **Petabytes not needed**
 - If a database query needs to scan a TB-sized table, it will be very slow and remove the "hot" data from the caches
 - I personally saw enormous benefit when processing hundreds of GB of log files

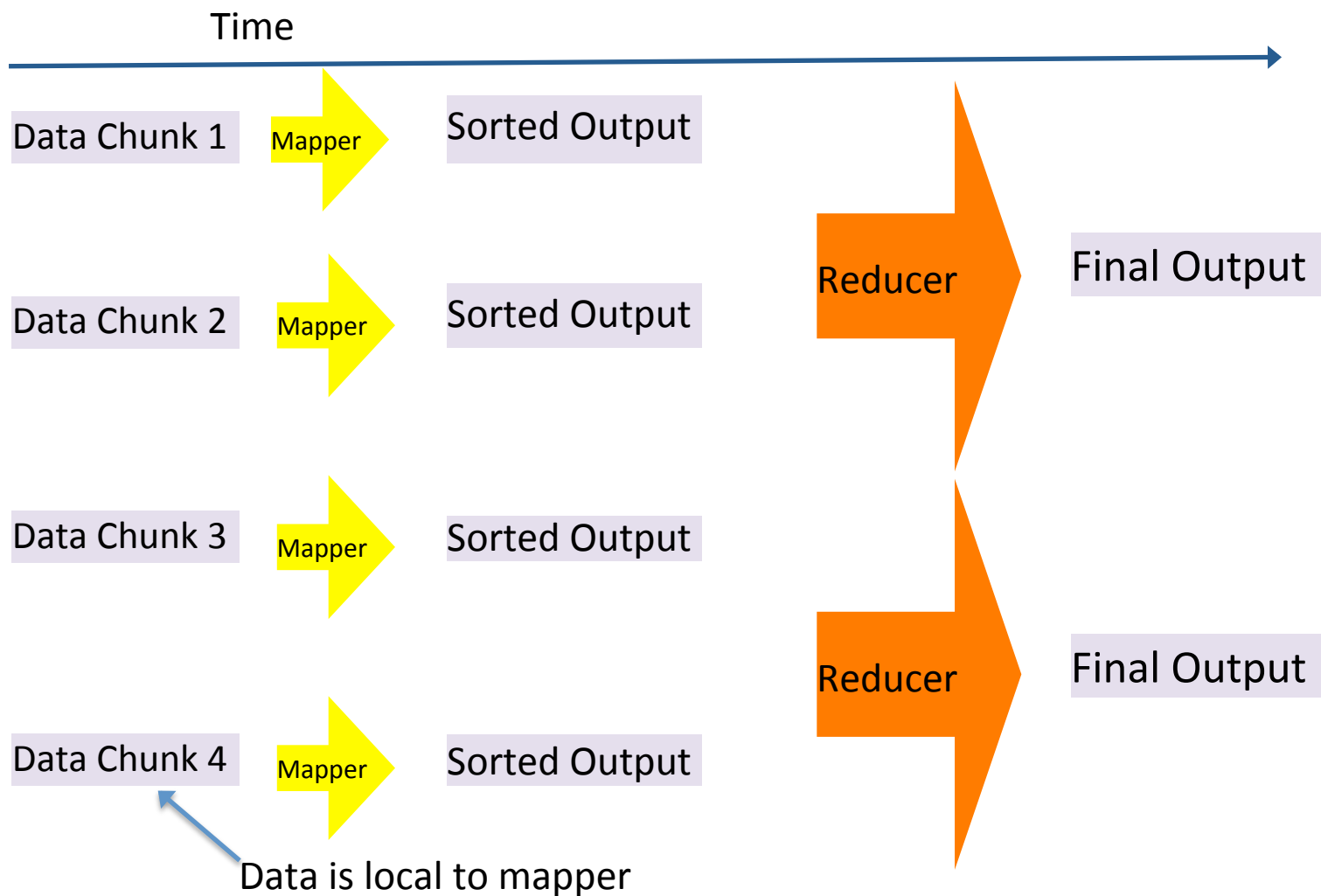
How does Hadoop fit into my existing environment?

