Introduction to Hadoop

Owen O'Malley
Yahoo!, CC&DI
owen@yahoo-inc.com

Who Am I?

- Two Different Hats
  - Yahoo! Architect on Hadoop Map/Reduce
    - Design, review, and implement features in Hadoop
    - Working on Hadoop full time since Feb 2006
  - VP of Apache for Hadoop
    - Chair of the Hadoop Program Management Committee
    - Responsible for
      - Building the Hadoop community
      - Interfacing between the Hadoop PMC and the Apache Board

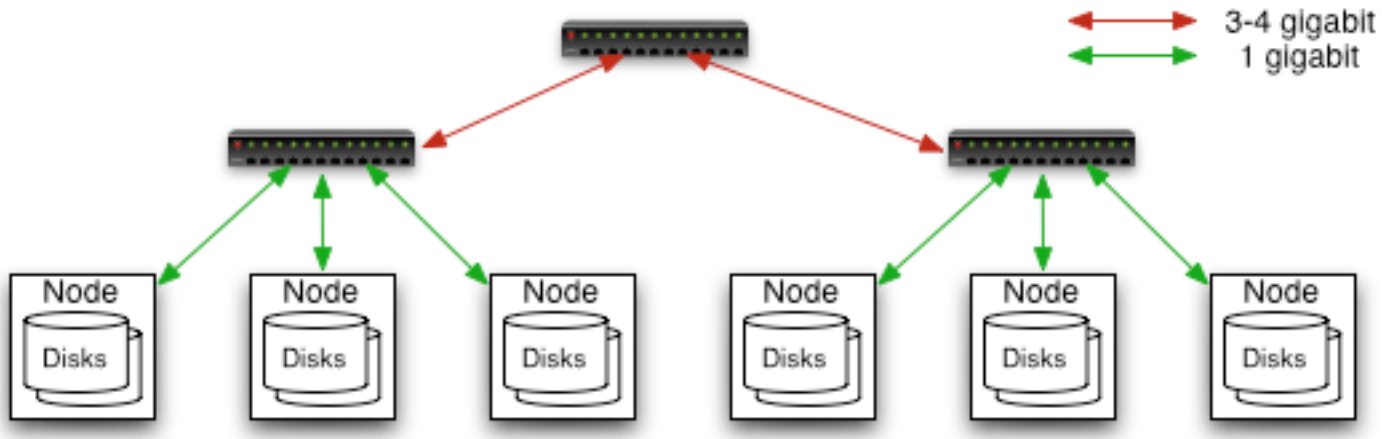
Problem

- How do you scale up applications?
  - 100's of terabytes of data
  - Takes 11 days to read on 1 computer
- Need lots of cheap computers
  - Fixes speed problem (15 minutes on 1000 computers), but…
  - Reliability problems
    - In large clusters, computers fail every day
    - Cluster size is not fixed
- Need common infrastructure
  - Must be efficient and reliable

Solution

- Open Source Apache Project
- Hadoop Core includes:
  - Distributed File System - distributes data
  - Map/Reduce - distributes application
- Written in Java
- Runs on
  - Linux, Mac OS/X, Windows, and Solaris
  - Commodity hardware

Commodity Hardware Cluster



- Typically in 2 level architecture
  - Nodes are commodity PCs
  - 40 nodes/rack
  - Uplink from rack is 8 gigabit
  - Rack-internal is 1 gigabit

Distributed File System

- Single namespace for entire cluster
  - Managed by a single *namenode*.
  - Files are append-only.
  - Optimized for streaming reads of large files.
- Files are broken in to large blocks.
  - Typically 128 MB
  - Replicated to several *datanodes*, for reliability
- Client talks to both namenode and datanodes
  - Data is not sent through the namenode.
  - Throughput of file system scales nearly linearly with the number of nodes.
- Access from Java, C, or command line.

Block Placement

- Default is 3 replicas, but settable
- Blocks are placed (writes are pipelined):
  - On same node
  - On different rack
  - On the other rack
- Clients read from closest replica
- If the replication for a block drops below target, it is automatically re-replicated.

