H_2O on Hadoop

[H_2O – The Open Source In-Memory Prediction Engine for Big Data]

Workshop Café, San Francisco, Dec. 12, 2013

Tom Kraljevic

0xdata

H_2O as a Standalone HDFS client

H_2O on Hadoop

Configuration settings

Stuff we learned

Questions

Why

I want to do run a Generalized Linear Model, and my data lives in HDFS.

What

Use H_2O standalone.

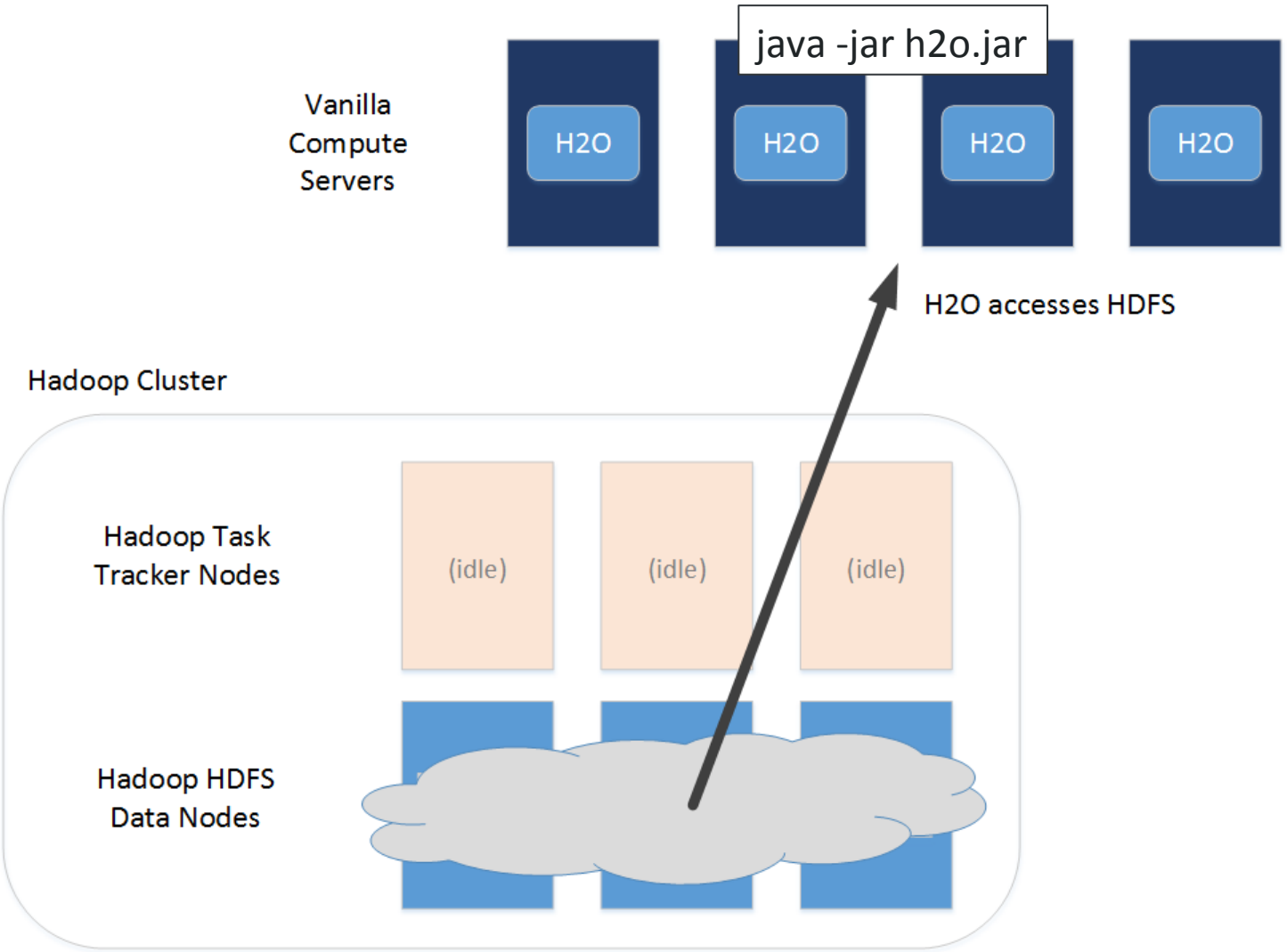
Why

Let's also use the CPUs and Memory of my Hadoop cluster!

