

THE #1 PROGRAMMER EXCUSE  
FOR LEGITIMATELY SLACKING OFF:

"MY CODE'S COMPILING."

HEY! GET BACK  
TO WORK!

COMPILING!

OH. CARRY ON.



# Hadoop: Apache's Open Source Implementation of Google's MapReduce Framework

Hacked Existence Team

Joey Calca & Ryan Anguiano

<http://hackedexistence.com>

# Cloud Computing

- Clouds are big piles of other people's machines, plus virtualization
- Remote
- Scalable
- Virtual
- High Level API
- Course Grain data processed in parallel

# How much data?

- Wayback Machine has 2 PB + 20 TB/month (2006)
- Google processes 20 PB a day (2008)
- “all words ever spoken by human being” ~ 5 EB
- NOAA has ~ 1PB climate data (2007)
- CERN’s :HC will generate 15 PB a year (2008)

Stats from The iSchool University of Maryland



# Saguaro Cluster

Research Group	High Performance Computing Initiative
Department	Fulton School
Primary Application	Various
# of Processor Cores	4560
Processor Architecture	Intel Xeon
Interconnect	InfiniBand
Memory	10240 GB (Total)
Storage	215 TB
OS	CentOS 5.3
Sys Admin Contact	Douglas Fuller

# Google's Map/Reduce

- Google 2004 at The Sixth Symposium on Operating System Design and Implementation
- Processing and Generating large data sets
- Many real world tasks are expressible in this model
- Automatically parallelized for a large cluster of commodity machines

# Google's Map/Reduce

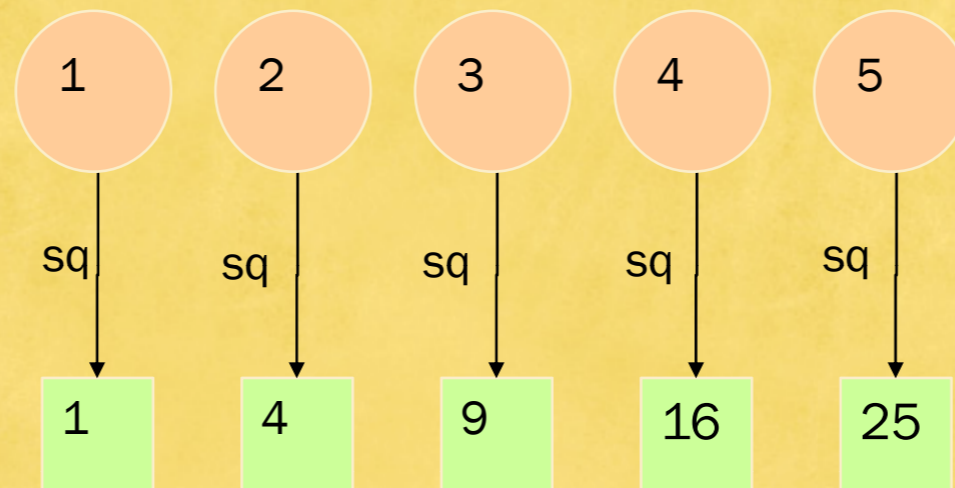
- Input -> Mapper -> Intermediate <key/value> Pairs -> Reducer -> Output
- Easy to utilize resources of large distributed system without any experience
- Highly scalable: typically processes many terabytes of data in parallel
- Upwards of 1,000 MapReduce jobs are executed on Googles clusters daily



- Apache Project's Open Source Implementation of MapReduce
- JAVA Based
- Hadoop has been demonstrated on clusters with 2000 nodes. The current design target is 10,000 node clusters.
- <http://hadoop.apache.org>

# Mapper

- Map is a special function that applies the function  $f$  to each element in the list
- $\text{Map}[f,(1,2,3,4,5)] \rightarrow \{f[1],f[2],f[3],f[4],f[5]\}$