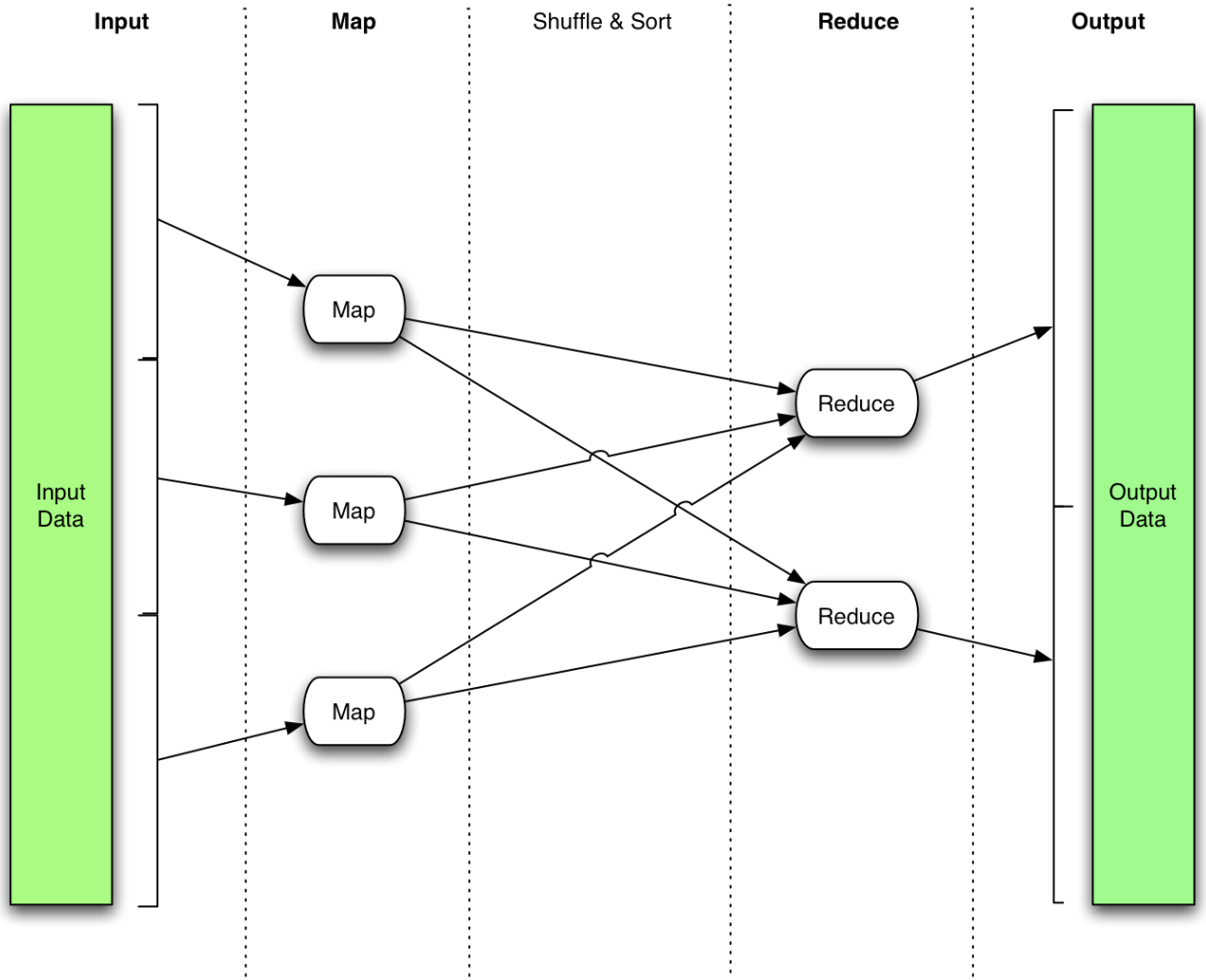
Data Correctness

- Data is checked with CRC32
- File Creation
  - Client computes checksum per 512 byte
  - DataNode stores the checksum
- File access
  - Client retrieves the data and checksum from DataNode
  - If Validation fails, Client tries other replicas

Map/Reduce

- Map/Reduce is a programming model for efficient distributed computing
- It works like a Unix pipeline:
  - cat input | grep | sort | uniq -c | cat > output
  - **Input** | **Map** | Shuffle & Sort | **Reduce** | **Output**
- Efficiency from
  - Streaming through data, reducing seeks
  - Pipelining
- A good fit for a lot of applications
  - Log processing
  - Web index building

Map/Reduce Dataflow



Map/Reduce features

- Fine grained Map and Reduce tasks
  - Improved load balancing
  - Faster recovery from failed tasks
- Automatic re-execution on failure
  - In a large cluster, some nodes are always slow or flaky
  - Framework re-executes failed tasks
- Locality optimizations
  - With large data, bandwidth to data is a problem
  - Map-Reduce + HDFS is a very effective solution
  - Map-Reduce queries HDFS for locations of input data
  - Map tasks are scheduled close to the inputs when possible

Why Yahoo! is investing in Hadoop

- We started with building better applications
  - Scale up web scale batch applications (search, ads, …)
  - Factor out common code from existing systems, so new applications will be easier to write
  - Manage the many clusters we have more easily
- The mission now includes research support
  - Build a **huge** data warehouse with many Yahoo! data sets
  - Couple it with a huge compute cluster and programming models to make using the data easy
  - Provide this as a service to our researchers
  - We are seeing great results!
    - Experiments can be run much more quickly in this environment

Hadoop Timeline

- 2004 – HDFS & map/reduce started in Nutch
- Dec 2005 – Nutch ported to map/reduce
- Jan 2006 – Doug Cutting joins Yahoo
- Feb 2006 – Hadoop splits out of Nutch and Yahoo starts using it.
- Apr 2006 – Hadoop sorts 1.9 TB on 188 nodes in 47 hours
- May 2006 – Yahoo sets up research cluster
- Jan 2008 – Hadoop is a top level Apache project
- Feb 2008 – Yahoo creating Webmap with Hadoop