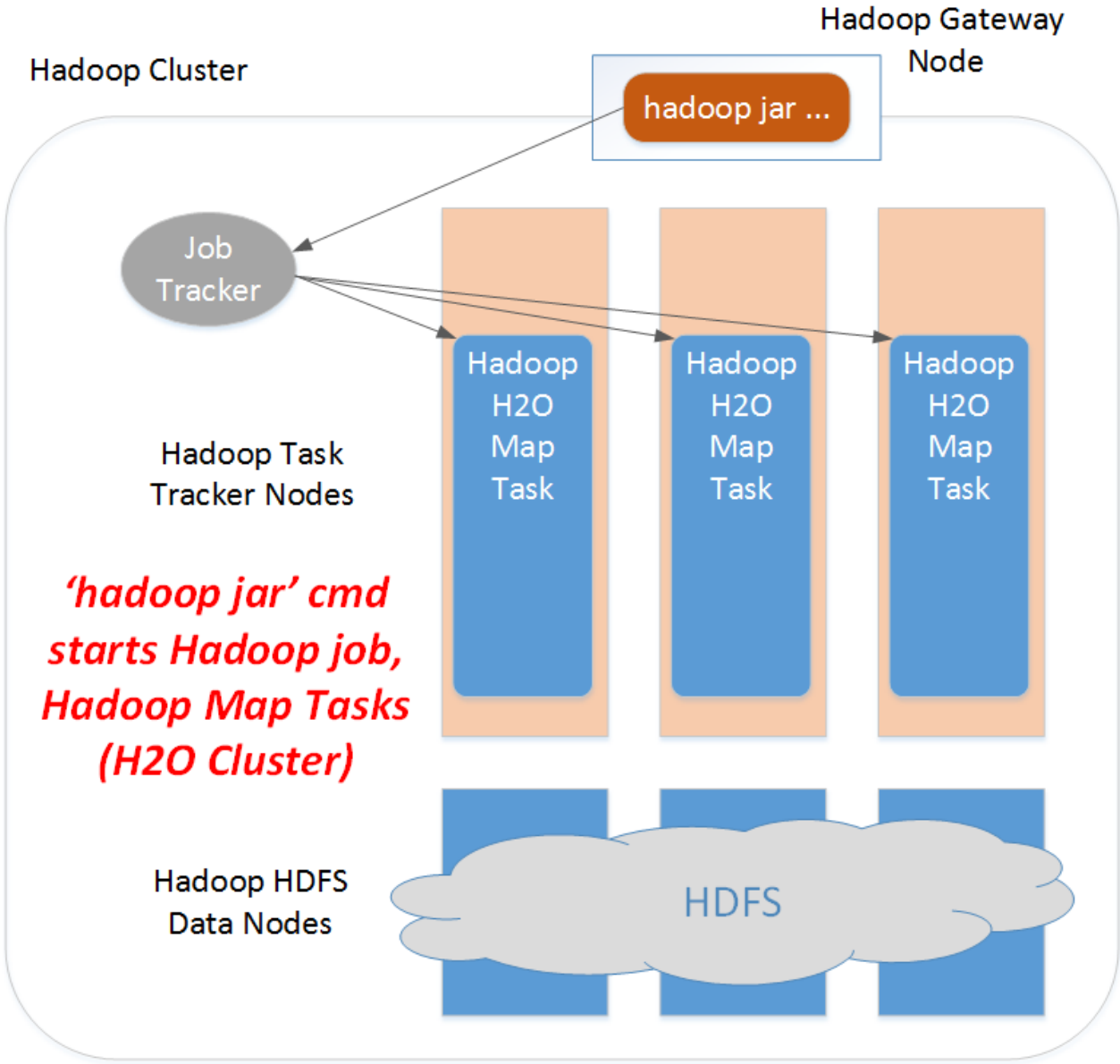
What

Run H_2O on Hadoop.

