Hadoop
divide and conquer petabyte-scale data

The Basics: HDFS & MapReduce
METADATA

Matthew McCullough
Ambient Ideas, LLC
matthewm@ambientideas.com
http://ambientideas.com/blog
@matthewmccull

Code
http://github.com/matthewmccullough/hadoop-intro
Why Hadoop?
HDFS
MapReduce
Your big data

- Data type?
- Business need?
- Processing?
- How large?
- Growth projections?
- What if you saved everything?
Why Hadoop?
HDFS
MapReduce
Why Hadoop?
I use Hadoop often.
I use Hadoop often.

That's the sound my Tivo makes every time I skip a commercial.

-Brian Goetz, author of Java Concurrency in Practice
BIG DATA
NO SQL
NO SQL
NO SQL
Not SQL
Not SQL
Not Only SQL
NOSQL

- Applies to data
  - No strong schemas
  - No foreign keys

- Applies to processing
  - No SQL-99 standard
  - No execution plan
The computer you are using right now may very well have the fastest GHz processor you'll ever own.
Scale up?
Scale out
WEB CRAWLING
open source

APACHE
MARKETING!
NAME
RESEARCH?
Daughter's stuffed toy
Lucene
Lucene\n\n\nNutch
Lucene

Nutch

Hadoop
Lucene

Mahout

Nutch

Hadoop
0.21.0 current version

Dozens of companies contributing

Hundreds of companies using
Applications and organizations using Hadoop include (alphabetically):

- **A9.com** - Amazon
  - We build Amazon's product search indices using the streaming API and pre-existing C++, Perl, and Python tools.
  - We process millions of sessions daily for analytics, using both the Java and streaming APIs.
  - Our clusters vary from 1 to 100 nodes.

- **Adobe**
  - We use Hadoop and HBase in several areas from social services to structured data storage and processing for internal use.
  - We currently have about 30 nodes running HDFS, Hadoop and HBase in clusters ranging from 5 to 14 nodes on both production and development. We plan a deployment on an 80 nodes cluster.
  - We constantly write data to HBase and run MapReduce jobs to process then store it back to HBase or external systems.
  - Our production cluster has been running since Oct 2008.
We have one of the world’s smaller Hadoop clusters (2 nodes @ 8 CPUs/node).
- Hadoop and Nutch used to analyze and index textual information.

- **Adknowledge** - Ad network
  - Hadoop used to build the recommender system for behavioral targeting, plus other clickstream analytics.
  - We handle 500MM clickstream events per day.
  - Our clusters vary from 50 to 200 nodes, mostly on EC2.
  - Investigating use of R clusters atop Hadoop for statistical analysis and modeling at scale.

- **Alibaba**
  - A 15-node cluster dedicated to processing sorts of business data dumped out of database and joining them together. These data will then be fed into iSearch, our vertical search engine.
  - Each node has 8 cores, 16G RAM and 1.4T storage.

- **Amazon Web Services**
  - We provide Amazon Elastic MapReduce. It's a web service that provides a hosted Hadoop framework running on the web-scale infrastructure of Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3).
  - Our customers can instantly provision as much or as little capacity as they like to perform data-intensive tasks for applications such as web indexing, data mining, log file analysis, machine learning, financial analysis, scientific simulation, and bioinformatics research.

- **AOL**
  - We use Hadoop for a variety of things ranging from ETL style processing and statistics generation to running advanced algorithms for doing behavioral advertising.
  - We have about 175 Hadoop nodes, with 4 clusters.
  - Each cluster has about 80 nodes.
We are one of six universities participating in IBM/Google's academic cloud computing initiative. Ongoing research and teaching efforts include projects in machine translation, language modeling, bioinformatics, email analysis, and image processing.

- **University of Nebraska Lincoln, Research Computing Facility**
  We currently run one medium-sized Hadoop cluster (200TB) to store and serve up physics data for the computing portion of the Compact Muon Solenoid (CMS) experiment. This requires a filesystem which can download data at multiple Gbps and process data at an even higher rate locally. Additionally, several of our students are involved in research projects on Hadoop.

- **Veoh**
  - We use a small Hadoop cluster to reduce usage data for internal metrics, for search indexing and for recommendation data.

- **Visible Measures Corporation** uses Hadoop as a component in our Scalable Data Pipeline, which ultimately powers VisibleSuite and other products. We use Hadoop to aggregate, store, and analyze data related to in-stream viewing behavior of Internet video audiences. Our current grid contains more than 128 CPU cores and in excess of 100 terabytes of storage, and we plan to grow that substantially during 2008.

- **VK Solutions**
  - We use a small Hadoop cluster in the scope of our general research activities at **VK Labs** to get a faster data access from web applications.
  - We also use Hadoop for filtering and indexing listing, processing log analysis, and for recommendation data.

- **Vuelos baratos**
  - We use a small Hadoop
- **Vuelos baratos**
  - We use a small Hadoop

- **WorldLingo**
  - Hardware: 44 servers (each server has: 2 dual core CPUs, 2TB storage, 8GB RAM)
  - Each server runs Xen with one Hadoop/HBase instance and another instance with web or application servers, giving us 88 usable virtual machines.
  - We run two separate Hadoop/HBase clusters with 22 nodes each.
  - Hadoop is primarily used to run HBase and Map/Reduce jobs scanning over the HBase tables to perform specific tasks.
  - HBase is used as a scalable and fast storage back end for millions of documents.
  - Currently we store 12 million documents with a target of 450 million in the near future.

- **Yahoo!**
  - More than 100,000 CPUs in >25,000 computers running Hadoop
  - Our biggest cluster: 4000 nodes (2*4cpu boxes w 4*1TB disk & 16GB RAM)
    - Used to support research for Ad Systems and Web Search
    - Also used to do scaling tests to support development of Hadoop on larger clusters
  - **Our Blog** - Learn more about how we use Hadoop.
  - >40% of Hadoop Jobs within Yahoo are Pig jobs.

- **Zvents**
  - 10 node cluster (Dual-Core AMD Opteron 2210, 4GB RAM, 1TB/node storage)
  - Run Naive Bayes classifiers in parallel over crawl data to discover event information
Bing
Why Hadoop?
HDFS
MapReduce
HDFS
VIRTUAL MACHINE
VM SOURCES

Yahoo
  True to the OSS distribution

Cloudera
  Desktop tools

Both VMWare based
STARTING UP
Tailing the logs
Zeus /Applications/Dev/hadoop-family/hadoop-0.20.1/logs

24 logs % 11:14:31
FILESYSTEM
The Google File System
Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google

ABSTRACT
We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance with minimal overhead, high performance, and iterative co-design with the underlying software. Our file system is designed to run on inexpensive commodity servers, and it delivers high aggregate performance at a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our applications and technologies and our implementation of a community design. The largest change in our design compared to previous systems is to treat the client as part of the distributed system. The client is visible to the server, and the server communicates with the client directly. The client can be on a dedicated server or on a desktop client.

