0xdata

H2O on Hadoop

July 22, 2013

H2O on Hadoop

Introduction

H2O is the open source math & machine learning engine for big data that brings distribution and parallelism to powerful algorithms while keeping the widely used languages of R and JSON as an API. H2O brings and elegant lego-like infrastructure that brings fine-grained parallelism to math over simple distributed arrays. Customers can use data locked in HDFS as a data source. H2O is a primary citizen of the Hadoop infrastructure & interacts naturally with the Hadoop JobTracker & TaskTrackers on all major distros.

H2O is 0xdata's math-on-big-data framework. H2O is open source under the Apache 2.0 license. See http://0xdata.com for more information about what H2O does and how to get it.

This whitepaper is appropriate for you if your organization has already made an investment in a Hadoop cluster, and you want to use Hadoop to launch and monitor H2O jobs.

Glossary

0xdata	Maker of H2O. Visit our website at http://0xdata.com.
H2O	H2O makes Hadoop do math. H2O is an Apache v2 licensed open source math and prediction engine.
Hadoop	An open source big-data platform. Cloudera, MapR, and Hortonworks are distro providers of Hadoop. Data is stored in HDFS (DataNode, NameNode) and processed through MapReduce and managed via JobTracker.
H2O node	H2O nodes are launched via Hadoop MapReduce and run on Hadoop DataNodes. (At a system level, an H2O node is a Java invocation of h2o.jar.) Note that while Hadoop operations are centralized around HDFS file accesses, H2O operations are memory-based when possible for best performance. (H2O reads the dataset from HDFS into memory and then attempts to perform all operations to the data in memory.)
H2O cluster	A group of H2O nodes that operate together to work on jobs. H2O scales by distributing work over many H2O nodes. (Note multiple H2O nodes can run on a single Hadoop node if sufficient resources are available.) All H2O nodes in an H2O cluster are peers. There is no "master" node.

Spilling	An H2O node may choose to temporarily "spill" data from memory onto disk. (Think of this like swapping.) In Hadoop environments, H2O spills to HDFS. Usage is intended to function like a temporary cache, and the spilled data is discarded when the job is done.
H2O Key,Value	H2O implements a distributed in-memory Key/Value store within the H2O cluster. H2O uses Keys to uniquely identify data sets that have been read in (pre-parse), data sets that have been parsed (into HEX format), and models (e.g. GLM) that have been created. For example, when you ingest your data from HDFS into H2O, that entire data set is referred to by a single Key.
Parse	The parse operation converts an in-memory raw data set (in CSV format, for example) into a HEX format data set. The parse operation takes a dataset named by a Key as input, and produces a HEX format Key,Value output.
HEX format	The HEX format is an efficient internal representation for data that can be used by H2O algorithms. A data set must be parsed into HEX format before you can operate on it.

System and Environment

Requirements for H2O

H2O node software requirements

- 64-bit Java 1.6 or higher (Java 1.7 is fine, for example)

H2O node hardware requirements

- HDFS disk (for spilling)
- (See resource utilization section below for a discussion of memory requirements)

Supported Hadoop software distributions

- Cloudera CDH3.x (3.5 is tested internally to 0xdata)
- Cloudera CDH4.x (4.3 is tested internally to 0xdata)
    - MapReduce v1 is tested
    - YARN support is in development
- MapR 2.x (2.1.3 is tested internally to 0xdata)

In general, supporting new versions of Hadoop has been straightforward. We have only needed to recompile a small portion of Java code that links with the specific .jar files for the new Hadoop version.

How H2O Nodes are Deployed on Hadoop

H2O nodes run as JVM invocations on Hadoop nodes. (Note that, for performance reasons, 0xdata recommends you avoid running an H2O node on the same hardware as the Hadoop NameNode if it can be avoided.)

For interactive use of H2O, we recommend deploying on a Hadoop cluster dedicated to this purpose. The user creates a long running service within the Hadoop cluster where the H2O cluster stays up for an extended period of time. This shows up in Hadoop Management as a Mapper with H2O_Name.

For batch mode use of H2O, an H2O cluster may be created for the purpose of one computation or related set of computations (run from within a script, for example). The cluster is created, the work is performed, the cluster dissolves, and resources are returned to Hadoop. While the cluster is up, the Hadoop JobTracker can be used to monitor the H2O nodes.

H2O nodes appear as mapper tasks in Hadoop. (Note that even though H2O nodes appear as mapper tasks, H2O nodes and algorithms are performing both map and reduce tasks within the H2O cluster; from a Hadoop standpoint, all of this appears as mapper work inside JobTracker.)

The user can specify how much memory an H2O node has available by specifying the mapper's Java heap size (Xmx). Memory given to H2O will be fully utilized and not be available for other Hadoop jobs.

An H2O cluster with N H2O nodes is created through the following process:

1. Start N mappers through Hadoop (each mapper being an H2O node). All mappers must come up simultaneously for the job to proceed.

2. No work may be sent to the H2O nodes until they find each other and form a cluster. (This means waiting for several seconds during the cluster formation stage.)

3. Send an H2O data operation request to one of the H2O node peers in the H2O cluster. (There is no "master" H2O node.)

0xdata provides an h2odriver jar file that performs steps 1 and 2 for you. (See the "Launch Example" section for details.)

Once the first work item is sent to an H2O cluster, the cluster will consider itself formed and not accept new H2O node members. After the cluster creation phase completes, H2O cluster membership is fixed for the lifetime of the cluster. If an H2O node within a cluster fails, the cluster dissolves and any currently running jobs are abandoned (H2O is an in-memory framework, so if part of an in-memory computation is lost, the entire computation must be abandoned and restarted).

H2O on Hadoop Resource Utilization Overview

Memory	Each H2O node runs as a single Java JVM invocation. The Java heap is specified via Xmx, and the user must plan for this memory to be fully utilized. Memory sizing depends on the data set size. For fastest parse speeds, the total java heap size across the entire H2O cluster should be 4-6x the data set size.
Network I/O	An H2O node does network I/O to read in the initial data set. H2O nodes also communicate (potentially heavily, copying the data again) during the parse step. During an algorithm job, for example GLM running on top of H2O's MapReduce, less data is passed around (merely the intermediate results of reducing); the math algorithms run on local data that lives in memory on the current H2O node.
Disk I/O	Reading in the initial data set requires HDFS accesses, which means that network data requests are going to HDFS data nodes, and the data nodes are reading from disk. An H2O node also uses disk to temporarily spill (otherwise known as swap) data to free up space in the Java heap. For a Hadoop environment, this means spilling to HDFS.
CPU	H2O is math-intensive, and H2O nodes will often max out the CPU available. • For batch invocations of H2O, plan for the allotted CPU to be heavily utilized during the full duration of the job. • For interactive use of H2O, there may be long periods of time when the CPU is not in use (depending on the interactive use pattern). Even though H2O is running as a long-running mapper task, the CPU will only be busy when H2O-level jobs are running in the H2O cluster.

How the User Interacts with H2O

The user has several options for interacting with H2O.

One way is to use a web browser and communicate directly with the embedded web server inside any of the H2O nodes. All H2O nodes contain an embedded web server, and they are all equivalent peers.

A second way is to interface with the H2O embedded web server via the REST API. The REST API accepts HTTP requests and returns JSON-formatted responses.

A third way is for the user to use the H2O.R package from 0xdata, which provides an R-language package for users who wish to use R. (This package uses H2O's REST API under the hood.)

Data sets are not transmitted directly through the REST API. Instead, the user sends a command (containing an HDFS path to the data set, for example) either through the browser or via the REST API to ingest data from disk.

The data set is then assigned a Key in H2O that the user may refer to in future commands to the web server.