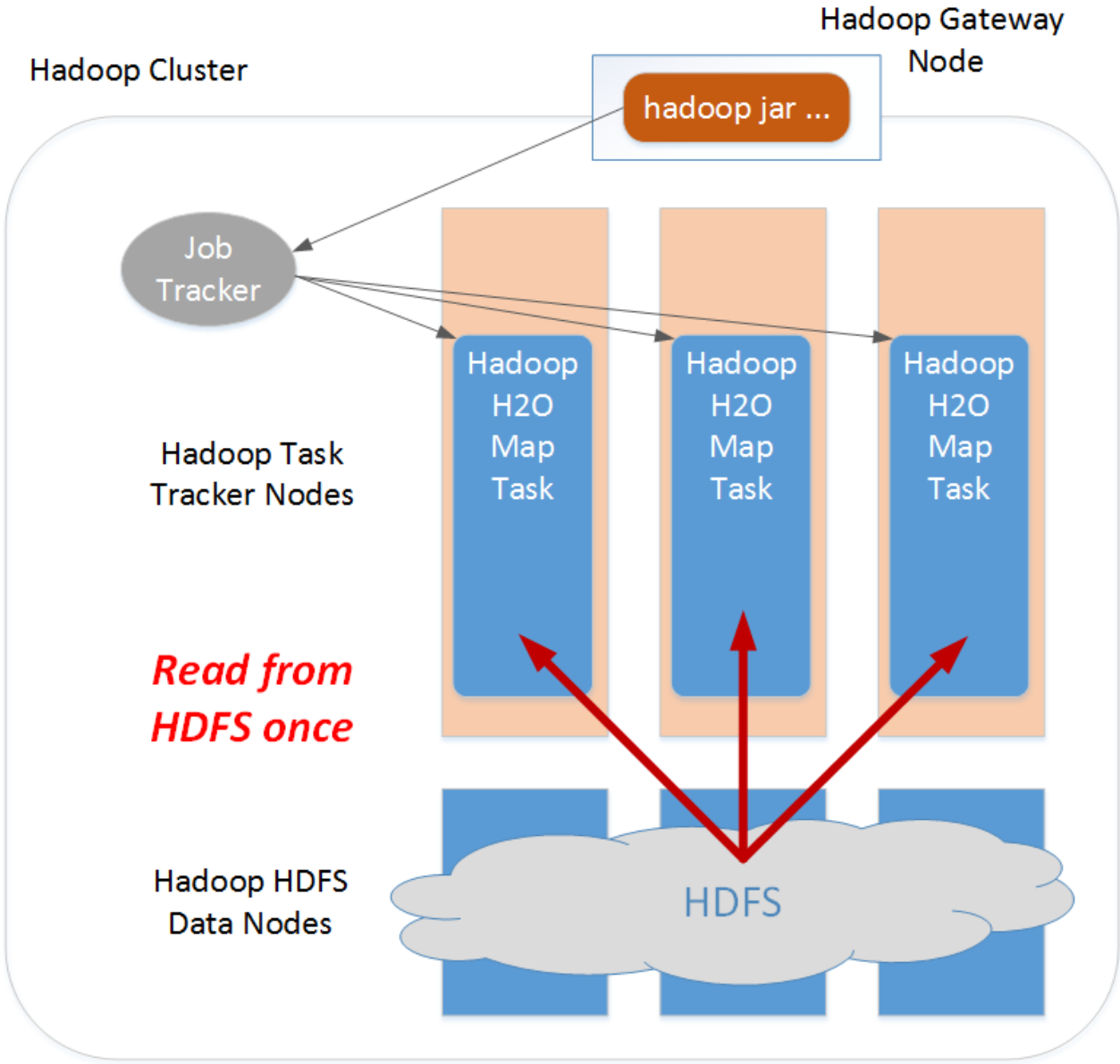
H_2O Standalone Deployment Using HDFS



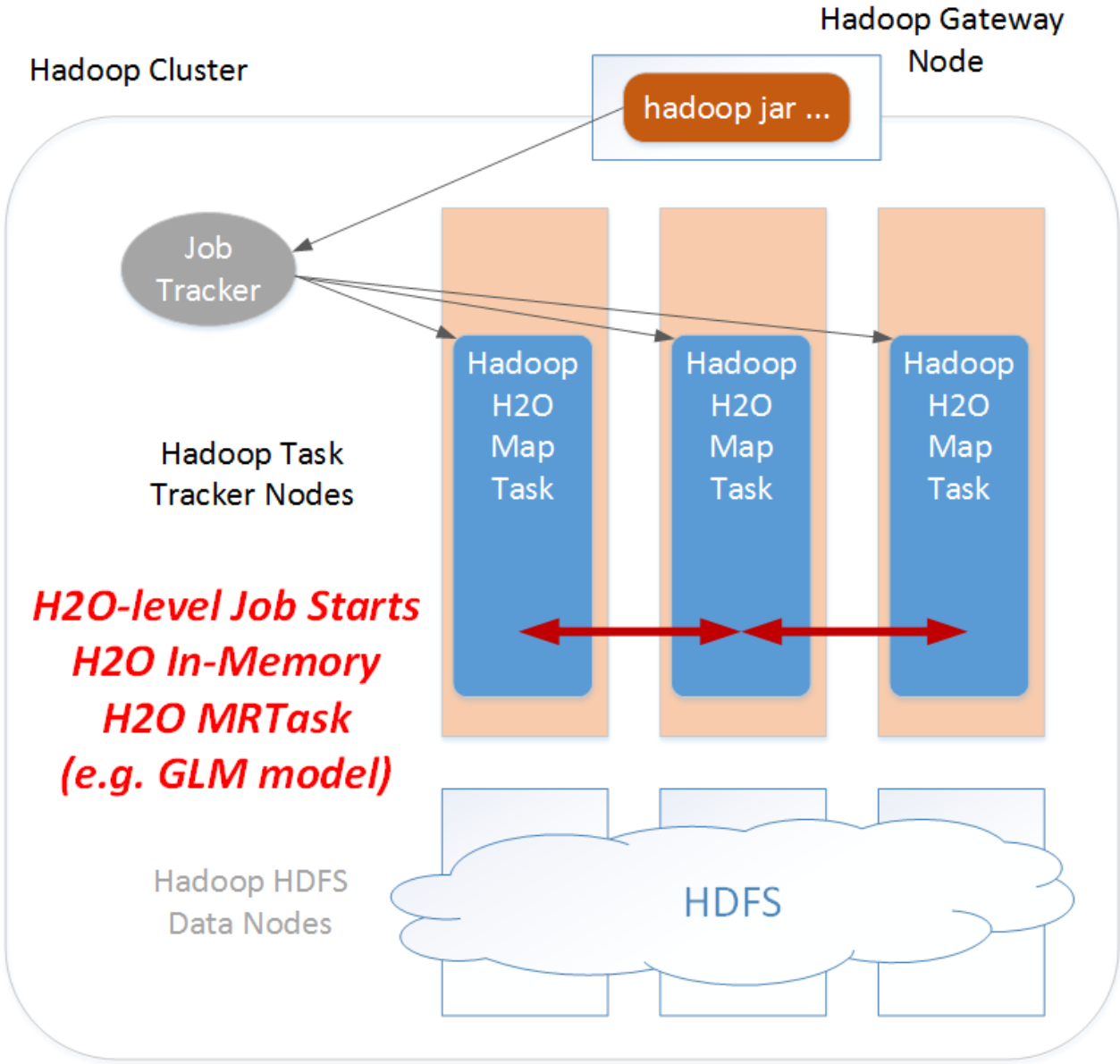
H_2O on Hadoop Deployment



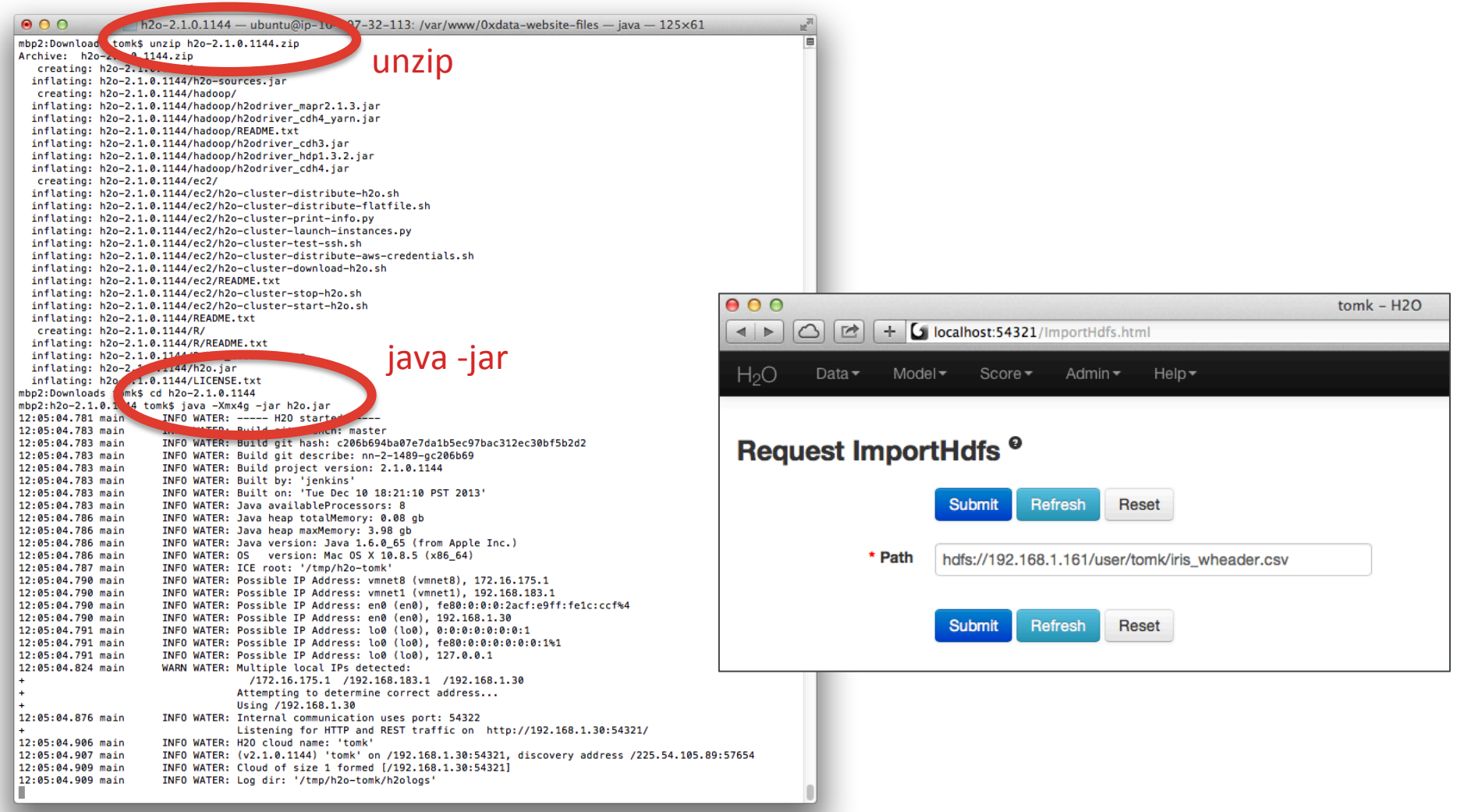
H_2O on Hadoop Deployment



H_2O on Hadoop Deployment



Standalone Mode Command-Line Invocation




```
mbp2:Downloads tomk$ unzip h2o-2.1.0.1144.zip
Archive:  h2o-2.1.0.1144.zip
   creating: h2o-2.1.0.1144/
  inflating: h2o-2.1.0.1144/h2o-sources.jar
   creating: h2o-2.1.0.1144/hadoop/
  inflating: h2o-2.1.0.1144/hadoop/h2odriver_mapr2.1.3.jar
  inflating: h2o-2.1.0.1144/hadoop/h2odriver_cdh4_yarn.jar
  inflating: h2o-2.1.0.1144/hadoop/README.txt
  inflating: h2o-2.1.0.1144/hadoop/h2odriver_cdh3.jar
  inflating: h2o-2.1.0.1144/hadoop/h2odriver_hdp1.3.2.jar
  inflating: h2o-2.1.0.1144/hadoop/h2odriver_cdh4.jar
   creating: h2o-2.1.0.1144/ec2/
  inflating: h2o-2.1.0.1144/ec2/h2o-cluster-distribute-h2o.sh
  inflating: h2o-2.1.0.1144/ec2/h2o-cluster-distribute-flatfile.sh
  inflating: h2o-2.1.0.1144/ec2/h2o-cluster-print-info.py
  inflating: h2o-2.1.0.1144/ec2/h2o-cluster-launch-instances.py
  inflating: h2o-2.1.0.1144/ec2/h2o-cluster-test-ssh.sh
  inflating: h2o-2.1.0.1144/ec2/h2o-cluster-distribute-aws-credentials.sh
  inflating: h2o-2.1.0.1144/ec2/h2o-cluster-download-h2o.sh
  inflating: h2o-2.1.0.1144/ec2/README.txt
  inflating: h2o-2.1.0.1144/ec2/h2o-cluster-stop-h2o.sh
  inflating: h2o-2.1.0.1144/ec2/h2o-cluster-start-h2o.sh