Categories and Subject Descriptors
D.4.3 [Distributed File Systems]

General Terms
Design, reliability, performance, measurement.

Keywords
Fault tolerance, scalability, data storage, clustered storage.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires specific permission and/or a fee.

ACM Copyright 2003, DocuLiability. New York, USA.
Copyright 2003 ACM 1-58113-727-5/03/0002/10-0 $5.00.

1. INTRODUCTION
We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google data processing needs. GFS shares many of the same goals as previous distributed file systems such as improved availability, reliability, scalability, and availability. However, its design has been driven by our need to run across a vast number of machines.

Our system is designed to run on inexpensive commodity servers. Thus, the cost of hardware can be amortized over a large number of servers. The lower cost of hardware is important to Google, as we have a very large number of machines (hundreds of thousands).

The system is designed to be scalable, with the ability to process very large amounts of data. As a result, the system is designed to run on a large number of machines (hundreds of thousands).

The system is designed to be available in the sense that it can be run on a large number of machines (hundreds of thousands).

The system is designed to be scalable, with the ability to process very large amounts of data. As a result, the system is designed to run on a large number of machines (hundreds of thousands).

The system is designed to be available in the sense that it can be run on a large number of machines (hundreds of thousands).

The system is designed to be scalable, with the ability to process very large amounts of data. As a result, the system is designed to run on a large number of machines (hundreds of thousands).

The system is designed to be available in the sense that it can be run on a large number of machines (hundreds of thousands).

The system is designed to be scalable, with the ability to process very large amounts of data. As a result, the system is designed to run on a large number of machines (hundreds of thousands).

The system is designed to be available in the sense that it can be run on a large number of machines (hundreds of thousands).

The system is designed to be scalable, with the ability to process very large amounts of data. As a result, the system is designed to run on a large number of machines (hundreds of thousands).

The system is designed to be available in the sense that it can be run on a large number of machines (hundreds of thousands).

The system is designed to be scalable, with the ability to process very large amounts of data. As a result, the system is designed to run on a large number of machines (hundreds of thousands).

The system is designed to be available in the sense that it can be run on a large number of machines (hundreds of thousands).

The system is designed to be scalable, with the ability to process very large amounts of data. As a result, the system is designed to run on a large number of machines (hundreds of thousands).

The system is designed to be available in the sense that it can be run on a large number of machines (hundreds of thousands).

The system is designed to be scalable, with the ability to process very large amounts of data. As a result, the system is designed to run on a large number of machines (hundreds of thousands).

The system is designed to be available in the sense that it can be run on a large number of machines (hundreds of thousands).

The system is designed to be scalable, with the ability to process very large amounts of data. As a result, the system is designed to run on a large number of machines (hundreds of thousands).

The system is designed to be available in the sense that it can be run on a large number of machines (hundreds of thousands).

The system is designed to be scalable, with the ability to process very large amounts of data. As a result, the system is designed to run on a large number of machines (hundreds of thousands).

The system is designed to be available in the sense that it can be run on a large number of machines (hundreds of thousands).

The system is designed to be scalable, with the ability to process very large amounts of data. As a result, the system is designed to run on a large number of machines (hundreds of thousands).

The system is designed to be available in the sense that it can be run on a large number of machines (hundreds of thousands).

The system is designed to be scalable, with the ability to process very large amounts of data. As a result, the system is designed to run on a large number of machines (hundreds of thousands).

The system is designed to be available in the sense that it can be run on a large number of machines (hundreds of thousands).

The system is designed to be scalable, with the ability to process very large amounts of data. As a result, the system is designed to run on a large number of machines (hundreds of thousands).

The system is designed to be available in the sense that it can be run on a large number of machines (hundreds of thousands).

The system is designed to be scalable, with the ability to process very large amounts of data. As a result, the system is designed to run on a large number of machines (hundreds of thousands).

The system is designed to be available in the sense that it can be run on a large number of machines (hundreds of thousands).

The system is designed to be scalable, with the ability to process very large amounts of data. As a result, the system is designed to run on a large number of machines (hundreds of thousands).

The system is designed to be available in the sense that it can be run on a large number of machines (hundreds of thousands).

The system is designed to be scalable, with the ability to process very large amounts of data. As a result, the system is designed to run on a large number of machines (hundreds of thousands).

The system is designed to be available in the sense that it can be run on a large number of machines (hundreds of thousands).
"scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware"
HDFS BASICS

- Open Source implementation of Google BigTable
- Replicated data store
- Stored in 64MB blocks
HDFS

- Rack location aware
- Configurable redundancy factor
- Self-healing
- Looks almost like *NIX filesystem
WHY HDFS?

- Random reads
- Parallel reads
- Redundancy
DATA OVERLOAD
DATA OVERLOAD

overflow from
traditional
RDBMS or log
files

destination?
DATA OVERLOAD

overflow from traditional RDBMS or log files

destination?
overflow from traditional RDBMS or log files
DATA OVERLOAD

overflow from traditional RDBMS or log files

destination?
HDFS File System Shell Guide

Overview
- cat
- chgrp
- chmod
- chown
- copyFromLocal
- copyToLocal
- count
- cp
- du
- dus
- expunge
- get
- getmerge
- ls
- lsr
- lsr
- mkdir
- makedir
TEST HDFS

- Upload a file
- List directories
HDFS UPLOAD

- Show contents of file in HDFS
- Show vocabulary of HDFS
HDFS CHALLENGES

- Writes are re-writes today
- Append is planned
- Block size, alignment
- Small file inefficiencies
- NameNode SPOF
Why Hadoop?
HDFS
MapReduce
MapReduce
MapReduce is functional programming on a distributed processing platform.
HADOOP'S PREMISE

Geography-aware distribution of brute-force processing
MAPREDUCE
the algorithm
MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat
jef@google.com, sanjay@google.com
Google, Inc.