- **Is Hadoop going to replace the RDBMS?**
 - Hadoop is for large data (structured or unstructured)
 - Hadoop is for problems databases are not good at
 - Hadoop is a new tool, not a simple replacement for other tools
 - HBase may overlap the RDBMS space for specific use cases
 - Storing a few billion key value pairs
- **Use Cases**
 - Log processing
 - Extract Transform Load (ETL)
 - Document clustering
 - Classification
 - XML transformations
 - “Low latency” tape cache
 - Indexing

What is MapReduce?

- **A distributed data processing metaphor originated by Google**
- **Hadoop MapReduce is an open source implementation of that metaphor**

MapReduce in One Slide



Benefits of MapReduce

- **Capable of processing vast amounts of data**
- **Scales linearly**
 - Same data problem will process 10x faster on 10x larger cluster
- **Individual failures have minimal impact**
 - Failures during a job cause only a small portion of the job to re-executed

Drawbacks of MapReduce

- **Job setup takes time (e.g., several seconds)**
 - MapReduce is not for real-time interaction
- **Requires deep understanding of the MapReduce paradigm**
- **Not all problems are easily expressed in MapReduce**

What problems does Hadoop solve?

- **Recommendation engine**
 - [Netflix](#), [Last.fm](#), [eHarmony](#)
- **Ad targeting, log processing, search optimization**
 - [ContextWeb](#), [eBay](#) and [Orbitz](#)
- **Machine learning and classification**
 - [Yahoo! Mail](#)'s spam detection
 - Credit card fraud detection
- **Graph analysis**
 - Social networks like [Facebook](#) and [LinkedIn](#)
 - "People you may know"

What is HDFS?

- **The distributed file system behind Hadoop**
- **Self-healing, high-bandwidth, clustered storage**
- **Based on Google File System (GFS)**
- **Built for streaming reads as opposed to random access**
- **How MapReduce achieves parallelism and data locality**

What is HDFS?

- **No random writes**
 - Files are append only once written
- **On write, files are broken up into blocks or chunks and distributed amongst the cluster**
- **Each server stores a fragment of the entire data set**

Data, data, everywhere!

- **Known Hadoop clusters:**
 - EBay - 5.3 PB
 - Facebook - 15 PB
 - Yahoo! - 16 PB (in their largest cluster)
 - Walmart - 1 PB
- **And it's growing..**
 - Gartner (2009): Enterprise Data will grow 650% in the next 5 years

Introduction to Apache Hadoop

What Is Hadoop



How MapReduce works

A Useful MapReduce Solution

An Easier Solution: Hive/Pig

Conclusion

MapReduce: The Mapper

- **Hadoop attempts to ensure that Mappers run on nodes which hold their portion of the data locally, to avoid network traffic**
 - Multiple Mappers run in parallel, each processing a portion of the input data
- **The Mapper reads data in the form of key/value pairs**
- **It outputs zero or more key/value pair**
- **Identity mapper**

```
1. public class IdentityMapper extends MapReduceBase
2. implements Mapper {
3.     public void map(Object key, Object value,
4.         OutputCollector output, Reporter report)
5.         throws IOException {
6.         output.collect(key, value);
7.     }
8. }
```

MapReduce: The Mapper (cont'd)

- **The Mapper may use or completely ignore the input key**
 - For example, a standard pattern is to read a line of a file at a time
 - In this case, the default InputFormat, TextInputFormat is used
 - The key is the byte offset into the file at which the line starts
 - The value is the contents of the line itself
 - In this case, the key is often unused
- **If it writes anything at all out, the output must be in the form of key/value pairs**

Example Mapper: Upper Case Mapper

- Turn input into upper case (pseudo-code):

```
1. public class UpperCaseMapper extends MapReduceBase
2. implements Mapper {
3.     public void map(String key, String value,
4.         OutputCollector output, Reporter report)
5.         throws IOException {
6.         output.collect(key.toUpperCase(), value.toUpperCase());
7.     }
8. }
```

Mapper Input

Twins, Win

Vikings, Lose

Mapper Output

TWINS, WIN

VIKINGS, LOSE

Example Mapper: Explode Mapper

- Output many key value pairs per input key (pseudo-code):

```
1. public class ExplodeMapper extends MapReduceBase
2. implements Mapper {
3.     public void map(String key, String value,
4.         OutputCollector output, Reporter report)
5.     throws IOException {
6.         String[] parts = value.split(":");
7.         for (String part : parts) {
8.             output.collect(part, key);
9.         }
10.    }
11. }
```