  inflating: h2o-2.1.0.1144/README.txt
   creating: h2o-2.1.0.1144/R/
  inflating: h2o-2.1.0.1144/R/README.txt
  inflating: h2o-2.1.0.1144/
  inflating: h2o-2.1.0.1144/h2o.jar
  inflating: h2o-2.1.0.1144/LICENSE.txt
mbp2:Downloads tomk$ cd h2o-2.1.0.1144
mbp2:h2o-2.1.0.1144 tomk$ java -Xmx4g -jar h2o.jar
12:05:04.781 main      INFO WATER: ----- H2O started  -----
12:05:04.783 main      INFO WATER: Build git branch: master
12:05:04.783 main      INFO WATER: Build git hash: c206b694ba07e7da1b5ec97bac312ec30bf5b2d2
12:05:04.783 main      INFO WATER: Build git describe: nn-2-1489-gc206b69
12:05:04.783 main      INFO WATER: Build project version: 2.1.0.1144
12:05:04.783 main      INFO WATER: Built by: 'jenkins'
12:05:04.783 main      INFO WATER: Built on: 'Tue Dec 10 18:21:10 PST 2013'
12:05:04.783 main      INFO WATER: Java availableProcessors: 8
12:05:04.786 main      INFO WATER: Java heap totalMemory: 0.08 gb
12:05:04.786 main      INFO WATER: Java heap maxMemory: 3.98 gb
12:05:04.786 main      INFO WATER: Java version: Java 1.6.0_65 (from Apple Inc.)
12:05:04.786 main      INFO WATER: OS   version: Mac OS X 10.8.5 (x86_64)
12:05:04.787 main      INFO WATER: ICE root: '/tmp/h2o-tomk'
12:05:04.790 main      INFO WATER: Possible IP Address: vmnet8 (vmnet8), 172.16.175.1
12:05:04.790 main      INFO WATER: Possible IP Address: vmnet1 (vmnet1), 192.168.183.1
12:05:04.790 main      INFO WATER: Possible IP Address: en0 (en0), fe80:0:0:0:2acf:e9ff:fe1c:ccf%4