Abstract

MapReduce is a simplification of the traditional programming model and architecture for large-scale data processing. It expresses parallel data processing as the composition of map and reduce operations. The programmer specifies a map function that processes a key-value pair, and a reduce function that combines the values for each key. Intermediate results are stored in persistent memory accessible across a network. Many such computations can be expressed in a few lines of code. The paper describes the design choices and implementation of MapReduce and presents two examples. The system is highly fault-tolerant, it processes terabytes of data per machine, and it has been tested on clusters of thousands of commodity machines. The experience gained while implementing MapReduce has resulted in a new distributed programming model that embeds the MapReduce model as a library.

1 Introduction

Over the past five years, the authors and many others at Google have implemented hundreds of special-purpose applications as distributed programs. These applications process large volumes of raw data, such as crawled web pages, search request logs, and user behavior data. Our experience has shown that the majority of these applications process data in a map-reduce style, a pattern that lends itself to scalability. We have implemented it as a library and released it to the public as Hadoop.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google’s clusters every day.

1.1 Stitching Together

Many programs use MapReduce as a library to stitch together various abstractions. Consider a problem where one wishes to count the frequency of a substring across a large corpus of text. Without MapReduce, one would need to write a program that iterates over the text and counts the substring frequency. With MapReduce, one specifies two functions, a mapper that iterates over the text and emits intermediate key-value pairs, and a reducer that sums the intermediate values for each key. This abstraction allows one to focus on the high-level problem and to resolve lower-level details in a systematic way. MapReduce is not the only programming model that accomplishes this, but it is highly powerful, easy to use, and highly fault-tolerant.

To appear in NSDI 2004
"A programming model and implementation for processing and generating large data sets"
VERIFY SETUP

- Launch “cloudera-training” VMWare instance
- Open a terminal window
- Verify hadoop
THE PROCESS

- **Map**\((k_1, v_1) \rightarrow \text{list}(k_2, v_2)\)
  - Every item is parallel candidate for Map

- **Shuffle** (group) pairs from all lists by key

- **Reduce**\((k_2, \text{list } (v_2)) \rightarrow \text{list}(v_3)\)
  - Reduce in parallel on each group of keys
Split

Map

Map

Map

Shuffle

Shuffle

Reduce

Reduce
MAPREDUCE

- Run the grep Shakespeare job
- View the status consoles
MAPREDUCE

a word counting conceptual example
THE GOAL

Provide the occurrence count of each distinct word across all documents
At four years old I acted out
Glad I am not four years old
Yet I still act like someone four
<table>
<thead>
<tr>
<th>at</th>
<th>glad</th>
<th>yet</th>
</tr>
</thead>
<tbody>
<tr>
<td>four</td>
<td>I</td>
<td>l</td>
</tr>
<tr>
<td>years</td>
<td>am</td>
<td>still</td>
</tr>
<tr>
<td>old</td>
<td>not</td>
<td>act</td>
</tr>
<tr>
<td>I</td>
<td>four</td>
<td>like</td>
</tr>
<tr>
<td>acted</td>
<td>years</td>
<td>someone</td>
</tr>
<tr>
<td>out</td>
<td>old</td>
<td>four</td>
</tr>
</tbody>
</table>
SHUFFLE
physically group (relocate) by key

at
four
four
four
old
old
acted
glad
I
I
am
years
years
yet
not
still
act
like
someone
out

Shuffle physically group (relocate) by key.
at = 1

four = 3

old = 2

acted = 1

glad = 1

l = 3

am = 1

years = 2

yet = 1

not = 1

still = 1

act = 1

like = 1

someone = 1

out = 1
<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>act</td>
<td>1</td>
</tr>
<tr>
<td>acted</td>
<td>1</td>
</tr>
<tr>
<td>at</td>
<td>1</td>
</tr>
<tr>
<td>am</td>
<td>1</td>
</tr>
<tr>
<td>four</td>
<td>3</td>
</tr>
<tr>
<td>I</td>
<td>3</td>
</tr>
<tr>
<td>glad</td>
<td>1</td>
</tr>
<tr>
<td>l</td>
<td>3</td>
</tr>
<tr>
<td>like</td>
<td>1</td>
</tr>
<tr>
<td>not</td>
<td>1</td>
</tr>
<tr>
<td>old</td>
<td>2</td>
</tr>
<tr>
<td>someone</td>
<td>1</td>
</tr>
<tr>
<td>still</td>
<td>1</td>
</tr>
<tr>
<td>years</td>
<td>2</td>
</tr>
<tr>
<td>yet</td>
<td>1</td>
</tr>
</tbody>
</table>
GREP.JAVA
package org.apache.hadoop.examples;

import java.util.Random;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred *
import org.apache.hadoop.mapred.lib *
import org.apache.hadoop.util.Tool
import org.apache.hadoop.util.ToolRunner;

/* Extracts matching regexs from input files and counts them. */
public class Grep extends Configured implements Tool {
    private Grep() {} // singleton

    public int run(String[] args) throws Exception {
        if (args.length < 3) {
            System.out.println("Grep <inDir> <outDir> <regex> [<group>]"");
            ToolRunner.printGenericCommandUsage(System.out);
            return -1;
        }

        Path tempDir = new Path("grep-temp-" + Integer.toString(new Random().nextInt(Integer.MAX_VALUE)));

        JobConf grepJob = new JobConf(getConf(), Grep.class);
        try {
            grepJob.setJobName("grep-search");
            FileInputFormat.setInputPaths(grepJob, args[0]);
            grepJob.setMapperClass(RegexMapper.class);
            grepJob.set("mapred.mapper.regex", args[2]);
            if (args.length == 4)
                grepJob.set("mapred.mapper.regex.group", args[3]);
            grepJob.setCombinerClass(LongSumReducer.class);
            grepJob.setReducerClass(LongSumReducer.class);
            FileOutputFormat.setOutputPath(grepJob, tempDir);
            grepJob.setOutputFormat(SequenceFileOutputFormat.class);
            grepJob.setOutputKeyClass(Text.class);
            grepJob.setOutputValueClass(LongWritable.class);
            JobClient.runJob(grepJob);

            JobConf sortJob = new JobConf(Grep.class);
            sortJob.setJobName("grep-sort");
            FileInputFormat.setInputPaths(sortJob, tempDir);
            sortJob.setInputFormat(SequenceFileInputFormat.class);
            sortJob.setMapperClass(InverseMapper.class);
            sortJob.setNumReduceTasks(1);
            FileOutputFormat.setOutputPath(sortJob, new Path(args[1]));
            sortJob.setOutputKeyComparatorClass// sort by decreasing freq
            LongWritable.DecreasingComparator.class);
            JobClient.runJob(sortJob);
        } finally {
            FileSystem.get(grepJob).delete(tempDir, true);
        }
        return 0;
    }