Running the Production WebMap

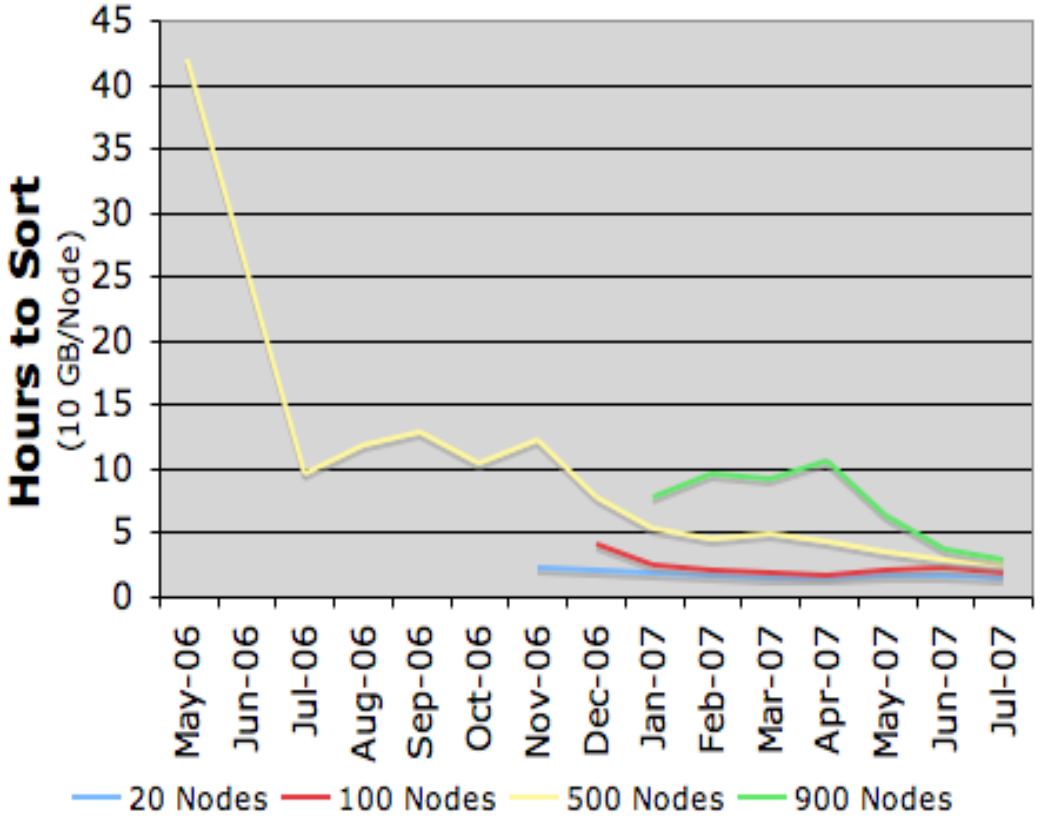
- Search needs a graph of the “known” web
  - Invert edges, compute link text, whole graph heuristics
- Periodic batch job using Map/Reduce
  - Uses a chain of ~100 map/reduce jobs
- Scale
  - 1 trillion edges in graph
  - Final output is 300 TB compressed
  - Runs on 10,000 cores
  - Raw disk used 5 PB
- Written mostly using Hadoop’s C++ interface

Research Clusters

- The grid team runs the research clusters as a service to Yahoo researchers
- Mostly data mining/machine learning jobs
- Most research jobs are *not* Java:
  - 42% Streaming
    - Uses Unix text processing to define map and reduce
  - 28% Pig
    - Higher level dataflow scripting language
  - 28% Java
  - 2% C++

Scaling Hadoop

- Benchmark sorting 10 GB / node
- Hardware held constant



Scaling Up Hadoop

- Started as a rough prototype with lots of rough edges
- Usability
  - Improved web UI
  - Save and display application logs
- Performance
  - Removed round trips to disk
  - Replaced inefficient data structures.
  - Improved block placement
- Reliability
  - Improved RPC error detection