12:05:04.790 main      INFO WATER: Possible IP Address: en0 (en0), 192.168.1.30
12:05:04.791 main      INFO WATER: Possible IP Address: lo0 (lo0), 0:0:0:0:0:0:0:1
12:05:04.791 main      INFO WATER: Possible IP Address: lo0 (lo0), fe80:0:0:0:0:0:0:1%1
12:05:04.791 main      INFO WATER: Possible IP Address: lo0 (lo0), 127.0.0.1
12:05:04.824 main      WARN WATER: Multiple local IPs detected:
+                                  /172.16.175.1  /192.168.183.1  /192.168.1.30
+                                 Attempting to determine correct address...
+                                 Using /192.168.1.30
12:05:04.876 main      INFO WATER: Internal communication uses port: 54322
+                                 Listening for HTTP and REST traffic on  http://192.168.1.30:54321/
12:05:04.906 main      INFO WATER: H2O cloud name: 'tomk'
12:05:04.907 main      INFO WATER: (v2.1.0.1144) 'tomk' on /192.168.1.30:54321, discovery address /225.54.105.89:57654
12:05:04.909 main      INFO WATER: Cloud of size 1 formed [/192.168.1.30:54321]
12:05:04.909 main      INFO WATER: Log dir: '/tmp/h2o-tomk/h2ologs'
```

H2O on Hadoop Client Invocation

```
tomk@mr-0xb1:~/h2o-2.1.0.1144/hadoop$ hadoop jar h2odriver_cdh4.jar water.hadoop.h2odriver
-libjars ../h2o.jar -mapperXmx 30g -nodes 3 -output hdfsOutDir
Determining driver host interface for mapper->driver callback...
    [Possible callback IP address: 192.168.1.161]
    [Possible callback IP address: 127.0.0.1]