How Data is Ingested into H2O

Data is pulled in to an H2O cluster from an HDFS file. The user specifies the HDFS file to H2O using the embedded web server (or programmatically using the REST API).

Supported input data file formats include CSV, Gzip-compressed CSV, MS Excel (XLS), ARRF, HIVE file format, and others. A typical Hadoop user can run a HIVE query, producing a folder containing many files, each containing a part of the full result. H2O conveniently ingests the HIVE folder as a complete data set into one Key.

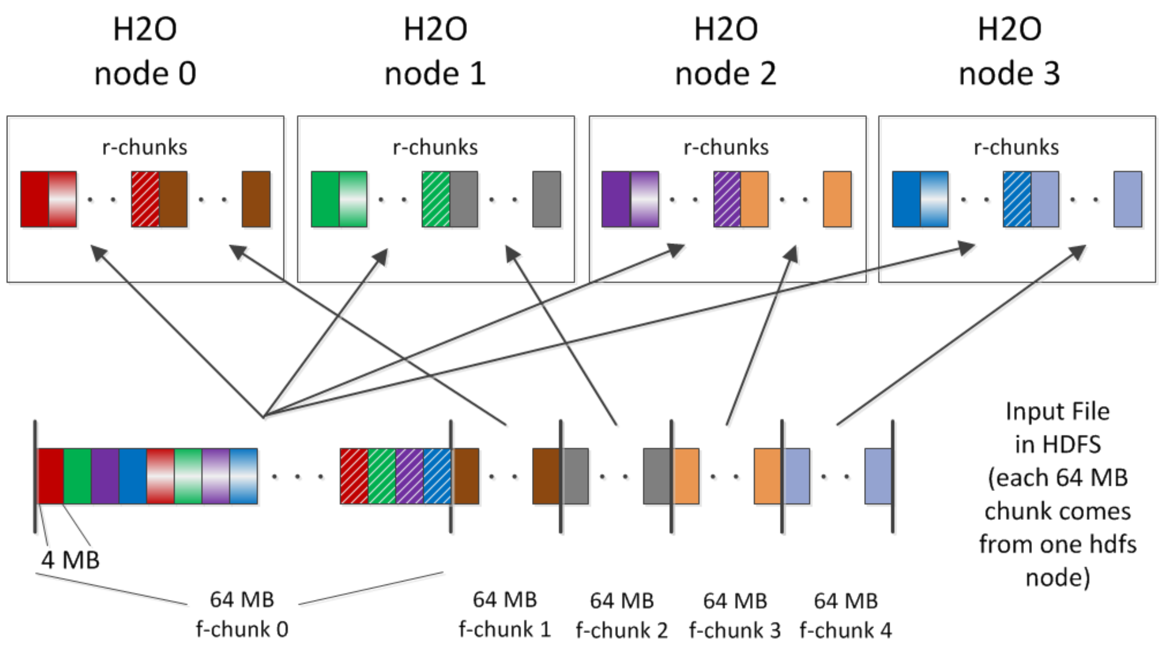
HDFS files are split across HDFS nodes in 64 MB chunks (referred to as file chunks, or f-chunks in the diagram "Raw Data Ingestion Pattern").

When H2O nodes are created, no attempt is made to place them on Hadoop nodes that have pieces of the HDFS input file on their local disk. (Locality optimizations may be added in the future.) Plan for the entire input file to be transferred across the network (albeit in parallel pieces). H2O nodes communicate with each other via both TCP and UDP.

The ingestion process reads f-chunks from the file system and stores the data into r-chunks ("r" in this context stands for a raw, unparsed data format) in H2O node memory.

The first 64 MB f-chunk is sprayed across the H2O cluster in 4 MB pieces. This ensures the data is spread across the H2O cluster for small data sets and parallelization is possible even for small data sets. Subsequent 64 MB f-chunks are sprayed across the H2O cluster as whole 64 MB pieces.

Raw (Pre-Parse) Data Ingestion Pattern

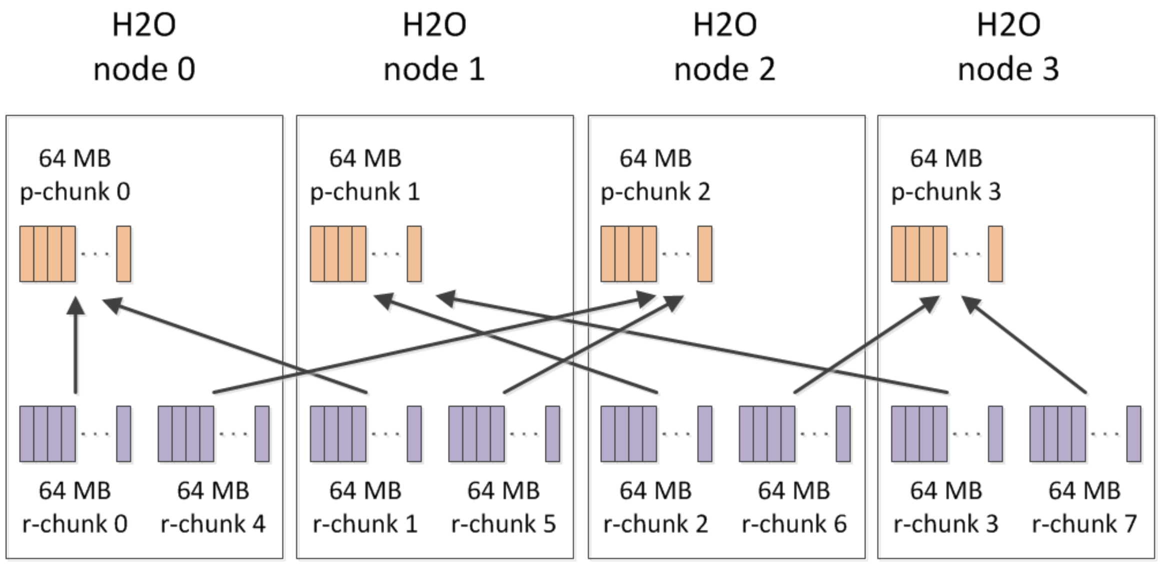


After ingestion, the parse process occurs (see "Parse Data Motion Pattern" diagram). Parsing converts 64 MB in-memory r-chunks (raw unparsed data) into 64 MB in-memory p-chunks (parsed data, which is in the HEX format). Parsing may reduce the overall in-memory data size because the HEX storage format is more efficient than storing uncompressed CSV text input data. (If the input data was compressed CSV to begin with, the size of the parsed HEX data is roughly the same.) Note that (as shown in the diagram) the parse process involves moving the data from one H2O node (where the r-chunk lives) to a different H2O node (where the corresponding p-chunk lives).

After the parse is complete, the parsed data set is in HEX format, and can be referred to by a Key. At this point, the user can feed the HEX data to an algorithm like GLM.