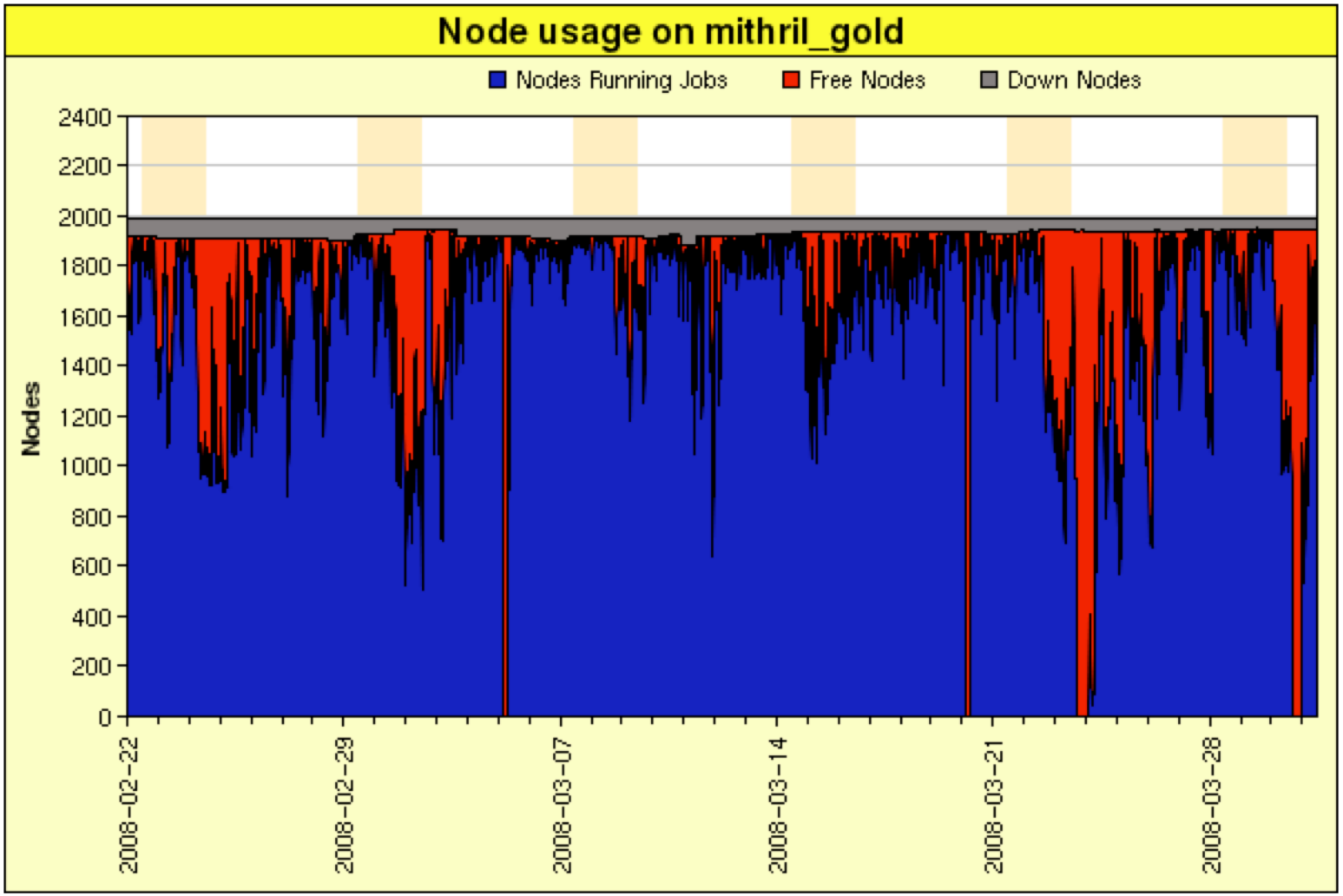
Terabyte Sort Benchmark

- Started by Jim Gray at Microsoft in 1998
- Sorting 10 billion 100 byte records
- Hadoop won the general category in 209 seconds
  - 910 nodes
  - 2 quad-core Xeons @ 2.0Ghz / node
  - 4 SATA disks / node
  - 8 GB ram / node
  - 1 gb ethernet / node
  - 40 nodes / rack
  - 8 gb ethernet uplink / rack
- Previous records was 297 seconds
- Only hard parts were:
  - Getting a total order
  - Converting the data generator to map/reduce

Hadoop clusters

- We have ~17,000 machines running Hadoop
- Our largest clusters are currently 2000 nodes
- Several petabytes of user data (compressed, unreplicated)
- We run hundreds of thousands of jobs every month