Using mapper->driver callback IP address and port: 192.168.1.161:60576
(You can override these with -driverif and -driverport.)
Driver program compiled with MapReduce V1 (Classic)
13/12/12 13:25:29 WARN conf.Configuration: mapred.map.child.java.opts is deprecated. Instead, use
mapreduce.map.java.opts
Memory Settings:
    mapred.child.java.opts:      -Xms30g -Xmx30g
    mapred.map.child.java.opts:  -Xms30g -Xmx30g
    Extra memory percent:        10
    mapreduce.map.memory.mb:     33792
Job name 'H2O_49792' submitted
JobTracker job ID is 'job_1386713862878_0002'
Waiting for H2O cluster to come up...
H2O node 192.168.1.163:54321 requested flatfile
H2O node 192.168.1.161:54323 requested flatfile
H2O node 192.168.1.162:54339 requested flatfile
Sending flatfiles to nodes...
    [Sending flatfile to node 192.168.1.163:54321]
    [Sending flatfile to node 192.168.1.161:54323]
    [Sending flatfile to node 192.168.1.162:54339]
H2O node 192.168.1.161:54323 reports H2O cluster size 1
H2O node 192.168.1.163:54321 reports H2O cluster size 1
H2O node 192.168.1.162:54339 reports H2O cluster size 1
H2O node 192.168.1.162:54339 reports H2O cluster size 3
H2O node 192.168.1.163:54321 reports H2O cluster size 3
H2O node 192.168.1.161:54323 reports H2O cluster size 3
H2O cluster (3 nodes) is up
(Note: Use the -disown option to exit the driver after cluster formation)
(Press Ctrl-C to kill the cluster)
Blocking until the H2O cluster shuts down...
```