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new Grep(), args);
        System.exit(res);
    }
}
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.lib.InverseMapper;
import org.apache.hadoop.util.ToolRunner;

public class Grep extends Configured implements Tool {
  private Grep() {} // singleton

  public int run(String[] args) throws Exception {
    if (args.length < 3) {
      System.out.println("Grep <inDir> <outDir> <regex> [<group>]");
      ToolRunner.printGenericCommandUsage(System.out);
      return -1;
    }

    Path tempDir =
        new Path("grep-temp-" +
            Integer.toString(new Random().nextInt(Integer.MAX_VALUE)));

    JobConf grepJob = new JobConf(getConf(), Grep.class);

    try {
      grepJob.setJobName("grep-search");

      FileInputFormat.setInputPaths(grepJob, args[0]);

      grepJob.setMapperClass(RegexMapper.class);
      grepJob.setReducerClass(LongSumReducer.class);
      FileOutputFormat.setOutputPath(grepJob, tempDir);
      grepJob.setOutputFormat(SequenceFileOutputFormat.class);
      grepJob.setOutputKeyClass(Text.class);
      grepJob.setOutputValueClass(LongWritable.class);

      JobClient.runJob(grepJob);

      JobConf sortJob = new JobConf(Grep.class);
      sortJob.setJobName("grep-sort");
      FileInputFormat.setInputPaths(sortJob, tempDir);
      sortJob.setInputFormat(SequenceFileInputFormat.class);
      sortJob.setMapperClass(InverseMapper.class);
      sortJob.setNumReduceTasks(1); // write a single file
      FileOutputFormat.setOutputPath(sortJob, new Path(args[1]));
      sortJob.setOutputKeyComparatorClass(LongWritable.DecreasingComparator.class);

      JobClient.runJob(sortJob);
    }
    finally {
      FileSystem.get(grepJob).delete(tempDir, true);
    }

    return 0;
  }

  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new Configuration(), new Grep(), args);
    System.exit(res);
  }
}
import java.util.Random;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.lib.SimpleTokenizer;
import org.apache.hadoop.mapred.lib.InverseMapper;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

/**
 * Extracts matching regexs from input files and counts them.
 */
public class Grep extends Configured implements Tool {
  private Grep() {}

  // singleton
  public int run(String[] args) throws Exception {
    if (args.length < 3) {
      System.out.println("Grep <inDir> <outDir> <regex> [ <group>]\n");
      ToolRunner.printGenericCommandUsage(System.out);
      return -1;
    }

    Path tempDir = new Path("grep-temp-" + Integer.toString(new Random().nextInt(Integer.MAX_VALUE)));
    JobConf grepJob = new JobConf(getConf(), Grep.class);
    try {
      grepJob.setJobName("grep-search");
      FileInputFormat.setInputPaths(grepJob, args[0]);
      grepJob.setMapperClass(RegexMapper.class);
      grepJob.set("mapred.mapper.regex", args[2]);
      if (args.length == 4)
        grepJob.set("mapred.mapper.regex.group", args[3]);

      grepJob.setCombinerClass(LongSumReducer.class);
      grepJob.setReducerClass(LongSumReducer.class);

      FileOutputFormat.setOutputPath(grepJob, tempDir);
      grepJob.setOutputFormat(SequenceFileOutputFormat.class);
      grepJob.setOutputKeyClass(Text.class);
      grepJob.setOutputValueClass(LongWritable.class);
      JobClient.runJob(grepJob);

      JobConf sortJob = new JobConf(Grep.class);
      sortJob.setJobName("grep-sort");
      FileInputFormat.setInputPaths(sortJob, tempDir);
      sortJob.setInputFormat(SequenceFileInputFormat.class);
      sortJob.setMapperClass(InverseMapper.class);
      sortJob.setNumReduceTasks(1);
      // write a single file
      FileOutputFormat.setOutputPath(sortJob, new Path(args[1]));
      sortJob.setOutputKeyComparatorClass(LongWritable.DecreasingComparator.class);
      JobClient.runJob(sortJob);
    } finally {
      FileSystem.get(grepJob).delete(tempDir, true);
    }
    return 0;
  }

  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new Configuration(), new Grep(), args);
    System.exit(res);
  }
}
/* Extracts matching regexs from input files and counts them. */
public class Grep extends Configured implements Tool {

    private Grep() {} // singleton

    public int run(String[] args) throws Exception {
        if (args.length < 3) {
            System.out.println("Grep <inDir> <outDir> <regex> [<group>]\n");
            ToolRunner.printGenericCommandUsage(System.out);
            return -1;
        }

        Path tempDir = new Path("grep-temp-" + new Random().nextInt(Integer.MAX_VALUE));

        JobConf grepJob = new JobConf(getConf(), Grep.class);
        try {
            String[] inputPaths = args[0].split(FileSplit.SPLIT_SEPARATOR);
            FileInputFormat.setInputPaths(grepJob, inputPaths);
            grepJob.setMapperClass(RegexMapper.class);
            grepJob.set("mapred.mapper.regex", args[2]);
            if (args.length == 4)
                grepJob.set("mapred.mapper.regex.group", args[3]);
            grepJob.setCombinerClass(LongSumReducer.class);
            grepJob.setReducerClass(LongSumReducer.class);
            FileOutputFormat.setOutputPath(grepJob, tempDir);
            grepJob.setOutputFormat(SequenceFileOutputFormat.class);
            grepJob.setOutputKeyClass(Text.class);
            grepJob.setOutputValueClass(LongWritable.class);
            JobClient.runJob(grepJob);

            JobConf sortJob = new JobConf(Grep.class);
            sortJob.setJobName("grep-sort");

            FileInputFormat.setInputPaths(sortJob, tempDir);
            sortJob.setInputFormat(SequenceFileInputFormat.class);
            sortJob.setMapperClass(InverseMapper.class);
            sortJob.setNumReduceTasks(1); // write a single file
            FileOutputFormat.setOutputPath(sortJob, new Path(args[1]));
            sortJob.setOutputKeyComparatorClass(LongWritable.DecreasingComparator.class); // sort by decreasing freq
            sortJob.setOutputKeyComparatorClass(LongWritable.DecreasingComparator.class);
            JobClient.runJob(sortJob);
        } finally {
            FileSystem.get(grepJob).delete(tempDir, true);
        }
        return 0;
    }