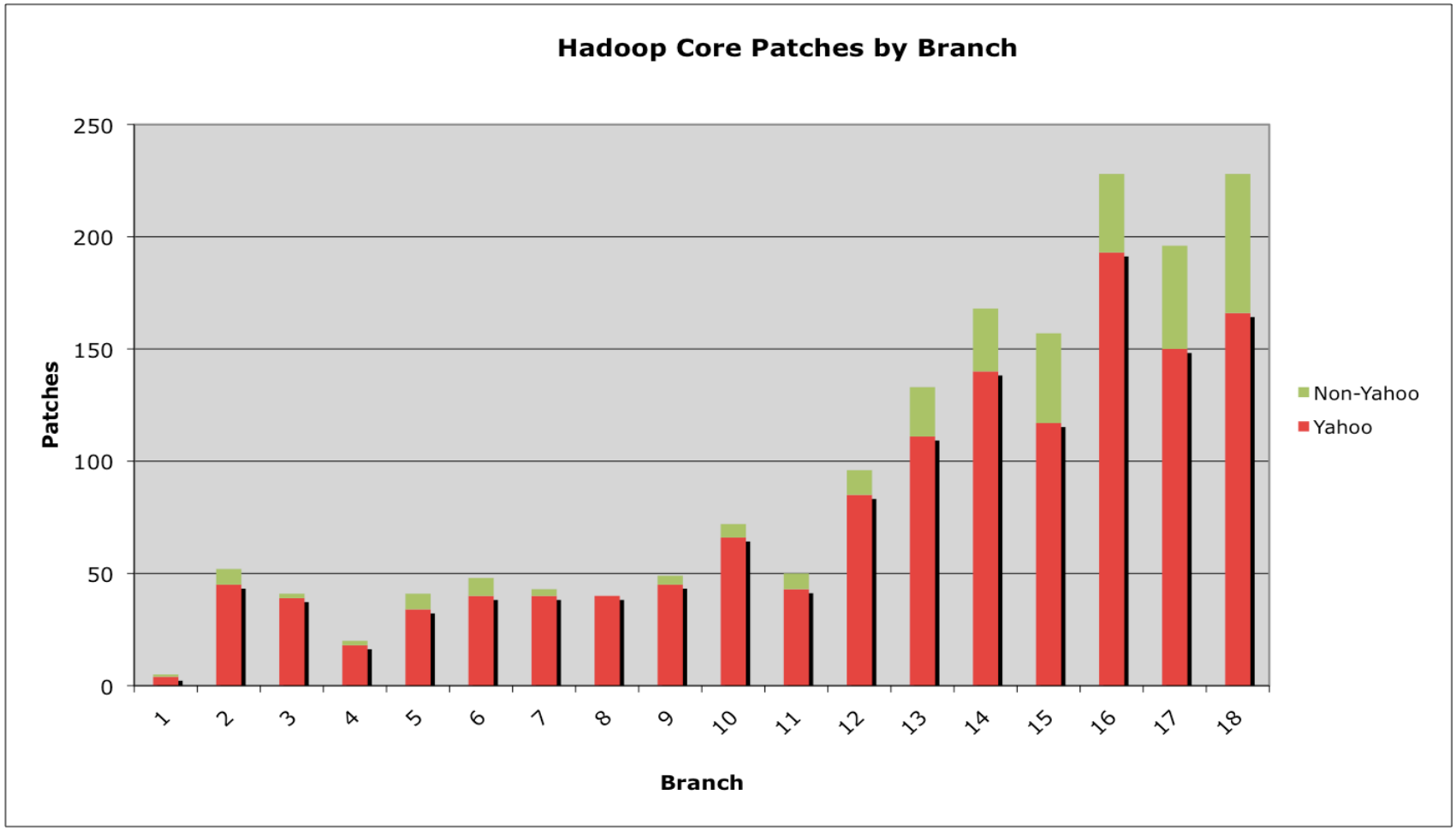
Research Cluster Usage



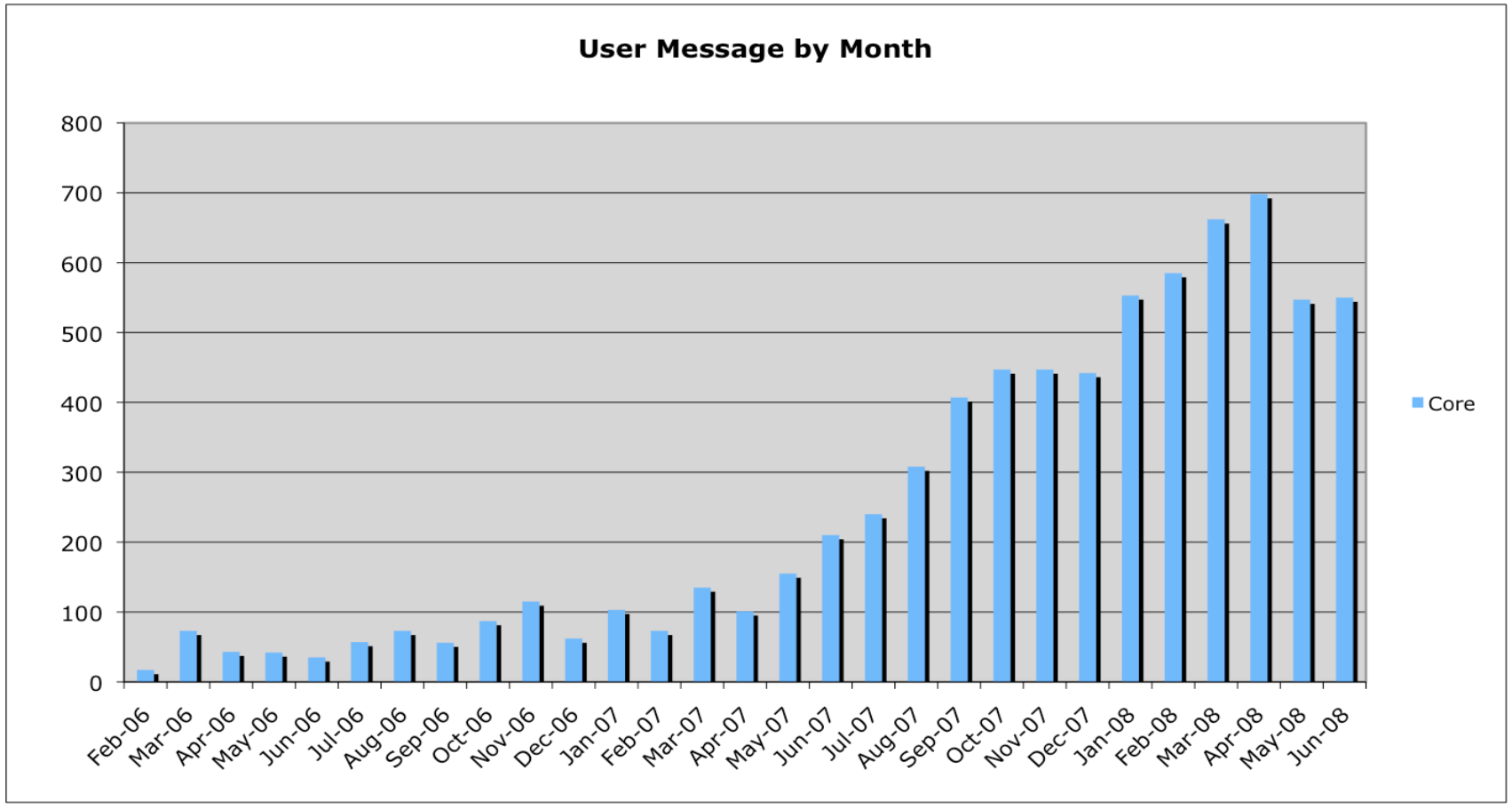
Hadoop Community

- Apache is focused on project communities
  - Users
  - Contributors
    - write patches
  - Committers
    - can commit patches **too**