H_2O is not the typical Hadoop MapReduce job

- Long-lived Hadoop mapper tasks that take up CPUs and memory
- No Hadoop reduce tasks
- No mapper task retry (user needs to restart H_2O cloud on failure)
- No HDFS “input file” to the Hadoop map phase
  - Null splits define number of Hadoop map tasks
- No HDFS “output files” produced by the Hadoop job.
- Multithreaded
- Big Java heap (e.g. 128 GB)
- Mappers talk to each other
  - Mappers need to be run at the same time

Hadoop Configuration

0xdata

mapreduce.job.ubertask.enable	false
[mapreduceMapMemoryMb = Xmx + 10% fudge factor;]	
mapreduce.map.memory.mb	mapreduceMapMemoryMb
[mapChildJavaOpts = "-Xms" + mapperXmx + " -Xmx" + mapperXmx;]	
mapred.child.java.opts	mapChildJavaOpts
mapred.map.child.java.opts	mapChildJavaOpts
mapreduce.map.speculative	false
mapred.map.tasks.speculative.execution	false
mapred.map.max.attempts	1
mapred.job.reuse.jvm.num.tasks	1

mapreduce.map.memory.mb	128 GB
mapreduce.map.java.opts.max.heap	100 GB
yarn.nodemanager.resource.memory-mb	128 GB

ResourceManager Configuration Safety Valve for yarn-site.xml

```
<property>
    <name>yarn.scheduler.maximum-allocation-mb</name>
    <value>131072</value>
</property>
```

- Empty splits determine the mapper count
- Use Hadoop counters to avoid having Hadoop kill your job (which doesn't map anything) after 10 minutes
- Hook System.exit() to get mappers to exit properly
- YARN /etc/alternatives setting
- YARN memory settings
- YARN FIFO scheduler mapper clumping
- Hortonworks -libjars distribution unpacks the .jar file

Debugging a mapper process is an experience

- Finding the process is a challenge
- Finding the output is a challenge
- Convincing Hadoop not to delete the output on a failure is a challenge
  - Even if your logger setup code gets a chance to run, you still have to find the output.
- Attaching with standard JDWP debuggers is a challenge
  - Finding the process is (again) a challenge
  - Debugger doesn't understand the mapper environment; attach didn't work.
- Finally resorted to POST-code style printing techniques
  - "Log" output to /tmp, bypassing everything, so I get to keep it

- Automated log collection upon failure
  - Fishing out logs from a cluster of 1000 Hadoop nodes is a great job for a tool.
- Driver vs. host checking of Hadoop version
  - Helps avoid subtle errors
- Custom YARN Application Master
- Experiment further with I/O optimization for H_2O Node placement based on data locality

Your Contributions Welcome

- H2O is the premier platform for ***In-Memory*** Predictive Analytics on Big Data
  - Open Source Apache 2 license
  - H_2O mapreduce (in-memory) is not Hadoop mapreduce (HDFS)!