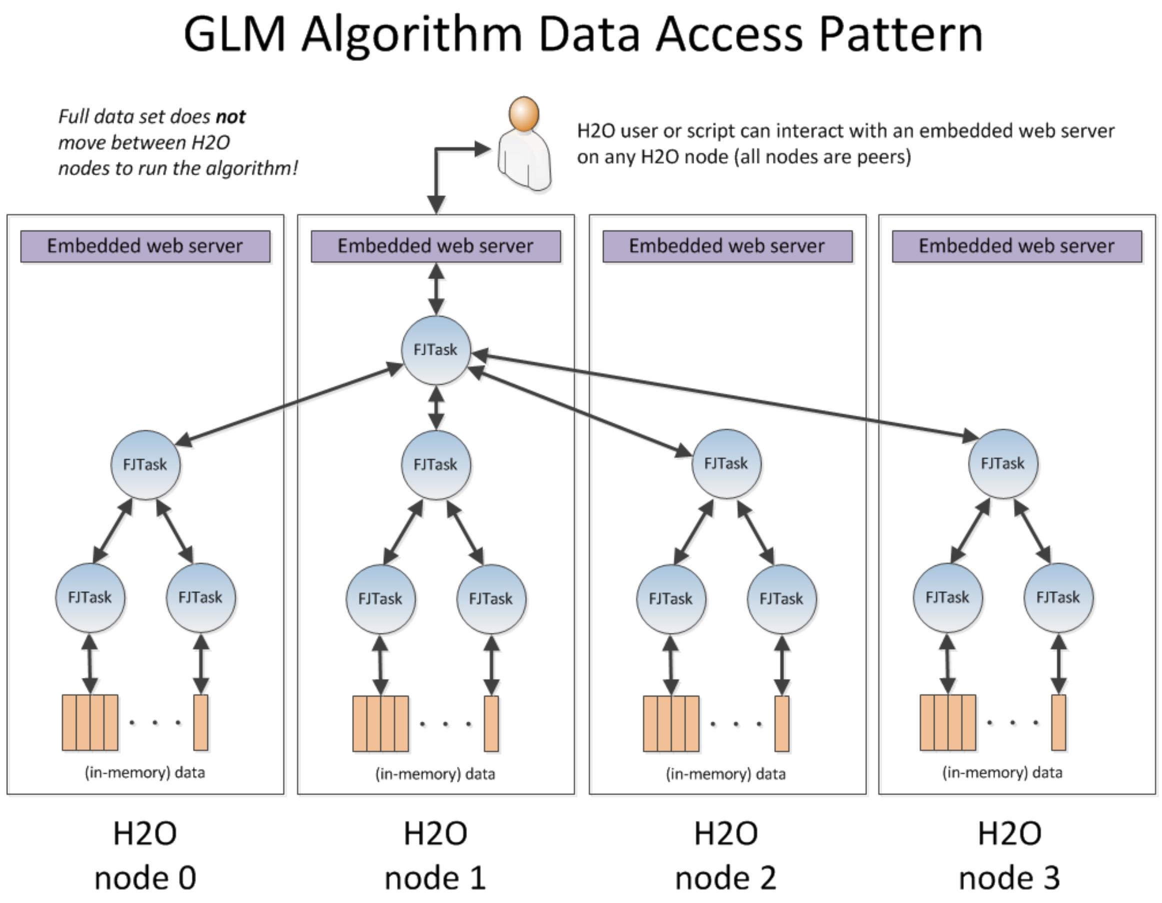
Note that after the data is parsed and residing in memory, it does not need to move again (with GLM, for example), and no additional data I/O is required.

Parse Data Motion Pattern



(Note: all r-chunks and p-chunks live in Java heap memory)

The GLM algorithm's data access pattern is shown in the diagram below.


GLM Algorithm Data Access Pattern
Full data set does **not** move between H2O nodes to run the algorithm!
H2O user or script can interact with an embedded web server on any H2O node (all nodes are peers)
Embedded web server
FJTask
(in-memory) data
H2O node 0
H2O node 1
H2O node 2
H2O node 3

Output from H2O

Output from H2O jobs can be written to HDFS, or be programmatically downloaded using the REST API.

How Algorithms Run on H2O

The H2O math algorithms (e.g. GLM) run on top of H2O's own highly optimized MapReduce implementation inside H2O nodes. H2O nodes within a cluster communicate with each other to distribute the work.

How H2O Interacts with Built-in Hadoop Monitoring

Since H2O nodes run as mapper tasks in Hadoop, administrators can see them in the normal JobTracker and TaskTracker frameworks. This provides process-level (i.e. JVM instance-level) visibility. (Recall, each H2O node is one Java JVM instance.)

For H2O users and job submitters, finer-grain information is available from the embedded web server from within each H2O node. This is accessible using a web browser or through the REST API.

Launch Example

0xdata provides h2odriver jar files for different flavors of Hadoop. Use the appropriate driver jar to start your H2O cluster with a ‘hadoop jar’ command line invocation.

In this example, we start a 4-node H2O cloud on a MapR cluster.

```
$ hadoop jar h2odriver_mapr2.1.3.jar water.hadoop.h2odriver -files
flatfile.txt -libjars h2o.jar -mapperXmx 10g -nodes 4 -output
output100
Determining driver host interface for mapper->driver callback...
    [Possible callback IP address: 192.168.1.171]
    [Possible callback IP address: 127.0.0.1]
Using mapper->driver callback IP address and port:
192.168.1.171:43034
(You can override these with -driverif and -driverport.)
Job name 'H2O_33004' submitted
JobTracker job ID is 'job_201307191330_0089'
Waiting for H2O cluster to come up...
H2O node 192.168.1.172:54321 reports H2O cluster size 1
H2O node 192.168.1.175:54321 reports H2O cluster size 1
H2O node 192.168.1.171:54321 reports H2O cluster size 1
H2O node 192.168.1.174:54321 reports H2O cluster size 1
H2O node 192.168.1.172:54321 reports H2O cluster size 2
H2O node 192.168.1.175:54321 reports H2O cluster size 2
H2O node 192.168.1.172:54321 reports H2O cluster size 3
H2O node 192.168.1.175:54321 reports H2O cluster size 4
H2O node 192.168.1.174:54321 reports H2O cluster size 4
H2O node 192.168.1.172:54321 reports H2O cluster size 4
H2O node 192.168.1.171:54321 reports H2O cluster size 4
H2O cluster (4 nodes) is up
(Press Ctrl-C to kill the cluster)
Blocking until the H2O cluster shuts down...
```

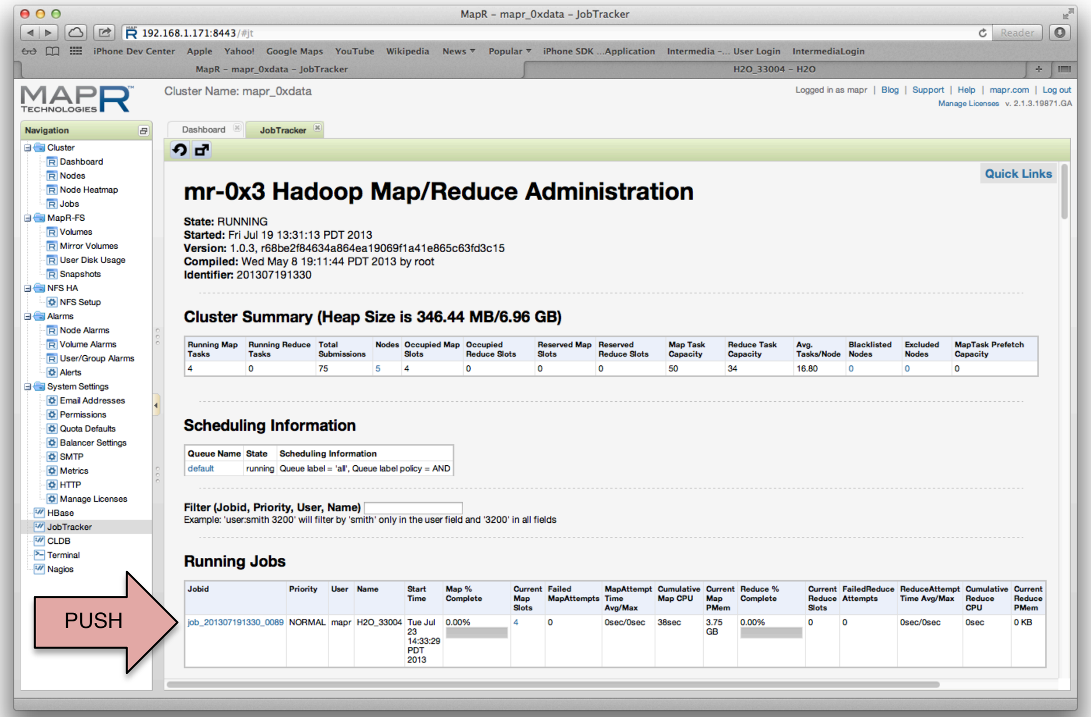
At this point, the H2O cluster is up, and you can interact with it using one of the nodes printed to stdout (e.g. http://192.168.1.175:54321).