  - Project Management Committee
    - vote on new committers and releases **too**
- Apache is a meritocracy
- Use, contribution, and diversity is growing
  - But we need and want more!

Size of Releases



Size of User Community



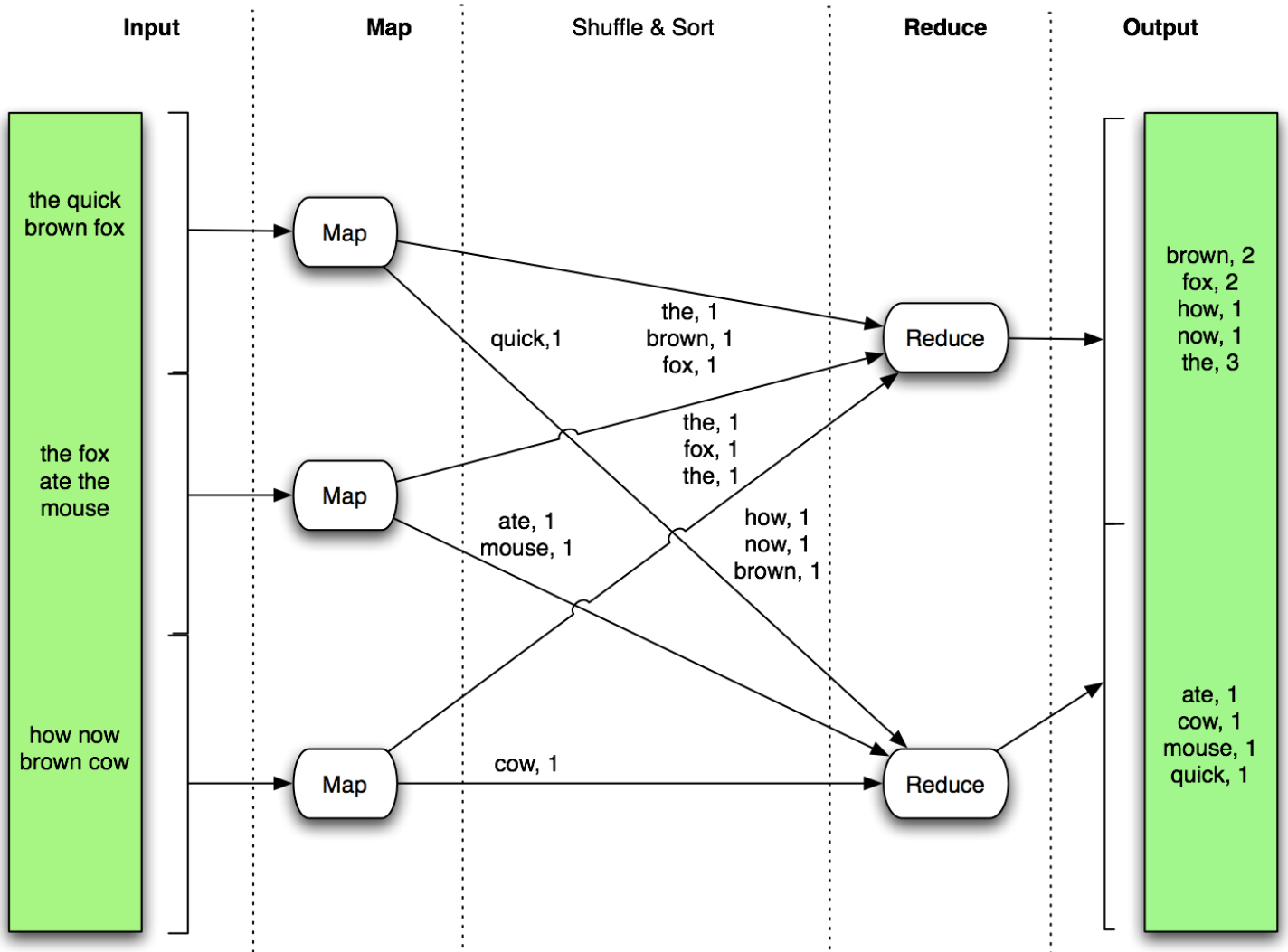
Who Uses Hadoop?