- H_2O can read your data from HDFS
  - Standalone or running on Hadoop

- Run H_2O on Hadoop to use CPUs and Memory from your existing Hadoop cluster
  - Get started easily with the gear you've already got


Chris Severs @ccsevers
29 Oct
Running H2O on YARN. 480 cores all maxed out. cc/ @hexadata @srisatish
from San Jose, CA
Reply
Retweeted
Favorite
More
K-Means on a Terabyte of data (Major Insurance Co.)
Now Certified on favorite hadoop vendors!
http://0xdata.com/partners
0xdata

Partners

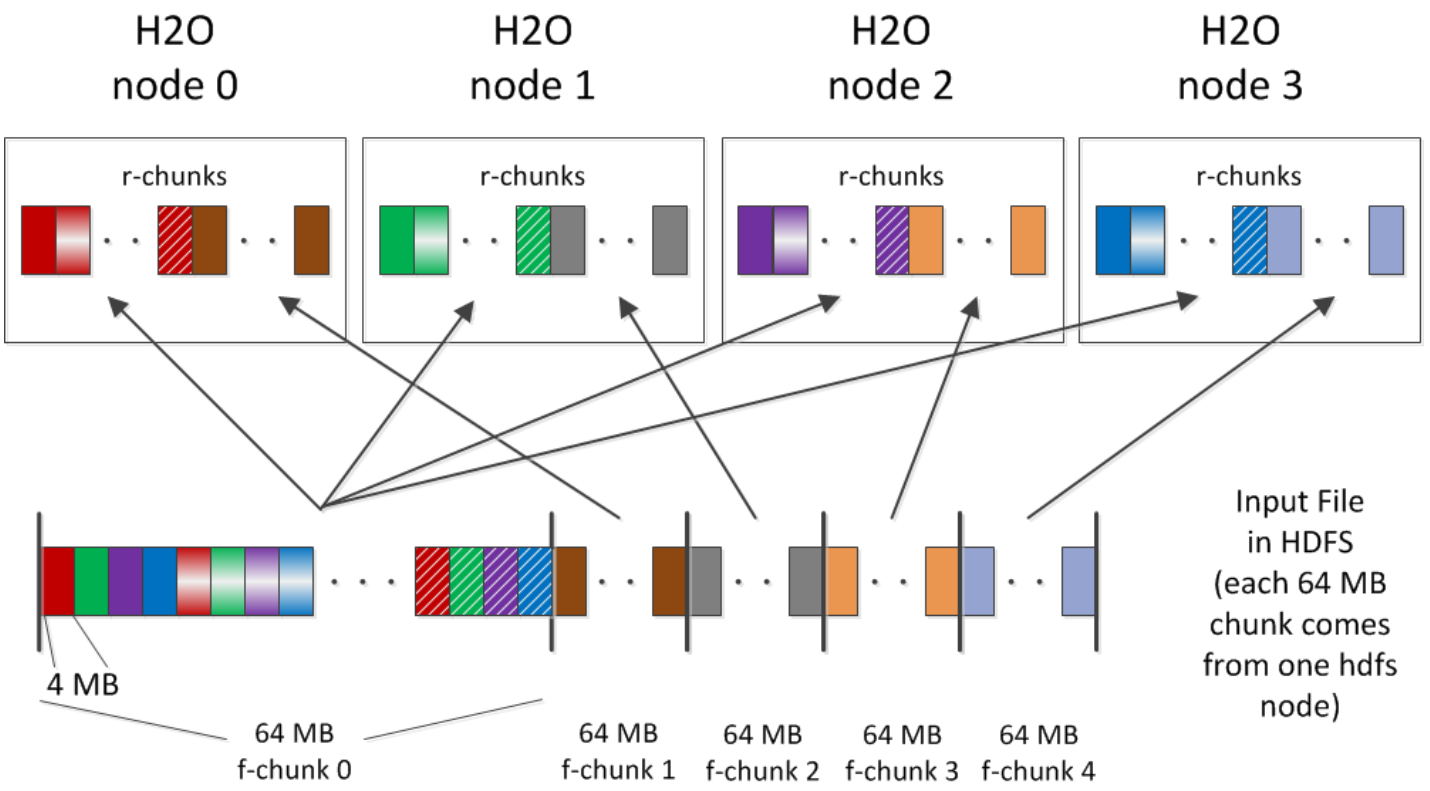

cloudera®
Ask Bigger Questions


MAPR™
TECHNOLOGIES


Hortonworks


hadoop

Raw (Pre-Parse) Data Ingestion Pattern



Why

I want to do run a Generalized Linear Model, and my data lives in HDFS.

What

Use H_2O.

How

```
(Command line) java -jar h2o.jar
(Web UI Menu)  Data -> Import HDFS
```

Why

I want to do Predictive Analytics on Big Data.

My data lives in HDFS.

I have an existing Hadoop cluster. Let's utilize its CPUs and Memory!

What

Run H_2O on Hadoop.

How

```
hadoop jar h2odriver_<hadoop_version>.jar [...]
(Web UI Menu)  Data -> Import HDFS
```