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new Grep(), args);
        System.exit(res);
    }
}
import java.util.Random;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.mapred.lib.*;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

package org.apache.hadoop.examples;

public class Grep extends Configured implements Tool {
  private Grep() {} // singleton

  public int run(String[] args) throws Exception {
    if (args.length < 3) {
      System.out.println("Grep <inDir> <outDir> <regex> [<group>]");
      ToolRunner.printGenericCommandUsage(System.out);
      return -1;
    }

    Path tempDir = new Path("grep-temp-" + Integer.toString(new Random().nextInt(Integer.MAX_VALUE)));
    JobConf grepJob = new JobConf(getConf(), Grep.class);
    try {
      grepJob.setJobName("grep-search");
      FileInputFormat.setInputPaths(grepJob, args[0]);
      grepJob.setMapperClass(RegexMapper.class);
      grepJob.set("mapred.mapper.regex", args[2]);
      if (args.length == 4)
        grepJob.set("mapred.mapper.regex.group", args[3]);
      grepJob.setCombinerClass(LongSumReducer.class);
      grepJob.setReducerClass(LongSumReducer.class);
      FileOutputFormat.setOutputPath(grepJob, tempDir);
      grepJob.setOutputFormat(SequenceFileOutputFormat.class);
      grepJob.setOutputKeyClass(Text.class);
      grepJob.setOutputValueClass(LongWritable.class);
      JobClient.runJob(grepJob);
      JobConf sortJob = new JobConf(Grep.class);
      sortJob.setJobName("grep-sort");
      FileInputFormat.setInputPaths(sortJob, tempDir);
      sortJob.setInputFormat(SequenceFileInputFormat.class);
      sortJob.setMapperClass(InverseMapper.class);
      sortJob.setNumReduceTasks(1); // write a single file
      FileOutputFormat.setOutputPath(sortJob, new Path(args[1]));
      sortJob.setOutputKeyComparatorClass(LongWritable.DecreasingComparator.class); // sort by decreasing freq
      JobClient.runJob(sortJob);
    } finally {
      FileSystem.get(grepJob).delete(tempDir, true);
    }
    return 0;
  }

  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new Configuration(), new Grep(), args);
    System.exit(res);
  }
}
RegExMapper.java
package org.apache.hadoop.mapred.lib;

import java.io.IOException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

/** A {@link Mapper} that extracts text matching a regular expression. */
public class RegexMapper<K> extends MapReduceBase implements Mapper<K, Text, Text, LongWritable> {
    private Pattern pattern;
    private int group;

    public void configure(JobConf job) {
        pattern = Pattern.compile(job.get("mapred.mapper.regex"));
        group = job.getInt("mapred.mapper.regex.group", 0);
    }

    public void map(K key, Text value, OutputCollector<Text, LongWritable> output, Reporter reporter) throws IOException {
        String text = value.toString();
        Matcher matcher = pattern.matcher(text);
        while (matcher.find()) {
            output.collect(new Text(matcher.group(group)), new LongWritable(1));
        }
    }
}
/** A {@link Mapper} that extracts text matching a regular expression. */

public class RegexMapper<K> extends MapReduceBase
        implements Mapper<K, Text, Text, LongWritable> {

    private Pattern pattern;
    private int group;

    public void configure(JobConf job) {
        pattern = Pattern.compile(job.get("mapred.mapper.regex"));
        group = job.getInt("mapred.mapper.regex.group", 0);
    }

    public void map(K key, Text value,
            OutputCollector<Text, LongWritable> output,
            Reporter reporter)
            throws IOException {
        String text = value.toString();
        Matcher matcher = pattern.matcher(text);
        while (matcher.find()) {
            output.collect(new Text(matcher.group(group)), new LongWritable(1));
        }
    }
}
View the Shakespeare output “part-XXXX” file

Why “part-XXXX” naming?
HAVE CODE, WILL TRAVEL

Code travels to the data

Opposite of traditional systems
STREAMING
UNIX VIA MAPREDUCE

- Any UNIX command
- Any shell-invokable script
  - Perl
  - Python
  - Ruby
UNIX VIA MAPREDUCE

- Line at a time
- Tab separator
hadoop jar contrib/streaming/
    hadoop-0.20.1-streaming.jar
    -input people.csv
    -output outputstuff
    -mapper 'cut -f 2 -d ,'
    -reducer 'uniq'
Run a streaming job
The Grid

DSLs

Tools
The Grid
MOTIVATIONS
1 Gigabyte
1 Terabyte
1 Petabyte
16 Petabytes
Near-Linear Hardware Scalability
APPLICATIONS

- **Protein folding**
  pharmaceutical research

- **Search Engine Indexing**
  walking billions of web pages

- **Product Recommendations**
  based on other customer purchases

- **Sorting**
  terabytes to petabytes in size

- **Classification**
  government intelligence
CONTEXTUAL ADS
Contextual Ads

Shopper A looking at Product X is told that 66% of other customers who bought X also bought Y. Customer B also bought Y. Customer C did not buy Y. Customer D also bought Y.
SELECT reccProd.name, reccProd.id
FROM products reccProd
WHERE purchases.customerId =
SELECT reccProd.name, reccProd.id
FROM products reccProd
WHERE purchases.customerId =

(SELECT customerId
FROM customers
WHERE purchases.productId = thisProd)

LIMIT 5
GRID BENEFITS
SCALABLE

- Data storage is pipelined
- Code travels to data
- Near linear hardware scalability
Optimized

- Preemptive execution
- Maximizes use of faster hardware
- New jobs trump P.E.
Fault Tolerant

- Configurable data redundancy
- Minimizes hardware failure impact
- Automatic job retries
- Self healing filesystem
SERVER FUNERALS

No pagers go off when machines die

Report of dead machines once a week

Clean out the carcasses
Robustness attributes prevented from bleeding into application code

- Data redundancy
- Node death
- Retries
- Data geography
- Parallelism
- Scalability
COMPONENTS
HADOOP COMPONENTS
## Hadoop Components

<table>
<thead>
<tr>
<th>Tool</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common</td>
<td>MapReduce</td>
</tr>
<tr>
<td>HDFS</td>
<td>Filesystem</td>
</tr>
<tr>
<td>Pig</td>
<td>Analyst MapReduce language</td>
</tr>
<tr>
<td>HBase</td>
<td>Column-oriented data storage</td>
</tr>
<tr>
<td>Hive</td>
<td>SQL-like language for HBase</td>
</tr>
<tr>
<td>ZooKeeper</td>
<td>Workflow &amp; distributed transactions</td>
</tr>
<tr>
<td>Chukwa</td>
<td>Log file processing</td>
</tr>
</tbody>
</table>
The Grid