- Amazon/A9
- Facebook
- Google
- IBM
- Joost
- Last.fm
- New York Times
- PowerSet (now Microsoft)
- Veoh
- Yahoo!
- More at http://wiki.apache.org/hadoop/PoweredBy

Word Count Example

- Mapper
  - Input: value: lines of text of input
  - Output: key: word, value: 1
- Reducer
  - Input: key: word, value: set of counts
  - Output: key: word, value: sum
- Launching program
  - Defines the job
  - Submits job to cluster

Word Count Dataflow



Example: Word Count Mapper

```
public static class MapClass 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 itr = new StringTokenizer(line);
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        output.collect(word, one);
      }
    }
  }
```

Example: Word Count Reducer

```
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));
   }
 }
```

Configuring a Job

- Jobs are controlled by configuring *JobConfs*
- JobConfs are maps from attribute names to string value
- The framework defines attributes to control how the job is executed.

```
conf.set("mapred.job.name", "MyApp");
```

- Applications can add arbitrary values to the JobConf

```
conf.set("my.string", "foo");
conf.setInteger("my.integer", 12);
```

- JobConf is available to all of the tasks

Putting it all together

- Create a launching program for your application
- The launching program configures:
  - The *Mapper* and *Reducer* to use
  - The output key and value types (input types are inferred from the *InputFormat)*
  - The locations for your input and output
- The launching program then submits the job and typically waits for it to complete

Putting it all together

```
public class WordCount {
……
public static void main(String[] args) throws IOException {
    JobConf conf = new JobConf(WordCount.class);
    // the keys are words (strings)
    conf.setOutputKeyClass(Text.class);
    // the values are counts (ints)
    conf.setOutputValueClass(IntWritable.class);

    conf.setMapperClass(MapClass.class);
    conf.setReducerClass(Reduce.class);
    conf.setInputPath(new Path(args[0]);
    conf.setOutputPath(new Path(args[1]);
    JobClient.runJob(conf);
…..
```

A Counter Example

- Bob wanted to count lines in text files totaling several terabytes
- He used
  - Identity Mapper (input copied directly to output)
  - A single Reducer that counts the lines and outputs the total
- What is he doing wrong ?
- This really happened!
- Take home message is that Hadoop is **powerful** and can be dangerous in the wrong hands…

Some handy tools

- Input/Output Formats
- Partitioners
- Combiners
- Compression
- Counters
- Speculation
- Zero reduces
- Distributed File Cache
- Tool

Input and Output Formats

- A Map/Reduce may specify how it's input is to be read by specifying an *InputFormat* to be used
- A Map/Reduce may specify how it's output is to be written by specifying an *OutputFormat* to be used
- These default to *TextInputFormat* and *TextOutputFormat,* which process line-based text data
- Another common choice is SequenceFileInputFormat and SequenceFileOutputFormat for binary data
- These are file-based, but they are not required to be

Partitioners

- Partitioners are application code that define how keys are assigned to reduces
- Default partitioning spreads keys evenly, but randomly
  - Uses *key.hashCode() % num_reduces*
- Custom partitioning is often required, for example, to produce a total order in the output
  - Should implement *Partitioner* interface
  - Set by calling `conf.setPartitionerClass(MyPart.class)`
  - To get a total order, sample the map output keys and pick values to divide the keys into roughly equal buckets and use that in your partitioner

Combiners

- When *maps* produce many repeated keys
  - It is often useful to do a local aggregation following the *map*
  - Done by specifying a *Combiner*
  - Goal is to decrease size of the transient data
  - Combiners have the same interface as Reduces, and often are the same class.
  - Combiners must **not** have side effects, because they run an indeterminate number of times.
  - In *WordCount*, `conf.setCombinerClass(Reduce.class);`