Mapper Input

host1, root:admin:noland
host2, root:prod

Mapper Output

root, host1
admin, host1
noland, host1
root, host2
prod, host2

Example Mapper: Filter Mapper

- Only output key/value pairs where the input value is a prime number (pseudo-code):

```
1. public class PrimeFilterMapper extends MapReduceBase
2. implements Mapper {
3.     public void map(String key, Integer value,
4.         OutputCollector output, Reporter report)
5.         throws IOException {
6.         if(isPrime(value)) {
7.             output.collect(key, value);
8.         }
9.     }
10. }
```

Mapper Input

key1, 7
key2, 10

Mapper Output

key1, 7

Example Mapper: Changing Keyspaces

- The key output by the Mapper does not need to be identical to the input key
- Output the word length as the key (pseudo-code):

```
1. public class LengthMapper extends MapReduceBase
2. implements Mapper {
3.     public void map(String key, String value,
4.         OutputCollector output, Reporter report)
5.         throws IOException {
6.         output.collect(value.length(), value);
7.     }
8. }
```

Mapper Input

key1, Twins

key2, Vikings

key3, Timberwolves

Mapper Output

5, Twins

7, Vikings

12, Timberwolves

Between the Map and the Reduce

- Output of Mapper is sorted while it's being generated
- The sorted output of the Mapper is delivered to the Reducer
- The output is also grouped by key

Map Output

Google, 1
Google, 1
Microsoft, 1
Google, 1
Microsoft, 1
Facebook, 1

S
o
r
t

Reduce Input

Facebook, (1)
Google, (1, 1, 1)
Microsoft, (1, 1)

MapReduce: The Reducer

- **After the Map phase is over, all the intermediate values for a given intermediate key are combined together into a list**
- **This list is given to a Reducer**
 - The intermediate keys, and their value lists, are passed to the Reducer in sorted key order
 - This step is known as the 'shuffle and sort'
- **The Reducer outputs zero or more final key/value pairs**
 - These are written to HDFS
 - In practice, the Reducer usually emits a single key/value pair for each input key

Example Reducer: Sum Reducer

- Add up all the values associated with each intermediate key (pseudo-code):

```
1. public class SumReducer extends MapReduceBase
2. implements Reducer {
3.     public void reduce(String key, Iterator<Long> values,
4.         OutputCollector output, Reporter reporter)
5.         throws IOException {
6.         long total = 0;
7.         while(values.hasNext()) {
8.             Long value = values.next();
9.             total += value;
10.        }
11.        output.collect(key, total);
12.    }
13. }
```

Reduce Input

Facebook, (1)
Google, (1, 1, 1)
Microsoft, (1, 1)

Reduce Output

Facebook, 1
Google, 3
Microsoft, 2

Example Reducer: Identity Reducer

- The Identity Reducer is very common (pseudo-code):

```
1. public class IdentityReducer extends MapReduceBase
2. implements Reducer {
3.     public void reduce(Object key, Iterator<Object> values,
4.         OutputCollector output, Reporter reporter)
5.         throws IOException {
6.         while(values.hasNext()) {
7.             output.collect(key, values.next());
8.         }
9.     }
10. }
```

Reduce Input

Facebook, (1)
Google, (1, 1, 1)
Microsoft, (1, 1)

Reduce Output

Facebook, 1
Google, 1
Google, 1
Google, 1
Microsoft, 1
Microsoft, 1

Introduction to Apache Hadoop

What Is Hadoop

How MapReduce works



A Useful MapReduce Solution

An Easier Solution: Hive/Pig

Conclusion

Writables

- In the previous “pseudo code” examples I used “String” and “Long”
- MapReduce key and values are not Java Native types but Hadoop specific types called “Writables”

Java	Writable
String	Text
Integer	IntWritable
Long	LongWritable
Double	DoubleWritable
Map	MapWritable

Our problem

- **Most companies have Apache or similar web server access logs**
- **Often these logs are used for a period and then thrown away**
- **Often a portion of these logs are stored in a database for analytics**
- **Given an Apache Access Log people typically want find the most common**
 - IP
 - Page
 - Etc
- **Or want to investigate**
 - Errors
 - Average page size
 - Etc