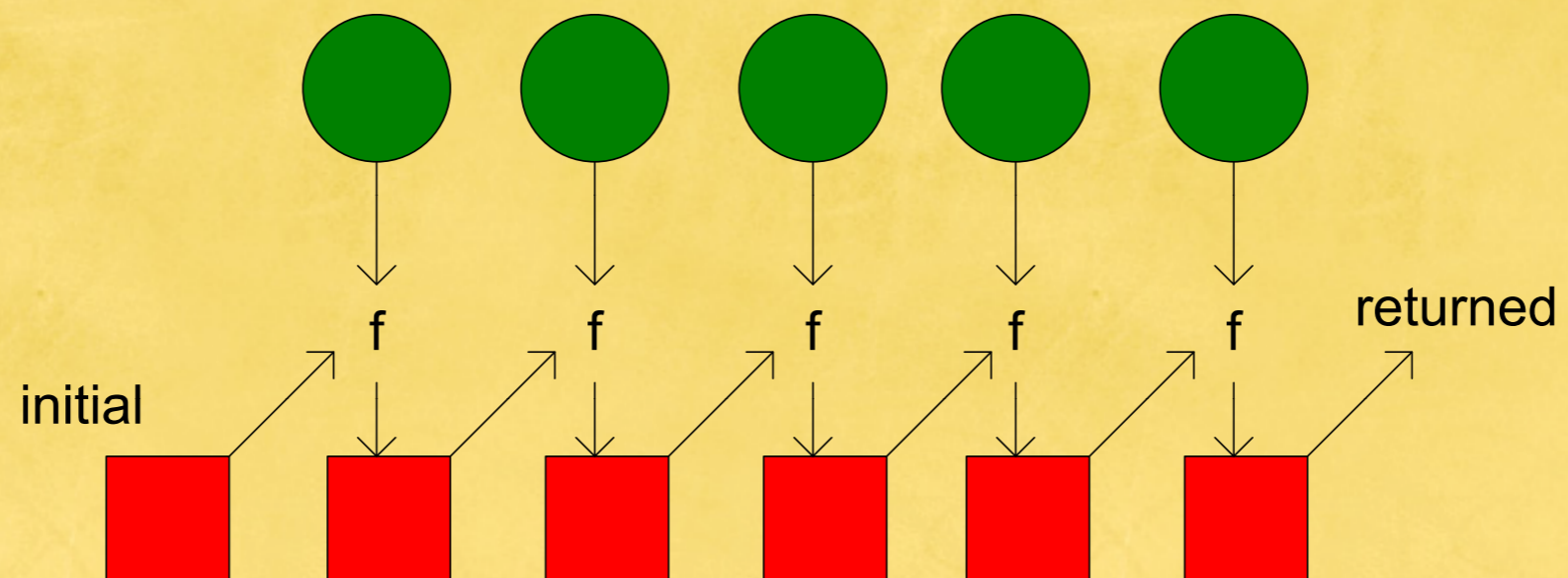
# Mapper

- Input:
  - The Entire Data Set
  - Maps all the input values to a key
  - `map()` is called once for each line of input
- Output
  - Collects `<key, value>` pairs
  - Passes to reducer as hashmap

# Reducer

- `Reduce[f,x,list]`
- Sets an accumulator
- Initial value is `x`
- Applies `f` to each element of the list plus the accumulator
- Result is the final value of the accumulator
- `Reduce[f,x,{a,b,c}] => f[f[f[x,a],b],c]`