For the most up-to-date additional deployment options, consult the driver help, as shown below:

```
$ hadoop jar h2odriver_mapr2.1.3.jar water.hadoop.h2odriver -help
```

Monitoring Example (MapR)

Top JobTracker View



mr-0x3 Hadoop Map/Reduce Administration

State: RUNNING
Started: Fri Jul 19 13:31:13 PDT 2013
Version: 1.0.3, r68be2f84634a864ea19069f1a41e865c63fd3c15
Compiled: Wed May 8 19:11:44 PDT 2013 by root
Identifier: 201307191330

Cluster Summary (Heap Size is 346.44 MB/6.96 GB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Excluded Nodes	MapTask Prefetch Capacity
4	0	75	5	4	0	0	0	50	34	16.80	0	0	0

Scheduling Information

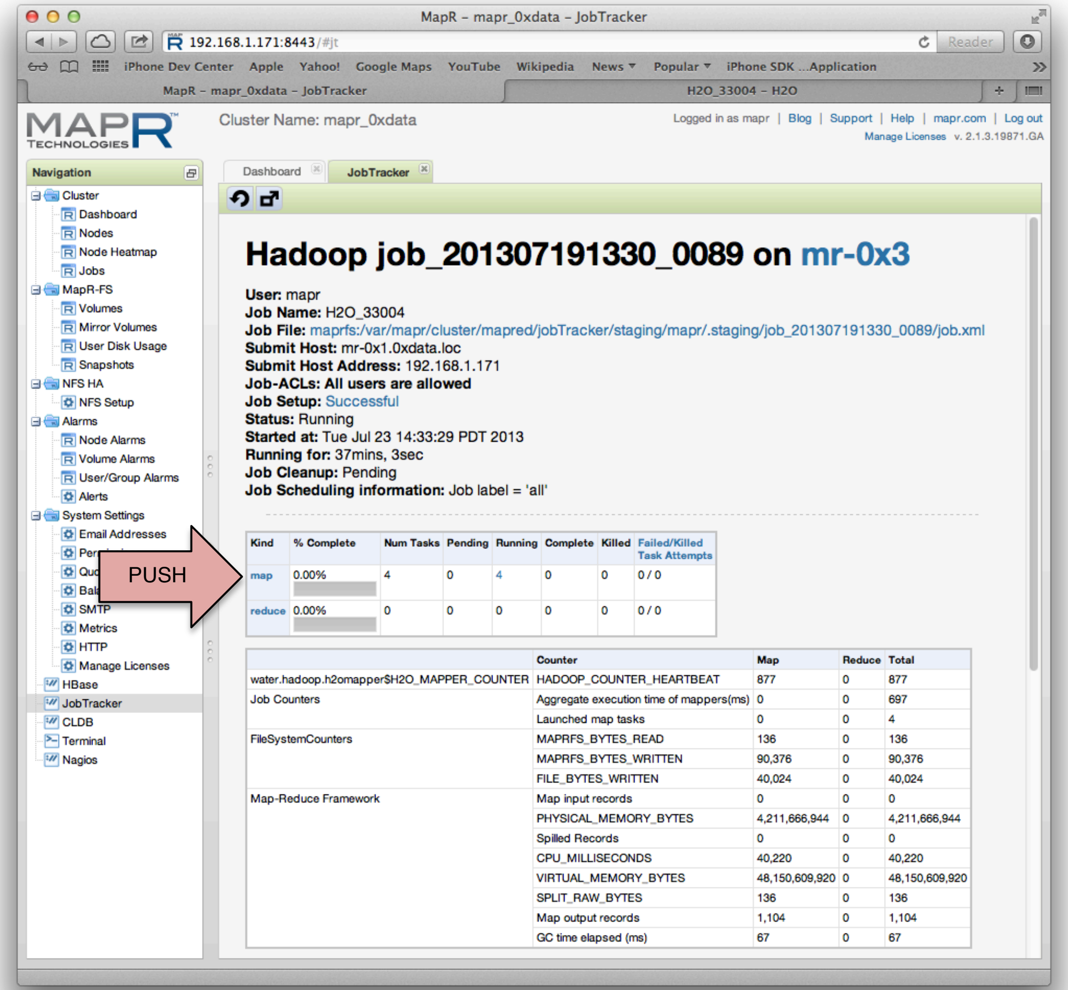
Queue Name	State	Scheduling Information
default	running	Queue label = 'all', Queue label policy = AND

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

Jobid	Priority	User	Name	Start Time	Map % Complete	Current Map Slots	Failed MapAttempts	MapAttempt Time Avg/Max	Cumulative Map CPU	Current Map PMem	Reduce % Complete	Current Reduce Slots	FailedReduce Attempts	ReduceAttempt Time Avg/Max	Cumulative Reduce CPU	Current Reduce PMem
job_201307191330_0089	NORMAL	mapr	H2O_33004	Tue Jul 23 14:33:29 PDT 2013	0.00%	4	0	0sec/0sec	38sec	3.75 GB	0.00%	0	0	0sec/0sec	0sec	0 KB

Running Job View



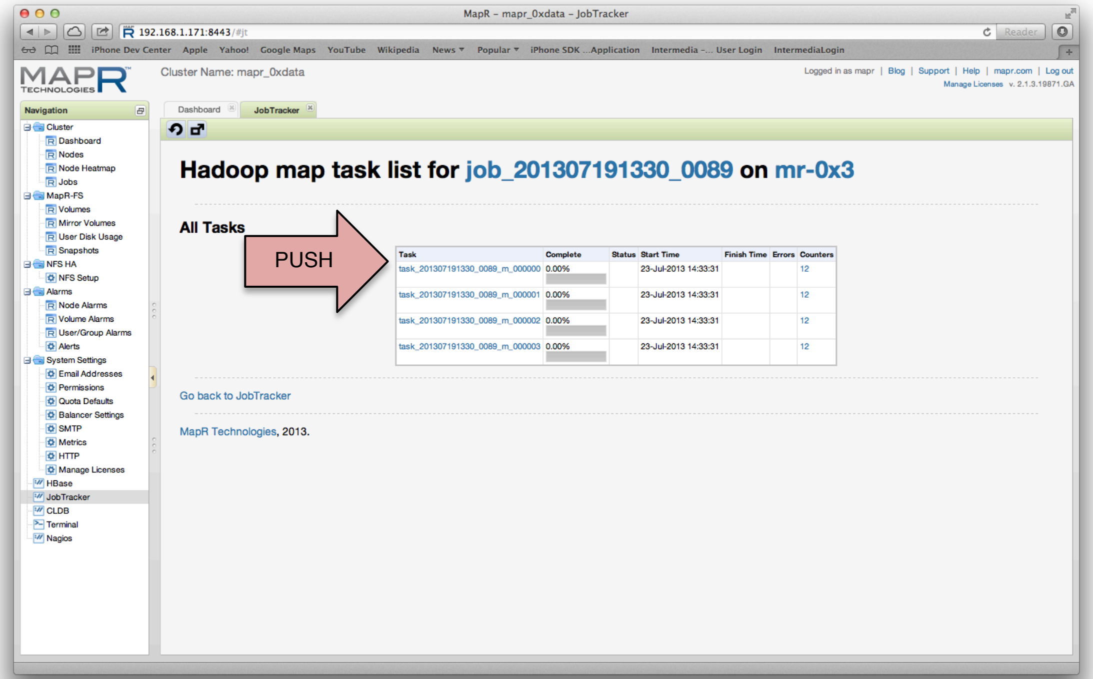
Hadoop job_201307191330_0089 on mr-0x3

User: mapr
Job Name: H2O_33004
Job File: maprfs:/var/mapr/cluster/mapred/jobTracker/staging/mapr/.staging/job_201307191330_0089/job.xml
Submit Host: mr-0x1.0xdata.loc
Submit Host Address: 192.168.1.171
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Running
Started at: Tue Jul 23 14:33:29 PDT 2013
Running for: 37mins, 3sec
Job Cleanup: Pending
Job Scheduling information: Job label = 'all'