Compression

- Compressing the outputs and intermediate data will often yield huge performance gains
  - Can be specified via a configuration file or set programatically
  - Set *mapred.output.compress* to *true* to compress job output
  - Set *mapred.compress.map.output* to *true* to compress map outputs
- Compression Types *(mapred.output.compression.type)*
  - “block” - Group of keys and values are compressed together
  - “record” - Each value is compressed individually
  - Block compression is almost always best
- Compression Codecs *(mapred(.map)?.output.compression.codec)*
  - Default (zlib) - slower, but more compression
  - LZO - faster, but less compression

Counters

- Often Map/Reduce applications have countable events
- For example, framework counts records in to and out of Mapper and Reducer
- To define user counters:

```
static enum Counter {EVENT1, EVENT2};
reporter.incrCounter(Counter.EVENT1, 1);
```

- Define nice names in a MyClass_Counter.properties file

```
CounterGroupName=My Counters
EVENT1.name=Event 1
EVENT2.name=Event 2
```

Speculative execution

- The framework can run multiple instances of slow tasks
  - Output from instance that finishes first is used
  - Controlled by the configuration variable *mapred.speculative.execution*
  - Can dramatically bring in long tails on jobs

Zero Reduces

- Frequently, we only need to run a filter on the input data
  - No sorting or shuffling required by the job
  - Set the number of reduces to 0
  - Output from maps will go directly to OutputFormat and disk

Distributed File Cache

- Sometimes need read-only copies of data on the local computer.
  - Downloading 1GB of data for each Mapper is expensive
- Define list of files you need to download in JobConf
- Files are downloaded once per a computer
- Add to launching program:

```
DistributedCache.addCacheFile(new URI("hdfs://nn:8020/foo"), conf);
```

- Add to task:

```
Path[] files = DistributedCache.getLocalCacheFiles(conf);
```

Tool

- Handle “standard” Hadoop command line options:
  - -conf file - load a configuration file named file
  - -D prop=value - define a single configuration property prop
- Class looks like:

```
public class MyApp extends Configured implements Tool {
  public static void main(String[] args) throws Exception {
    System.exit(ToolRunner.run(new Configuration(),
                               new MyApp(), args));
  }
  public int run(String[] args) throws Exception {
    .... getConf() ...
  }
}
```

Non-Java Interfaces

- Streaming
- Pipes (C++)
- Pig

Streaming

- What about non-programmers?
  - Can define Mapper and Reducer using Unix text filters
  - Typically use grep, sed, python, or perl scripts
- Format for input and output is: **key \t value \n**
- Allows for easy debugging and experimentation
- Slower than Java programs

```
bin/hadoop jar hadoop-streaming.jar -input in-dir -output out-dir
  -mapper streamingMapper.sh -reducer streamingReducer.sh
```

- Mapper: `sed -e 's| |\n|g' | grep .`
- Reducer: `uniq -c | awk '{print $2 "\t" $1}'`

Pipes (C++)

- C++ API and library to link application with
- C++ application is launched as a sub-process of the Java task
- Keys and values are std::string with binary data
- Word count map looks like:

```
class WordCountMap: public HadoopPipes::Mapper {
public:
  WordCountMap(HadoopPipes::TaskContext& context){}
  void map(HadoopPipes::MapContext& context) {
    std::vector<std::string> words =
      HadoopUtils::splitString(context.getInputValue(), " ");
    for(unsigned int i=0; i < words.size(); ++i) {
      context.emit(words[i], "1");
    }}};
```

Pig

- Scripting language that generates Map/Reduce jobs
- User uses higher level operations
  - Group by
  - Foreach
- Word Count:

```
input = LOAD 'in-dir' USING TextLoader();
words = FOREACH input GENERATE
   FLATTEN(TOKENIZE(*));
grouped = GROUP words BY $0;
counts = FOREACH grouped GENERATE group,
   COUNT(words);
STORE counts INTO 'out-dir';
```

What's Next?

- Better scheduling
  - Pluggable scheduler
  - Queues for controlling resource allocation
- Total Order Sampler and Partitioner
- Hive
- Interface cleanups
- Table store library
- HDFS and Map/Reduce security
- High Availability via Zookeeper
- Name Node scaling

Q&A

- For more information:
  - Website: http://hadoop.apache.org/core
  - Mailing lists:
    - core-dev@hadoop.apache
    - core-user@hadoop.apache
  - IRC: #hadoop on irc.freenode.org

Debugging & Diagnosis

- Run job with the Local Runner
  - Set mapred.job.tracker to “local”
  - Runs application in a single process and thread
- Run job on a small data set on a 1 node cluster
  - Can be done on your local dev box
- Set *keep.failed.task.files* to true
  - This will keep files from failed tasks that can be used for debugging
  - Use the IsolationRunner to run just the failed task
- Java Debugging hints
  - Send a *kill -QUIT* to the Java process to get the call stack, locks held, deadlocks