# Reducer



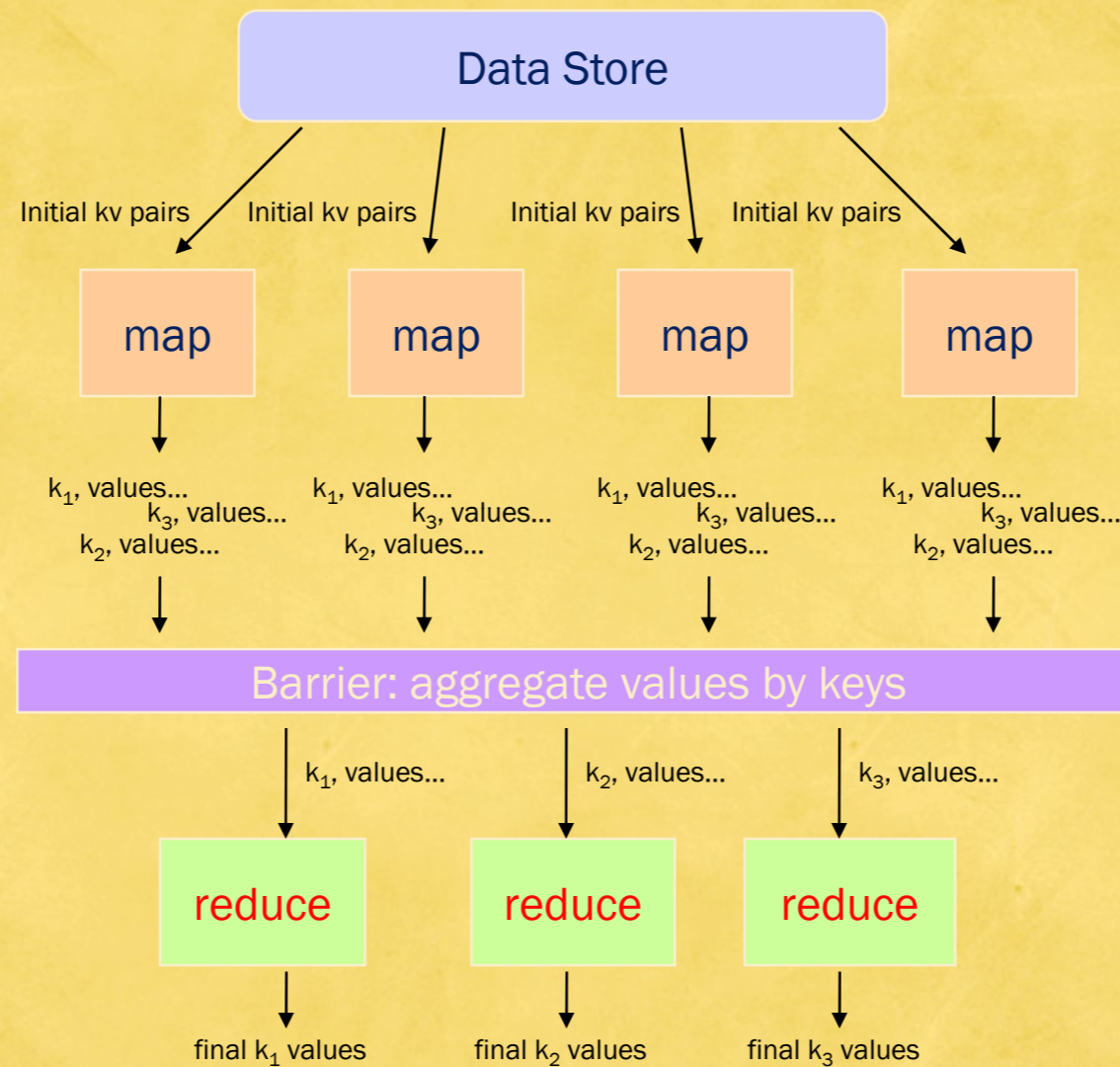
# Reducer

- Input
  - The output  $\langle KV \rangle$  hashmap from the mapper
  - $f(x)$  is performed on every  $x$  with a common key
- Output
  - A  $\langle KV \rangle$  list of the output of `reduce()`

# Map/Reduce Framework

- Map is implicitly parallel
- Order of application of function does not matter
- Reduce is executed in serial on a single node
- The results of map() are copied and sorted then sent to the reduce()

# Map/Reduce Framework



# Map/Reduce Framework

- Programmer does not have to handle:
  - Work distribution
  - Scheduling
  - Networking
  - Synchronization
  - Fault recovery (if a map or reduce node fails)
  - Moving data between nodes

# Master Node

- Assigns tasks and data to each node
- Hosts an http JobTracker on port 50030
- Queries each node
- Kills any task that does not respond
- Re-Batches killed tasks out to next available node

# Streaming

- Ability to port mappers and reducers to any language that is executable on each node
- Input is read from `stdin()`
- ```
def read_input(file):  
    for line in file:  
        yield line.rstrip()
```

# Streaming

- Output is a hashmap, which is a string in the form:  
<Key (tab) Value>
- Output is written to stdout()
- `print "%s\t%s" % (key, value)`

# Streaming

- The utility packages all executables into a single JAR
- JAR is sent to all nodes
- Distributed Cache files are symlinked to the current working directory

# Streaming

```
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/contrib/streaming/hadoop-0.19.0-streaming.jar \  
-input inputDirs \  
-output outputDir \  
-mapper mapperExecutable \  
-reducer reducerExecutable \  
-file PathOfFilesToBePackaged \  
-cacheFile 'hdfs://pathToFile#symlink' \  
-jobconf mapred.job.name='jobName'  
  
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/contrib/streaming/hadoop-0.19.0-streaming.jar \  
-input /datasets/Netflix-dataset/training_set_reorg/* \  
-output pyNetflix1Output \  
-mapper pyMapper.py \  
-reducer pyReducer.py \  
-file /home/ranguiano/workspace/pyNetflix1/pyMapper.py \  
-file /home/ranguiano/workspace/pyNetflix1/pyReducer.py \  
-cacheFile 'hdfs://s49-1.local:9001/datasets/Netflix-dataset/movie_titles.txt#movie_titles.txt' \  
-jobconf mapred.job.name='pyNetflix1'
```

# Streaming

- `-mapper` and `-reducer` can be set to a java class or any file that can be executed locally
- Files and/or Archives can be distributed to each node or to distributed cache