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	0.00%	4	0	4	0	0	0 / 0
reduce	0.00%	0	0	0	0	0	0 / 0

	Counter	Map	Reduce	Total
water.hadoop.h2omapper$H2O_MAPPER_COUNTER	HADOOP_COUNTER_HEARTBEAT	877	0	877
Job Counters	Aggregate execution time of mappers(ms)	0	0	697
	Launched map tasks	0	0	4
FileSystemCounters	MAPRFS_BYTES_READ	136	0	136
	MAPRFS_BYTES_WRITTEN	90,376	0	90,376
	FILE_BYTES_WRITTEN	40,024	0	40,024
Map-Reduce Framework	Map input records	0	0	0
	PHYSICAL_MEMORY_BYTES	4,211,666,944	0	4,211,666,944
	Spilled Records	0	0	0
	CPU_MILLISECONDS	40,220	0	40,220
	VIRTUAL_MEMORY_BYTES	48,150,609,920	0	48,150,609,920
	SPLIT_RAW_BYTES	136	0	136
	Map output records	1,104	0	1,104
	GC time elapsed (ms)	67	0	67

List of Mappers View



Hadoop map task list for job_201307191330_0089 on mr-0x3

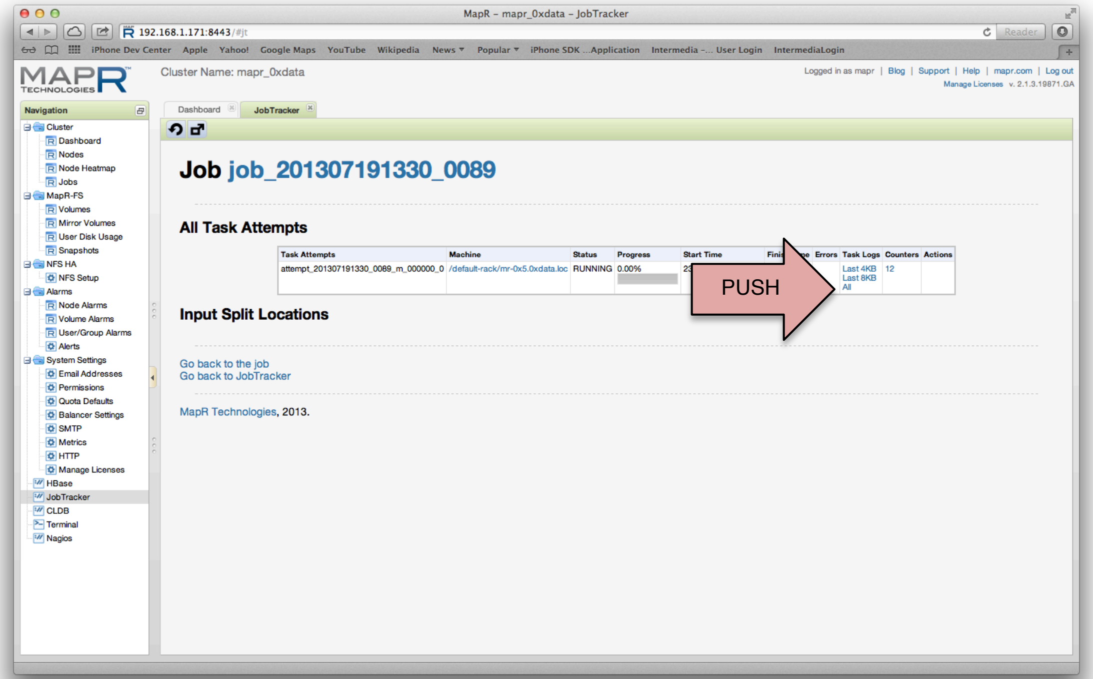
All Tasks

Task	Complete	Status	Start Time	Finish Time	Errors	Counters
task_201307191330_0089_m_000000	0.00%		23-Jul-2013 14:33:31			12
task_201307191330_0089_m_000001	0.00%		23-Jul-2013 14:33:31			12
task_201307191330_0089_m_000002	0.00%		23-Jul-2013 14:33:31			12
task_201307191330_0089_m_000003	0.00%		23-Jul-2013 14:33:31			12

Go back to JobTracker

MapR Technologies, 2013.

Mapper Task View



Job job_201307191330_0089

All Task Attempts

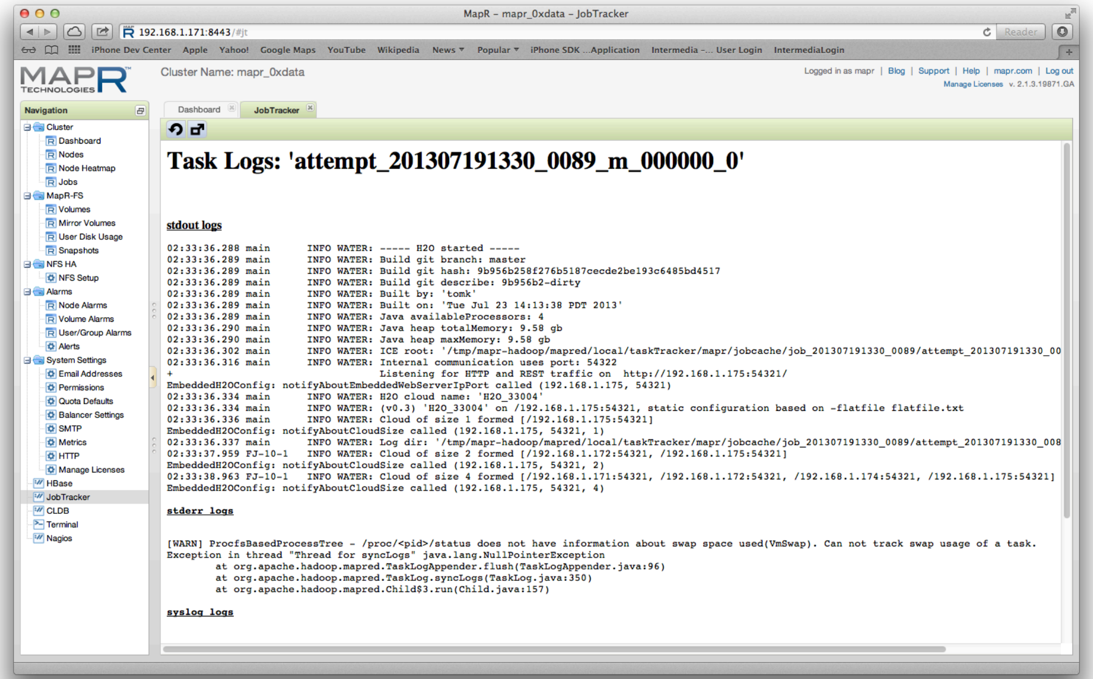
Task Attempts	Machine	Status	Progress	Start Time	Finis…me	Errors	Task Logs	Counters	Actions
attempt_201307191330_0089_m_000000_0	/default-rack/mr-0x5.0xdata.loc	RUNNING	0.00%	23			Last 4KB Last 8KB All	12	

Input Split Locations

Go back to the job
Go back to JobTracker

MapR Technologies, 2013.