Counting Common IPs

- In this presentation we will tackle counting page views by IP
- Recall the default `InputFormat`, `TextInputFormat` gives us:
 - Key: byte offset of line
 - Value: the line itself
- Raw data in Apache Log Format

```
training@ubuntu:~$ hadoop fs -cat access_log/input | tail -n5
67.85.101.186 - - [28/Apr/2011:02:19:53 -0400] "GET /sun380/open.jpg HTTP/1.1" 200 74134 "http://obsolyte.com/sun380/" "Opera/9.80 (Windows NT 5.1; U; en) Presto/2.6.30 Version/10.62"
67.85.101.186 - - [28/Apr/2011:02:20:11 -0400] "GET /favicon.ico HTTP/1.1" 200 318 "http://obsolyte.com/sun380/" "Opera/9.80 (Windows NT 5.1; U; en) Presto/2.6.30 Version/10.62"
218.213.130.154 - - [28/Apr/2011:02:22:36 -0400] "GET /sunFAQ/faq_m68k/mc68008.html HTTP/1.1" 404 304 "-" "ichiro/4.0 (http://help.goo.ne.jp/door/crawler.html)"
218.213.130.154 - - [28/Apr/2011:02:25:06 -0400] "GET /sunFAQ/faq_framebuffer/preformatted/colormap_info HTTP/1.1" 404 325 "-" "ichiro/4.0 (http://help.goo.ne.jp/door/crawler.html)"
218.213.130.154 - - [28/Apr/2011:02:28:16 -0400] "GET /sunFAQ/faq_framebuffer/FrameBuffer.html HTTP/1.1" 404 315 "-" "ichiro/4.0 (http://help.goo.ne.jp/door/crawler.html)"
```

Mapper Input and Output

Map Input	Map Output
(byte offset, '95.108.158.236 - - [24/Apr/2011:04:06:01 -0400] "GET ...')	("95.108.158.236", 1)
(byte offset, '96.27.4.134 - - [24/Apr/2011:04:20:11 -0400] "GET ...')	("96.27.4.134 ", 1)
(byte offset, '100.52.4.13 - - [24/Apr/2011:04:21:11 -0400] "POST...')	("100.52.4.13", 1)
(byte offset, '5.27.4.14 - - [24/Apr/2011:05:20:11 -0400] "HEAD...')	("5.27.4.14", 1)
(byte offset, '5.27.4.14 - - [24/Apr/2011:05:20:13 -0400] "GET ...')	("5.27.4.14", 1)

IP Count Mapper

```
1. static final String logEntryPattern = "^([\\d.]+) (\\S+) (\\S+) " +
2. "\\[([\\w:/]+\\s+[\\-]\\d{4})\\] \\\"(.+?)\\\" (\\d{3}) (\\d+) \\\"([^\"]+)\\\" \\\"([^\"]+)\\\"";
3. public class IPCountMapper extends MapReduceBase implements
4.     Mapper<Object, Text, Text, LongWritable> {
5.     Pattern pattern = Pattern.compile(logEntryPattern);
6.     Text outputKey = new Text();
7.     LongWritable one = new LongWritable(1);
8.     public void map(Object key, Text value,
9.         OutputCollector<Text, LongWritable> output, Reporter report)
10.        throws IOException {
11.         Matcher matcher = pattern.matcher(value.toString());
12.         if (!matcher.matches()) {
13.             report.incrCounter("Apache", "NoMatch", 1);
14.         } else if (9 != matcher.groupCount()) {
15.             report.incrCounter("Apache", "BadMatch", 1);
16.         } else {
17.             String ip = matcher.group(1);
18.             outputKey.set(ip);
19.             output.collect(outputKey, one);
20.         }
21.     }
22. }
```

Sort and Group by Key

- Hadoop **SORTS** and **GROUPS** by **KEY** for us

Map Output	Reduce Input
("95.108.158.236", 1)	("100.52.4.13", [1, 1, 1, 1])
("96.27.4.134 ", 1) ("96.27.4.134 ", 1) ("96.27.4.134 ", 1) ...	("5.27.4.14", [1, 1])
("100.52.4.13", 1) ("100.52.4.13", 1) ("100.52.4.13", 1) ("100.52.4.13", 1)	("95.108.158.236", [1])
("5.27.4.14", 1) ("5.27.4.14", 1)	("96.27.4.134 ", [1, 1, 1, ...])