# Reporting

- Stdin/Stdout used for data, Stderr used for communication to Master Node
- Counter must be reported after every output line to track job progress  
report:counter:pyNetflixI,mapper,I
- Status messages can be used to track errors in log files  
report:status:Movie not found

# HDFS

- Hadoop Distributed File System (HDFS) - Google uses GoogleFileSystem (GFS)
- High fault-tolerant, low cost hardware
- High throughput, streaming access to data
- Data is split on 64 meg blocks and replicated in storage



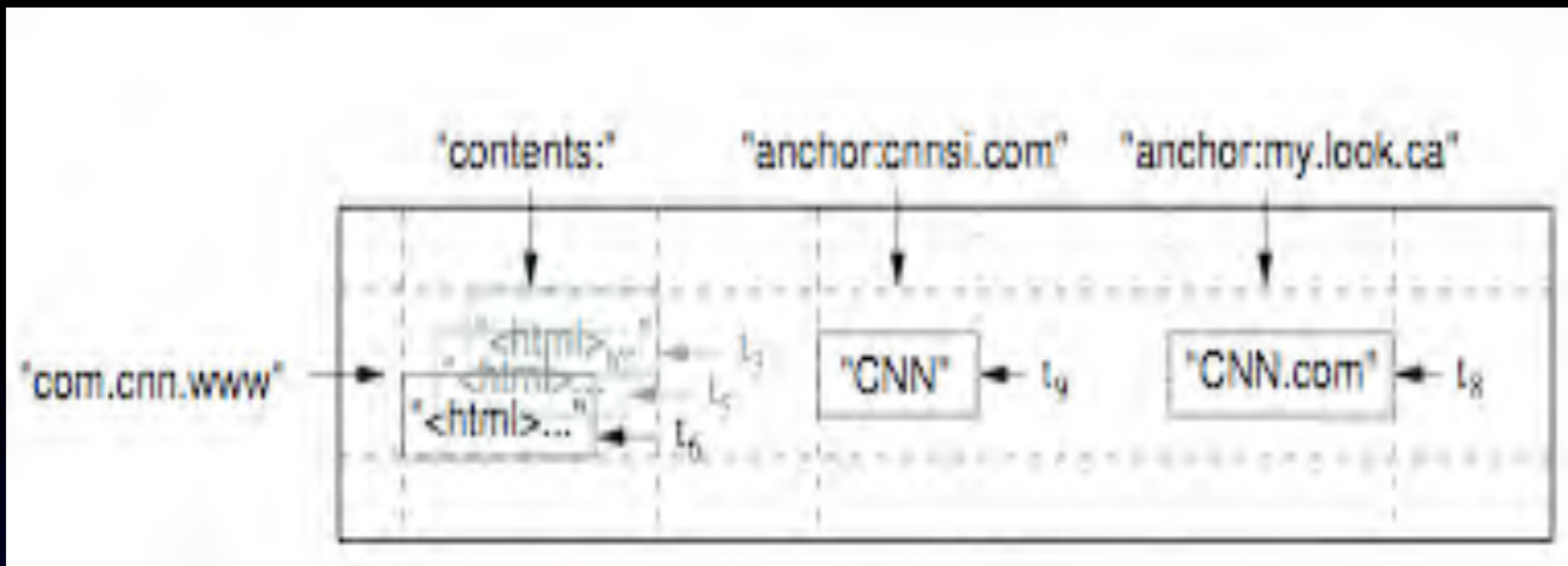
- HBase is equivalent to Google's BigTable
- NON-RELATIONAL DATABASE
- Is not built for real-time querying
- Moving away from per-user actions
- Towards per-action data sets



- Distributed
- Multi-dimensional
- De-Normalized Data
- HBase is not an SQL Database

# HBase Tables

- Table Schema defines Column Families
- Column Family contains multiple Columns
- Each Column has Versions (Z-axis)
- Everything except table name stored as `byte[]`



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

\*Taken from HBase Documentation

# Amazon's Elastic Compute Cloud (EC2)

- Web service that provides resizable compute capacity in Amazon's Cloud.
- Hadoop is packaged as a public EC2 image (an AMI) so it is easy for us to get up and running with a cluster.
- `ec2-describe-images -a | grep hadoop-ec2-images`
- Extremely simple to setup an elastic hadoop cloud
- <http://aws.amazon.com/ec2/>