Mapper Log Output



Task Logs: 'attempt_201307191330_0089_m_000000_0'

stdout logs

```
02:33:36.288 main      INFO WATER: ----- H2O started -----
02:33:36.289 main      INFO WATER: Build git branch: master
02:33:36.289 main      INFO WATER: Build git hash: 9b956b258f276b5187cecde2be193c6485bd4517
02:33:36.289 main      INFO WATER: Build git describe: 9b956b2-dirty
02:33:36.289 main      INFO WATER: Built by: 'tomk'
02:33:36.289 main      INFO WATER: Built on: 'Tue Jul 23 14:13:38 PDT 2013'
02:33:36.289 main      INFO WATER: Java availableProcessors: 4
02:33:36.290 main      INFO WATER: Java heap totalMemory: 9.58 gb
02:33:36.290 main      INFO WATER: Java heap maxMemory: 9.58 gb
02:33:36.302 main      INFO WATER: ICE root: '/tmp/mapr-hadoop/mapred/local/taskTracker/mapr/jobcache/job_201307191330_0089/attempt_201307191330_00
02:33:36.316 main      INFO WATER: Internal communication uses port: 54322
+                                  Listening for HTTP and REST traffic on  http://192.168.1.175:54321/
EmbeddedH2OConfig: notifyAboutEmbeddedWebServerIpPort called (192.168.1.175, 54321)
02:33:36.334 main      INFO WATER: H2O cloud name: 'H2O_33004'
02:33:36.334 main      INFO WATER: (v0.3) 'H2O_33004' on /192.168.1.175:54321, static configuration based on -flatfile flatfile.txt
02:33:36.336 main      INFO WATER: Cloud of size 1 formed [/192.168.1.175:54321]
EmbeddedH2OConfig: notifyAboutCloudSize called (192.168.1.175, 54321, 1)
02:33:36.337 main      INFO WATER: Log dir: '/tmp/mapr-hadoop/mapred/local/taskTracker/mapr/jobcache/job_201307191330_0089/attempt_201307191330_008
02:33:37.959 FJ-10-1   INFO WATER: Cloud of size 2 formed [/192.168.1.172:54321, /192.168.1.175:54321]
EmbeddedH2OConfig: notifyAboutCloudSize called (192.168.1.175, 54321, 2)
02:33:38.963 FJ-10-1   INFO WATER: Cloud of size 4 formed [/192.168.1.171:54321, /192.168.1.172:54321, /192.168.1.174:54321, /192.168.1.175:54321]
EmbeddedH2OConfig: notifyAboutCloudSize called (192.168.1.175, 54321, 4)
```

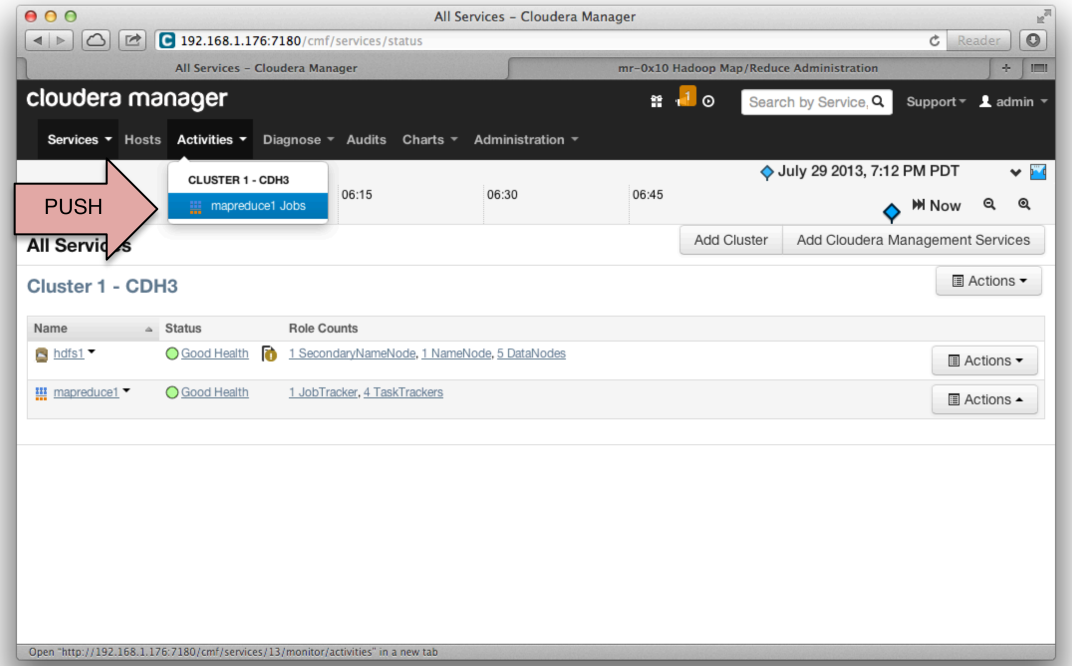
stderr logs

```
[WARN] ProcfsBasedProcessTree - /proc/<pid>/status does not have information about swap space used(VmSwap). Can not track swap usage of a task.
Exception in thread "Thread for syncLogs" java.lang.NullPointerException
        at org.apache.hadoop.mapred.TaskLogAppender.flush(TaskLogAppender.java:96)
        at org.apache.hadoop.mapred.TaskLog.syncLogs(TaskLog.java:350)
        at org.apache.hadoop.mapred.Child$3.run(Child.java:157)
```

syslog logs

Monitoring Example (Cloudera Manager)

All Services View


All Services – Cloudera Manager
192.168.1.176:7180/cmf/services/status
All Services – Cloudera Manager
mr-0x10 Hadoop Map/Reduce Administration
cloudera manager
Search by Service,
Support
admin
Services
Hosts
Activities
Diagnose
Audits
Charts
Administration
CLUSTER 1 - CDH3
mapreduce1 Jobs
PUSH
July 29 2013, 7:12 PM PDT
06:15
06:30
06:45
Now
Add Cluster
Add Cloudera Management Services
Actions