Reducer Input and Output

- Reduce sums 1's and uses total as key

Reduce Input	Reduce Output
("100.52.4.13", [1, 1, 1, 1])	(4, "100.52.4.13")
("5.27.4.14", [1, 1])	(2, "5.27.4.14")
("95.108.158.236", [1])	(1, "95.108.158.236")
("96.27.4.134 ", [1, 1, 1, ...])	(59, "96.27.4.134 ")

Sum Reducer

- Note total is is the output key

```
1. public class SumReducer extends MapReduceBase implements
2.     Reducer<Text, LongWritable, LongWritable, Text> {
3.     private LongWritable total = new LongWritable();
4.     public void reduce(Text key, Iterator<LongWritable> values,
5.         OutputCollector<LongWritable, Text> output, Reporter reporter)
6.         throws IOException {
7.         long count = 0;
8.         while (values.hasNext()) {
9.             LongWritable value = values.next();
10.            count += value.get();
11.        }
12.        total.set(count);
13.        output.collect(total, key);
14.    }
15. }
```

Reducer Output

- Reducer output is an unsorted list of frequency counts

Reduce Output	Required Output
(4, "100.52.4.13")	(59, "96.27.4.134 ")
(2, "5.27.4.14")	(4, "100.52.4.13")
(1, "95.108.158.236")	(2, "5.27.4.14")
(59, "96.27.4.134 ")	(1, "95.108.158.236")

Sort by Frequency

- **Recall the framework sorts by key**
- **In the reducer, we emitted frequency as our key**
 - (4, “100.52.4.13”)
 - (2, “5.27.4.14”)
 - (1, “95.108.158.236”)
 - (59, “96.27.4.134”)
- **Send file back through map reduce using Identity Mapper and Identity Reduce and the result will be a file sorted by frequency**

Sort by Frequency

- Hadoop includes all needed classes
- Driver code is below

```
1. JobConf job = new JobConf(conf);
2. job.setJarByClass(ApacheMostCommonIP.class);
3. job.setMapperClass(IdentityMapper.class);
4. job.setMapOutputKeyClass(LongWritable.class);
5. job.setMapOutputValueClass(Text.class);
6. job.setReducerClass(IdentityReducer.class);
7. job.setOutputKeyComparatorClass(LongWritable.DecreasingComparator.class);
8. job.setInputFormat(SequenceFileInputFormat.class);
9. FileOutputFormat.setOutputPath(job, remoteOutput);
10. FileInputFormat.addInputPath(job, remoteInput);
11. JobClient.runJob(job);
```

Java MapReduce Result

- Output files are called “part-00000” where the number is the reducer number
- Below we had one reducer

```
training@ubuntu:~$ head output/part-00000
261      81.209.177.145
232      174.133.177.66
231      76.124.159.88
189      65.175.176.72
189      87.250.253.243
125      92.14.101.156
125      67.195.114.59
124      173.18.205.66
115      38.101.148.126
112      218.213.130.154
```

Introduction to Apache Hadoop

What Is Hadoop

How MapReduce works

A Useful MapReduce Solution

 **An Easier Solution: Hive/Pig**

Conclusion

Hive & Pig

- **Facebook (Hive) and Yahoo (Pig) were early Hadoop users**
- **Pain points included:**
 - Extensive development time to write MapReduce
 - Wanted "ad hoc" queries
 - Hadoop is good for semi-structured data, but sometimes a schema is desirable

The right tool for the job

- **Hive is similar to SQL, familiar to data analysts**
- **Pig is a data flow, scripting language**

Example of Hive

```
hive> CREATE EXTERNAL
> TABLE access_log
> (ip STRING,
> identity STRING,
> username STRING,
> dt STRING,
> request STRING,
> code STRING,
> bytes STRING,
> referer STRING,
> browser STRING)
> ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'
> WITH SERDEPROPERTIES ("input.regex" = "([\d.]+) (\S+) (\S+) \[[([\w:/]
+[\s[+\-]\d{4})\]] \"(.+?)\" (\d{3}) (\d+) \"([^\"]+)\|\" \"([^\"]+)\|\"")
> LOCATION '/user/training/access_log';
Found class for org.apache.hadoop.hive.contrib.serde2.RegexSerDe
OK
Time taken: 4.067 seconds
hive> █
```