# Amazon's Pricing

## EC2

| Standard On-Demand Instances | Linux/UNIX Usage | Windows Usage    |
|------------------------------|------------------|------------------|
| Small (Default)              | \$0.10 per hour  | \$0.125 per hour |
| Large                        | \$0.40 per hour  | \$0.50 per hour  |
| Extra Large                  | \$0.80 per hour  | \$1.00 per hour  |
| High CPU On-Demand Instances | Linux/UNIX Usage | Windows Usage    |
| Medium                       | \$0.20 per hour  | \$0.30 per hour  |
| Extra Large                  | \$0.80 per hour  | \$1.20 per hour  |

## S3 (Amazon's Simple Storage Service)

### Storage

- \$0.150 per GB – first 50 TB / month of storage used
- \$0.140 per GB – next 50 TB / month of storage used
- \$0.130 per GB – next 400 TB / month of storage used
- \$0.120 per GB – storage used / month over 500 TB

### Data Transfer

- \$0.170 per GB – first 10 TB / month data transfer out
- \$0.130 per GB – next 40 TB / month data transfer out
- \$0.110 per GB – next 100 TB / month data transfer out
- \$0.100 per GB – data transfer out / month over 150 TB

### Requests

- \$0.01 per 1,000 PUT, COPY, POST, or LIST requests
- \$0.01 per 10,000 GET and all other requests\*

\* No charge for delete requests

# Netflix Prize

## 2 GB dataset of movie/user/ratings

Training\_set1.txt...Training\_set17770.txt:

- MovieIDs range from 1 to 17770 sequentially.
- CustomerIDs range from 1 to 2649429, with gaps. There are 480189 users.
- Ratings are on a five star scale from 1 to 5.
- Dates have the format YYYY-MM-DD.

```
1: [Movie 1 of 17770]
[CustomerID,Rating,Date]
1116, 3, 2006-04-17
2, 5, 2007-07-07
```

## Netflix Prize

- Default input dataset creates one mapper per file
- Inefficient when dealing with 17,770 files
- Need to optimize # of files to the number of mappers available
- Awk script used to reorganize input dataset into 104 files to be used on 100 procs
- Insures that all mappers are being utilized while optimizing file I/O

# Netflix Prize

netflixReorg.awk:

```
# tokenize on ":"
```

```
BEGIN { FS = ":" }
```

```
# if it is the first line, movieID = first token
```

```
{if( FNR == 1 ) movieID = $1
```

```
# if it is not the first line,
```

```
output movieID "," first token
```

```
if ( FNR != 1 ) print movieID "," $1 }
```

# Netflix Prize

- Efficiency gained by reorganizing input dataset
- Netflix I - 43:27
- Netflix I Reorg - 9:55
- pyNetflix I - 13:02
- awkNetflix I - 9:04

# Netflix I Program

- Produce statistical information about each movie in the dataset
- It took the entire Netflix dataset as input
- Produced the first date rated, last date rated, total rating count and average rating for each movie as the output

# Netflix I Mapper

- Input: Netflix Prize Training Set
- output:  $\langle \text{movieID}, \text{ratingAndDateRated} \rangle$
- one  $\langle K, V \rangle$  pair for each movieID in the input data set

# Netflix I Mapper Code

- `Netflix I / MyMapper.java`

# pyNetflix I Mapper Code

- `pyNetflix I /pyMapper.py`

# awkNetflix | Mapper Code

- awkNetflix | /awkMapper.awk

# Mapper Comparison

| Netflix I | Java                       | Python                           | Awk                        |
|-----------|----------------------------|----------------------------------|----------------------------|
| Map Task  | Best: 8 sec<br>Avg: 12 sec | Best: 27 sec<br>Avg: 1 min 5 sec | Best: 9 sec<br>Avg: 15 sec |

# Netflix2 Reducer

- The Netflix2 program calculates statistics based on the users in the dataset
- Netflix2 Mapper output
- `<userID, movieID : rating : dateRated>`
- Netflix2 Reducer output
- `<userID, ratingCount : avgRating : ratingDelay : movieRatingDateList >`

# Netflix2 Reducer Code

- `Netflix2/MyReducer.java`

# pyNetflix2 Reducer Code

- `pyNetflix2/pyReducer.py`

# Reducer Comparison

| Netflix 2   | Java         | Python       |
|-------------|--------------|--------------|
| Reduce Task | 2 min 58 sec | 8 min 45 sec |

# Shoutouts



- Dr. Adrian Sannier - University Technology Officer
- Dr. Dan Stanzione Jr. - Director of High Performance Computing Initiative
- Dr. Raghu Santanam - Associate Professor
- Nathan Kerr and Jeff Conner

# Thank you

Joey Calca  
r3dfish@

hackedexistence.com

Ryan Anguiano  
bl4ckbird@

hackedexistence.com

<http://hackedexistence.com>

## Questions?