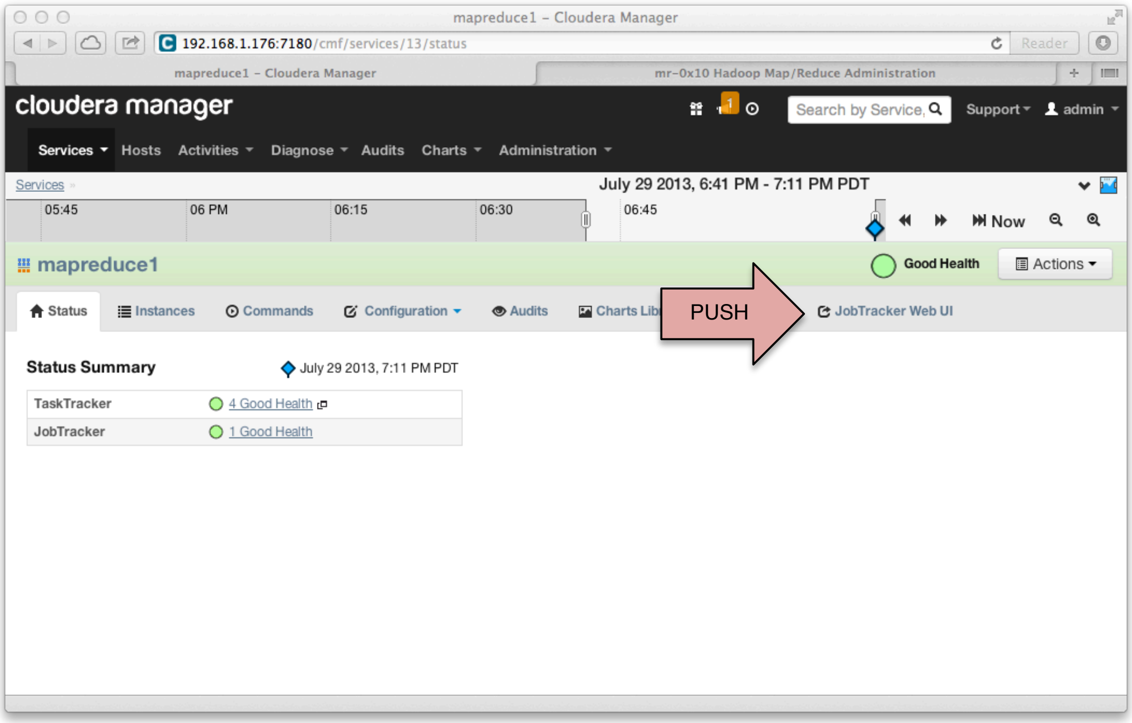
All Services

Cluster 1 - CDH3

Name	Status	Role Counts	
hdfs1	Good Health	1 SecondaryNameNode, 1 NameNode, 5 DataNodes	Actions
mapreduce1	Good Health	1 JobTracker, 4 TaskTrackers	Actions

Open "http://192.168.1.176:7180/cmf/services/13/monitor/activities" in a new tab

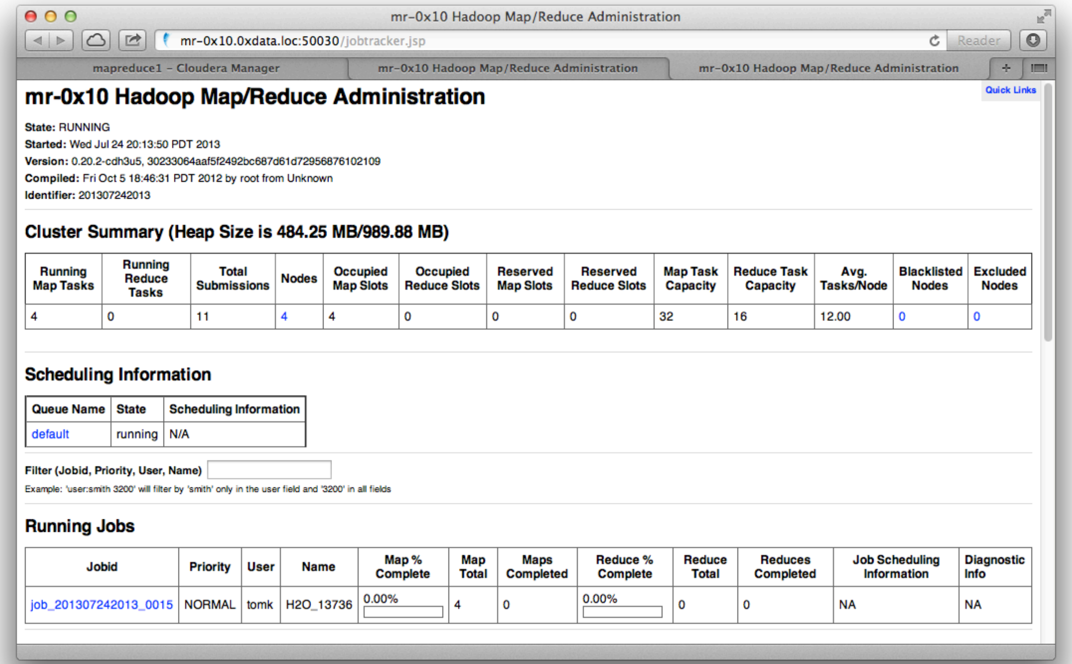
MapReduce Service View



Status Summary July 29 2013, 7:11 PM PDT

TaskTracker	4 Good Health
JobTracker	1 Good Health

Top JobTracker View


mr-0x10 Hadoop Map/Reduce Administration
mr-0x10.0xdata.loc:50030/jobtracker.jsp
Quick Links

mr-0x10 Hadoop Map/Reduce Administration

State: RUNNING
Started: Wed Jul 24 20:13:50 PDT 2013
Version: 0.20.2-cdh3u5, 30233064aaf5f2492bc687d61d72956876102109
Compiled: Fri Oct 5 18:46:31 PDT 2012 by root from Unknown
Identifier: 201307242013

Cluster Summary (Heap Size is 484.25 MB/989.88 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Excluded Nodes
4	0	11	4	4	0	0	0	32	16	12.00	0	0

Scheduling Information

Queue Name	State	Scheduling Information
default	running	N/A

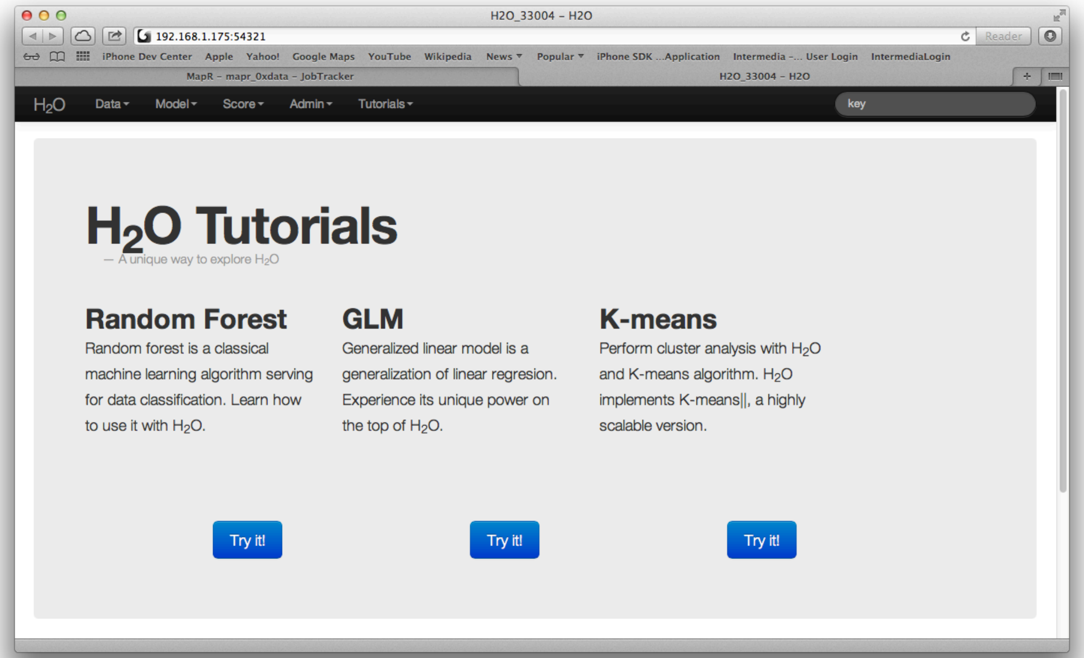
Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information	Diagnostic Info
job_201307242013_0015	NORMAL	tomk	H2O_13736	0.00%	4	0	0.00%	0	0	NA	NA

Monitoring Example (H2O Browser UI)

H2O Main View


H2O_33004 – H2O
192.168.1.175:54321
iPhone Dev Center Apple Yahoo! Google Maps YouTube Wikipedia News Popular iPhone SDK ...Application Intermedia -... User Login IntermediaLogin
MapR – mapr_0xdata – JobTracker
H2O_33004 – H2O
H2O Data Model Score Admin Tutorials
key

H_2O Tutorials

— A unique way to explore H_2O

Random Forest

Random forest is a classical machine learning algorithm serving for data classification. Learn how to use it with H_2O.