DSLs

Tools
DSLs
Sync, Async

- Pig is asynchronous
- Hive is asynchronous
- HBase is near realtime
- RDBMS is realtime
PIG
Pig Basics

- Yahoo-authored add-on
- High-level language for authoring data analysis programs
- Console
Person = LOAD 'people.csv' using PigStorage('','');
Names = FOREACH Person GENERATE $2 AS name;
OrderedNames = ORDER Names BY name ASC;
GroupedNames = GROUP OrderedNames BY name;
NameCount = FOREACH GroupedNames
  GENERATE group, COUNT(OrderedNames);
store NameCount into 'names.out';
pig-example (master)> pig -x local get-names-from-people.pig
2010-04-20 05:58:52,460 [main] INFO org.apache.pig.backend.local.executionengine.LocalPigLauncher - Successfully stored result in: "file:/Users/mccm06/Documents/Teach/Courses/Hadoop-Intro/hadoop-examples.git/pig-example/names.out"
2010-04-20 05:58:52,460 [main] INFO org.apache.pig.backend.local.executionengine.LocalPigLauncher - Records written : 5158
2010-04-20 05:58:52,462 [main] INFO org.apache.pig.backend.local.executionengine.LocalPigLauncher - Bytes written : 0
2010-04-20 05:58:52,462 [main] INFO org.apache.pig.backend.local.executionengine.LocalPigLauncher - 100% complete!
2010-04-20 05:58:52,462 [main] INFO org.apache.pig.backend.local.executionengine.LocalPigLauncher - Success!!
[pig-example (master)> ]
HIVE
HIVE BASICS

 Authored by facebook

 SQL interface to HBase

 Hive is low-level

 Hive-specific metadata

 Data warehousing
SELECT * FROM shakespeare
WHERE freq > 100
SORT BY freq ASC
LIMIT 10;
SYNC, ASYNC

- **RDBMS** SQL is **realtime**
- **Hive** is primarily **asynchronous**
HBASE
Adobe - We currently have about 30 nodes running HDFS, Hadoop and HBase in clusters ranging from 5 to 14 nodes on both production and development. We plan a deployment on an 80 nodes cluster. We are using HBase in several areas from social services to structured data and processing for internal use. We constantly write data to HBase and run mapreduce jobs to process then store it back to HBase or external systems. Our production cluster has been running since Oct 2008.

Drawn to Scale Consulting consults on HBase, Hadoop, Distributed Search, and Scalable architectures.

Filmweb is a film web portal with a large dataset of films, persons and movie-related entities. We have just started a small cluster of 3 HBase nodes to handle our web cache persistency layer. We plan to increase the cluster size, and also to start migrating some of the data from our databases which have some demanding scalability requirements.

Flurry provides mobile application analytics. We use HBase and Hadoop for all of our analytics processing, and serve all of our live requests directly out of HBase on our 16-node production cluster with billions of rows over several tables.

GumGum is an in-image ad network. We use HBase 0.20 on a 4-node Amazon EC2 Large Instance (m1.large) cluster for both real-time data and analytics. Our production cluster has been running since June 2010.

Kalooga is a discovery service for image galleries. We use Hadoop, Hbase, Chukwa and Pig on a 20-node cluster for our crawling, analysis and events processing.

Lily is an open source content repository backed by HBase and SOLR from Outerthought - scalable content applications.

Mahalo, "...the world’s first human-powered search engine". All the markup that powers the wiki is stored in HBase. It's been in use for a few months now. MediaWiki - the same software that power Wikipedia - has version/revision control. Mahalo’s in-house editors produce a lot of revisions per day, which was not working well in a RDBMS. An hbase-based solution for this was built and tested, and the data migrated out of MySQL and into HBase. Right now it's at something like 6 million items in HBase. The upload tool runs every hour from a shell script to back up that data, and on 6 nodes takes about 5-10 minutes to run - and does not slow down production at all.

Meetup is on a mission to help the world’s people self-organize into local groups. We use Hadoop and HBase to power a site-wide, real-time activity feed system for all of our members and groups. Group activity is written directly to HBase, and indexed per member, with the member's custom feed served directly from HBase for incoming requests. We're running HBase 0.20.0 on a 11 node cluster.

Ning uses HBase to store and serve the results of processing user events and log files, which allows us to provide near-real time analytics and reporting. We use a small cluster of commodity machines with 4 cores and 16GB of RAM per machine to handle all our analytics and reporting needs.
Stumbleupon and Su.pr use HBase as a real time data storage and analytics platform. Serving directly out of HBase, various site features and statistics are kept up to date in a real time fashion. We also use HBase a map-reduce data source to overcome traditional query speed limits in MySQL.

SubRecord Project is an Open Source project that is using HBase as a repository of records (persisted map-like data) for the aspects it provides like logging, tracing or metrics. HBase and Lucene index both constitute a repo/storage for this platform.

Shopping Engine at Tokenizer is a web crawler; it uses HBase to store URLs and Outlinks (AnchorText + LinkedURL): more than a billion. It was initially designed as Nutch-Hadoop extension, then (due to very specific 'shopping' scenario) moved to SOLR + MySQL(InnoDB) (ten thousands queries per second), and now - to HBase. HBase is significantly faster due to: no need for huge transaction logs, column-oriented design exactly matches 'lazy' business logic, data compression, MapReduce support. Number of mutable 'indexes' (term from RDBMS) significantly reduced due to the fact that each 'row::column' structure is physically sorted by 'row'. MySQL InnoDB engine is best DB choice for highly-concurrent updates. However, necessity to flash a block of data to harddrive even if we changed only few bytes is obvious bottleneck. HBase greatly helps: not-so-popular in modern DBMS 'delete-insert', 'mutable primary key', and 'natural primary key' patterns become a big advantage with HBase.

Trend Micro uses HBase as a foundation for cloud scale storage for a variety of applications. We have been developing with HBase since version 0.1 and production since version 0.20.0.

Twitter runs HBase across its entire Hadoop cluster. HBase provides a distributed, read/write backup of all mysql tables in Twitter's production backend, allowing engineers to run MapReduce jobs over the data while maintaining the ability to apply periodic row updates (something that is more difficult to do with vanilla HDFS). A number of applications including people search rely on HBase internally for data generation. Additionally, the operations team uses HBase as a timeseries database for cluster-wide monitoring/performance data.