Shell “feels” like MySQL

```
hive> DESCRIBE access_log;
OK
ip          string    from deserializer
identity    string    from deserializer
username    string    from deserializer
dt          string    from deserializer
request     string    from deserializer
code        string    from deserializer
bytes       string    from deserializer
referer     string    from deserializer
browser     string    from deserializer
Time taken: 0.095 seconds
hive> █
```

Hive Solves the Problem

```
hive> SELECT COUNT(1) AS cnt, ip FROM access_log  
> WHERE ip IS NOT NULL GROUP BY ip ORDER BY cnt;
```

.....

```
112      218.213.130.154  
115      38.101.148.126  
124      173.18.205.66  
125      92.14.101.156  
125      67.195.114.59  
189      65.175.176.72  
189      87.250.253.243  
231      76.124.159.88  
232      174.133.177.66  
261      81.209.177.145  
Time taken: 53.46 seconds  
hive>
```

The Problem Changes

- **“Can you obtain the top 10 URL’s which had a non-200 response code and count and group them by URL and response code ordering them by count in descending order?”**
- **“Sure”**

Hive Solves Problem

```
hive> SELECT COUNT(1) AS cnt, code, request FROM access_log  
> WHERE code != 200 GROUP BY request, code ORDER BY cnt DESC LIMIT 10; █
```

.....

OK

```
270      404      GET /robots.txt HTTP/1.1  
113      404      GET /robots.txt HTTP/1.0  
21       206      GET /sun_ss5/ss5_int.jpg HTTP/1.1  
20       206      GET /faq/sca_adapter.jpg HTTP/1.1  
19       206      GET /sun_ss5/ss5_server.jpg HTTP/1.1  
18       206      GET /sun_ss5/ss5_rear.jpg HTTP/1.1  
18       404      GET /conf/conf.cgi HTTP/1.0  
17       206      GET /faq/68pinscsi.jpg HTTP/1.1  
17       206      GET /faq/50pinscsi.jpg HTTP/1.1  
17       206      GET /sun_ss5/ss5_jumpers.jpg HTTP/1.1
```

Time taken: 53.243 seconds

```
hive> █
```

The Problem Changes

- **“Our website has been hacked, we run PHP?”**
- **“Sure”**

Hive Solves Problem

- Until now, we could have solved our problems with a database and good indexes on reasonably sizes data sets
- At this time, we need to access semi-structured data

```
hive> SELECT * FROM access_log WHERE request RLIKE '=http';
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201104181232_0303, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201104181232_0303
Kill Command = /usr/lib/hadoop-0.20/bin/hadoop job -Dmapred.job.tracker=localhost:8021 -kill job_201104181232_0303
2011-05-09 14:03:55,545 Stage-1 map = 0%, reduce = 0%
2011-05-09 14:04:07,594 Stage-1 map = 100%, reduce = 0%
2011-05-09 14:04:10,612 Stage-1 map = 100%, reduce = 100%
Ended Job = job_201104181232_0303
OK
71.123.44.26 - - 24/Apr/2011:23:29:08 -0400 GET //products_new.php?action=buy_now&products_id=http://www.livlife.us/images/thefamily/ciao.txt??? HTTP/1.1 404 293 - libwww-perl/5.831
Time taken: 16.598 seconds
```

Hive and Java

- **Hive is written in Java and supports extension via Java**
 - User Defined Functions
 - SerDe's like RegEx SerDe which reads a file based on a user defined RegEx

Hive, JDBC & ODBC

- **Hive supports most of the SQL-92 language**
- **JDBC (Java Database Connectivity) and ODBC(Open Database Connectivity) connectors**
- **BI vendors like MicroStrategy, Pentaho and Jaspersoft integrating with Hadoop through Hive**

Introduction to Apache Hadoop

What Is Hadoop

How MapReduce works

A Useful MapReduce Solution

An Easier Solution: Hive/Pig



Conclusion

Conclusion

- **Hadoop is data storage and analysis platform for large volumes of data**
- **Hadoop will sit along side, not replace your existing RDBMS**
- **Hadoop has many tools to ease data analysis**
- **Cloudera offers training and consulting in addition to a stable Apache Hadoop distribution and enterprise tools**
- **brock@cloudera.com**