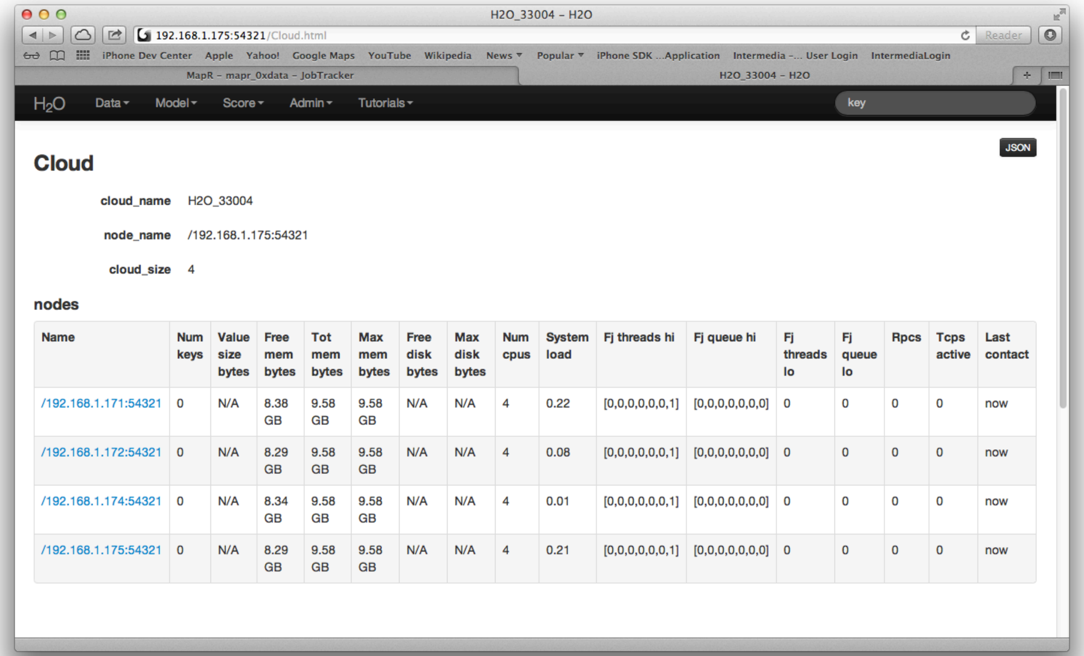
GLM

Generalized linear model is a generalization of linear regresion. Experience its unique power on the top of H_2O.

K-means

Perform cluster analysis with H_2O and K-means algorithm. H_2O implements K-means||, a highly scalable version.

H2O Cluster Status View



Cloud

cloud_name	H2O_33004
node_name	/192.168.1.175:54321
cloud_size	4

nodes

Name	Num keys	Value size bytes	Free mem bytes	Tot mem bytes	Max mem bytes	Free disk bytes	Max disk bytes	Num cpus	System load	Fj threads hi	Fj queue hi	Fj threads lo	Fj queue lo	Rpcs	Tcps active	Last contact
/192.168.1.171:54321	0	N/A	8.38 GB	9.58 GB	9.58 GB	N/A	N/A	4	0.22	[0,0,0,0,0,0,1]	[0,0,0,0,0,0,0]	0	0	0	0	now
/192.168.1.172:54321	0	N/A	8.29 GB	9.58 GB	9.58 GB	N/A	N/A	4	0.08	[0,0,0,0,0,0,1]	[0,0,0,0,0,0,0]	0	0	0	0	now
/192.168.1.174:54321	0	N/A	8.34 GB	9.58 GB	9.58 GB	N/A	N/A	4	0.01	[0,0,0,0,0,0,1]	[0,0,0,0,0,0,0]	0	0	0	0	now
/192.168.1.175:54321	0	N/A	8.29 GB	9.58 GB	9.58 GB	N/A	N/A	4	0.21	[0,0,0,0,0,0,1]	[0,0,0,0,0,0,0]	0	0	0	0	now

Appendix A:

Latest version of the appendix is here:

https://github.com/0xdata/h2o/blob/master/hadoop/README.txt

```
RUNNING H2O NODES IN HADOOP
===========================

Note: You may want to do all of this from the machine where you plan
to launch the hadoop jar job from.  Otherwise you will end up having
to copy files around.

(If you grabbed a prebuilt h2o-*.zip file, copy it to a hadoop machine
and skip to the PREPARE section below.)


GET H2O TREE FROM GIT
---------------------

$ git clone https://github.com/0xdata/h2o.git
$ cd h2o


BUILD CODE
----------

$ make


COPY BUILD OUTPUT TO HADOOP NODE
--------------------------------

Copy target/h2o-*.zip <to place where you intend to run hadoop command>


PREPARE JOB INPUT ON HADOOP NODE
--------------------------------

$ unzip h2o-*.zip
$ cd h2o-*
$ cd hadoop

Create flatfile.txt.

(Note: The flat file must contain the list of possible IP addresses an
       H2O node (i.e. mapper) may be scheduled on.  One IP address
       per line.)

Here is an example flatfile.txt:
$ cat flatfile.txt
192.168.1.150
192.168.1.151
192.168.1.152
```

```
192.168.1.153
192.168.1.154
192.168.1.155
```

For your convenience, we have included a tool to help you genearate
a flatfile. This is only meant to assist you, and may encounter
Java exceptions if DNS and DHCP are not fully configured.
This generator tool is still experimental, please double check the
output yourself before relying on it.

```
$ hadoop jar h2odriver_cdh4.jar water.hadoop.gen_flatfile -jt
<jobtracker:port> > flatfile.txt
```

```
(Note: Make sure to use the h2odriver flavor for the correct version
       of hadoop!  We recommend running the hadoop command from a
       machine in the hadoop cluster.)

(Note: Port 8021 is the default jobtracker port for Cloudera.
       Port 9001 is the default jobtracker port for MapR.)
```

RUN JOB

```
$ hadoop jar h2odriver_cdh4.jar water.hadoop.h2odriver [-jt
<jobtracker:port>] -files flatfile.txt -libjars ../h2o.jar -mapperXmx
1g -nodes 1 -output hdfsOutputDirName
```

```
(Note: -nodes refers to H2O nodes.  This may be less than or equal to
       the number of hadoop machines running TaskTrackers where hadoop
       mapreduce Tasks may land.)

(Note: Make sure to use the h2odriver flavor for the correct version
       of hadoop!  We recommend running the hadoop command from a
       machine in the hadoop cluster.)

(Note: Port 8021 is the default jobtracker port for Cloudera.
       Port 9001 is the default jobtracker port for MapR.)
```

MONITOR JOB

Use standard job tracker web UI. (http://<jobtrackerip>:50030)
Different distros sometimes have different job tracker Web UI ports.
The cloudera default is 50030.

SHUT DOWN THE CLUSTER

Bring up H2O web UI: http://<h2onode>:54321
Choose Admin->Shutdown

(Note: Alternately use the "hadoop job -kill" command.)

```
FOR MORE INFORMATION
--------------------

$ hadoop jar hadoop/h2odriver_cdh4.jar water.hadoop.h2odriver -help
```

Document history

Date	Author	Description
2013-May-22	TMK	Initial version.
2013-June-22	TMK	Updated algorithm picture.
2013-July-23	TMK	Added examples.
2013-July-29	SA, TMK	Changed document template. Added Appendix A. Added more examples.