Veoh Networks uses HBase to store and process visitor(human) and entity(non-human) profiles which are used for behavioral targeting, demographic detection, and personalization services. Our site reads this data in real-time (heavily cached) and submits updates via various batch map/reduce jobs. With 25 million unique visitors a month storing this data in a traditional RDBMS is not an option. We currently have a 24 node Hadoop/HBase cluster and our profiling system is sharing this cluster with our other Hadoop data pipeline processes.

VideoSurf - "The video search engine that has taught computers to see". We're using Hbase to persist various large graphs of data and other statistics. Hbase was a real win for us because it let us store substantially larger datasets without the need for manually partitioning the data and it's column-oriented nature allowed us to create schemas that were substantially more efficient for storing and retrieving data.

Visible Technologies - We use Hadoop, HBase, Katta, and more to collect, parse, store, and search hundreds of millions of Social Media content. We get incredibly fast throughput and very low latency on commodity hardware. HBase enables our business to exist.

WorldLingo - The WorldLingo Multilingual Archive. We use HBase to store millions of documents that we scan using Map/Reduce jobs to machine translate them into all or selected target languages from our set of available machine translation languages. We currently store 12 million documents but plan to eventually reach the 450 million mark. HBase allows us to scale out as we need to grow our storage capacities. Combined with Hadoop to keep the data replicated and therefore fail-safe we have the backbone our service can rely on now and in the future. WorldLingo is using HBase since December 2007 and is along with a few others one of the longest running HBase installation. Currently we are running the latest HBase 0.20 and serving directly from it: MultilingualArchive.
Sumbleupon and Su.pr use HBase as a real time data storage and analytics platform. Serving directly out of HBase, various site features and statistics are kept up to date in a real time fashion. We also use HBase a map-reduce data source to overcome traditional query speed limits in MySQL.

SubRecord Project is an Open Source project that is using HBase as a repository of records (persisted map-like data) for the aspects it provides like logging, tracing or metrics. HBase and Lucene index both constitute a repo/storage for this platform.

Shopping Engine at Tokenizer is a web crawler; it uses HBase to store URLs and Outlinks (AnchorText + LinkedURL): more than a billion. It was initially designed as Nutch-Hadoop extension, then (due to very specific 'shopping' scenario) moved to SOLR + MySQL(InnoDB) (ten thousands queries per second), and now - to HBase. HBase is significantly faster due to: no need for huge transaction logs, column-oriented design exactly matches 'lazy' business logic, data compression, MapReduce support. Number of mutable 'indexes' (term from RDBMS) significantly reduced due to the fact that each 'row::column' structure is physically sorted by 'row'. MySQL InnoDB engine is best DB choice for highly-concurrent updates. However, necessity to flash the block of data to harddrive even if we changed only few bytes is obvious bottleneck. HBase greatly helps: not-so-popular in modern DBMS 'delete-insert', 'mutable primary key', and 'natural primary key' patterns become a big advantage with HBase.

Trend Micro uses HBase as a foundation for cloud scale storage for a variety of applications. We have been developing with HBase since version 0.1 and production since version 0.20.0.

Twitter runs HBase across its entire Hadoop cluster. HBase provides a distributed, read/write backup of all mysql tables in Twitter's production backend, allowing engineers to run MapReduce jobs over the data while maintaining the ability to apply periodic row updates (something that is more difficult to do with vanilla HDFS). A number of applications including people search rely on HBase internally for data generation. Additionally, the operations team uses HBase as a timeseries database for cluster-wide monitoring/performance data.

Veoh Networks uses HBase to store and process visitor(human) and entity(non-human) profiles which are used for behavioral targeting, demographic detection, and personalization services. Our site reads this data in real-time (heavily cached) and submits updates via various batch map/reduce jobs. With 25 million unique visitors a month storing this data in a traditional RDBMS is not an option. We currently have a 24 node Hadoop/HBase cluster and our profiling system is sharing this cluster with our other Hadoop data pipeline processes.

VideoSurf - "The video search engine that has taught computers to see". We're using Hbase to persist various large graphs of data and other statistics. Hbase was a real win for us because it let us store substantially larger datasets without the need for manually partitioning the data and it's column-oriented nature allowed us to create schemas that were substantially more efficient for storing and retrieving data.

Visible Technologies - We use Hadoop, HBase, Katta, and more to collect, parse, store, and search hundreds of millions of Social Media content. We get incredibly fast throughput and very low latency on commodity hardware. HBase enables our business to exist.

WorldLingo - The WorldLingo Multilingual Archive. We use HBase to store millions of documents that we scan using Map/Reduce jobs to machine translate them into all or selected target languages from our set of available machine translation languages. We currently store 12 million documents but plan to eventually reach the 450 million mark. HBase allows us to scale out as we need to grow our storage capacities. Combined with Hadoop to keep the data replicated and therefore fail-safe we have the backbone our service can rely on now and in the future. WorldLingo is using HBase since December 2007 and is along with a few others one of the longest running HBase installation. Currently we are running the latest HBase 0.20 and serving directly from it: MultilingualArchive.
HBASE BASICS

- Map-oriented storage
- Key value pairs
- Column families
- Stores to HDFS
- Fast
- Usable for synchronous responses
hbase>
help

create 'mylittletable', 'mylittlecolumnfamily'
describe 'mylittletable'

put 'mylittletable', 'r2', 'mylittlecolumnfamily', 'x'

get 'mylittletable', 'r2'
scan 'mylittletable'
Sqoop is a tool designed to import data from relational databases into Hadoop. Sqoop uses JDBC to connect to a database. It examines each table’s schema and automatically generates the necessary classes to import data into the Hadoop Distributed File System (HDFS). Sqoop then creates and launches a MapReduce job to read tables from the database via DBInputFormat, the JDBC–based InputFormat. Tables are read into a set of files in HDFS. Sqoop supports both SequenceFile and text–based target and includes performance enhancements for loading data from MySQL.

Getting Sqoop
Sqoop is an open-source program contributed to the Apache Hadoop project. The most recent release of Cloudera’s Distribution for Hadoop contains Sqoop.

If you’re already using our RPM packages, running the command `yum update hadoop` should bring you up to date. Debian users can do the same with `apt-get update`.

Instructions
The instructions below describe how to get started using Sqoop to import your data into Hadoop:

- Example Usage
- Connecting to a Database Server
- Listing Available Tables
Sqoop

A cloudera utility

- Imports from RDBMS
- Outputs plaintext, SequenceFile, or Hive
sqoop --connect jdbc:mysql://database.example.com/
Tools
MONITORING
WEB STATUS PANELS

- **NameNode**
  http://localhost:50070/

- **JobTracker**
  http://localhost:50030/
localhost Hadoop Map/Reduce Administration

State: RUNNING
Started: Fri Apr 30 09:49:45 EDT 2010
Version: 0.20.1, r810220
Compiled: Tue Sep 1 20:55:56 UTC 2009 by oom
Identifier: 201004300949

Cluster Summary (Heap Size is 80.56 MB/994.81 MB)

<table>
<thead>
<tr>
<th>Maps</th>
<th>Reduces</th>
<th>Total Submissions</th>
<th>Nodes</th>
<th>Map Task Capacity</th>
<th>Reduce Task Capacity</th>
<th>Avg. Tasks/Node</th>
<th>Blacklisted Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4.00</td>
<td>0</td>
</tr>
</tbody>
</table>

Scheduling Information

<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Scheduling Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Filter (Jobid, Priority, User, Name)

Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

none

Completed Jobs
EXTENDED FAMILY MEMBERS
Cascading
ANOTHER DSL

- Java-based
- Different vocabulary
- Abstraction from MapReduce
- Uses Hadoop
- Cascalog for Clojure
Welcome

Cascading is a Query API and Query Planner used for defining and executing complex, scale-free, and fault tolerant data processing workflows on a Hadoop cluster.

Cascading is a thin Java library that sits on top of Hadoop's MapReduce layer. It is not a new text based query syntax (like Pig) or another complex system that must be installed on a cluster and maintained (like Hive). Though Cascading is both complimentary to and is a valid alternative to either application.

Cascading is simply a query processing API that lets the developer quickly assemble complex distributed processes without having to "think" in MapReduce. And to efficiently schedule them based on their dependencies. Obviously similar to data processing applications are supported as well, as complex applications tend to start simple.

Cascading is Open Source and dual licensed under the GPL and OEM/Commercial Licenses. OEM/Commercial Licenses and Developer Support can be obtained through Concurrent, Inc.

Cascading has a strong community of users and contributors, see our Cascading modules page for related projects and extensions.

Cascading, extensions, and related libraries are also hosted in the Conjar maven repository maintained by Concurrent, Inc. The repository is open to the public.

Read more about Cascadings features or thumb through the Cascading User Guide.

Recent Events

Memcached, Membase, and ElasticSearch Integration

September 22, 2010 12:40 PM | Permalink

We have added a link to the Cascading.Memcached project on GitHub to the Modules and Extensions page.

This sub-project provides Memcached API integration allowing Cascading Flows to push data into various memcached API compliant applications like ElasticSearch and Membase.
Introducing Cascalog: a Clojure-based query language for Hadoop

I’m very excited to be releasing Cascalog as open-source today. Cascalog is a Clojure-based query language for Hadoop inspired by Datalog.

Highlights

- **Simple** - Functions, filters, and aggregators all use the same syntax. Joins are implicit and natural.
- **Expressive** - Logical composition is very powerful, and you can run arbitrary Clojure code in your query with little effort.
- **Interactive** - Run queries from the Clojure REPL.
- **Scalable** - Cascalog queries run as a series of MapReduce jobs.
- **Query anything** - Query HDFS data, database data, and/or local data by making use of Cascading’s "Tap" abstraction.
- **Careful handling of null values** - Null values can make life difficult. Cascalog has a feature called "non-nullable variables" that makes dealing with nulls trivial.
Karmasphere
HADOOP STUDIO

- Free and commercial offerings
- Workflow designer
- IDE based testing
Cloudera Desktop
Cloudera Desktop

A unified user interface for users and operators of Hadoop clusters.

Cloudera Desktop is a graphical user interface for the many tools required to operate and develop for Hadoop. These tools are collected into a desktop environment and delivered as a Web application that simplifies cluster administration and job development. With Cloudera Desktop, the world-class performance and scalability of Apache Hadoop is now accessible to anyone in your organization.
GRID ON THE CLOUD
ELASTIC MAP REDUCE

- Cluster MapReduce by the hour
- Easiest grid to set up
- Storage on S3
## EMR Pricing

<table>
<thead>
<tr>
<th></th>
<th>US – N. Virginia</th>
<th>US – N. California</th>
<th>EU – Ireland</th>
<th>Amazon Elastic MapReduce Price per hour</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standard On-Demand Instances</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small (Default)</td>
<td>$0.085 per hour</td>
<td></td>
<td>$0.015 per hour</td>
<td></td>
</tr>
<tr>
<td>Large</td>
<td>$0.34 per hour</td>
<td></td>
<td>$0.06 per hour</td>
<td></td>
</tr>
<tr>
<td>Extra Large</td>
<td>$0.68 per hour</td>
<td></td>
<td>$0.12 per hour</td>
<td></td>
</tr>
<tr>
<td><strong>High Memory On-Demand Instances</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m2.2xlarge</td>
<td>$1.20 per hour</td>
<td></td>
<td>$0.21 per hour</td>
<td></td>
</tr>
<tr>
<td>m2.4xlarge</td>
<td>$2.40 per hour</td>
<td></td>
<td>$0.42 per hour</td>
<td></td>
</tr>
<tr>
<td><strong>High CPU On-Demand Instances</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td>$0.17 per hour</td>
<td></td>
<td>$0.03 per hour</td>
<td></td>
</tr>
<tr>
<td>Extra Large</td>
<td>$0.68 per hour</td>
<td></td>
<td>$0.12 per hour</td>
<td></td>
</tr>
</tbody>
</table>
SUMMARY
SAPIR-WHORF HYPOTHESIS...
REMIXED
SAPIR-WHORF HYPOTHESIS... REMIXED

The **ability** to store and process **massive data** influences what you decide to store.
In the next several years, the scale of data problems we are aiming to solve will grow near-exponentially.
HADOOP

divide and conquer petabyte-scale data
REFERENCES
REFERENCES

- http://wiki.apache.org/hadoop/Hbase/PoweredBy
- http://mahout.apache.org/
- http://mahout.apache.org/taste.html
- http://www.cascading.org/
REFERENCES

- http://hadoop.apache.org
- http://delicious.com/matthew.mccullough/hadoop
- http://code.google.com/edu/parallel/
- http://www.cloudera.com/