Using H_2O on Hadoop

http://0xdata.com

Document history

Date	Author	Description
2013-May-22	TMK	Initial version.

Introduction

This document discusses the integration of 0xdata's H2O framework with Hadoop.

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.

(Note that H2O can also run standalone without Hadoop. Customers interested in running a configuration without Hadoop can reference the documentation at https://github.com/0xdata/h2o/wiki for instructions.)

Glossary

0xdata	0xdata is a leading provider of scalable high-performance math solutions for big data. Visit our website at http://0xdata.com.
H2O	0xdata's scalable big-data framework for math.
Hadoop cluster	The open source big-data platform that you are probably already familiar with, have made an investment in, and wish to leverage. A Hadoop cluster consists of Hadoop nodes.
H2O node	A single JVM instance running h2o.jar. (Although multiple H2O nodes can run on a single Hadoop node, 0xdata discourages this for best performance.)
H2O cluster	A group of H2O nodes that operate together to work on jobs. H2O scales by distributing work (for example, via MapReduce) over many H2O nodes. Although multiple H2O nodes can run on a single Hadoop node, 0xdata recommends running one H2O node per Hadoop node for best performance. 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.) Ideally, the hardware on which the H2O node runs will have local disk storage for this. Network disk may be used, but will be inefficient; usage is intended to function like a temporary cache, the spilled data remains local to the node, 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.
Parse	The parse operation converts an in-memory raw data set (in CSV format, for example) into a HEX format data set.
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

- ? GB memory
- (preferred) ? GB local disk (for spilling)
- (not preferred) ? GB network disk (for spilling)

Supported Hadoop software distributions

- Cloudera CDH3 (3.5 is tested)
- Cloudera CDH4
- MapR

How H2O Nodes are Deployed on Hadoop

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

For batch mode use of H2O, an H2O cluster may be created for the purpose of one computation or related set of computations. The cluster is created, the work is performed, the cluster dissolves, and resources are returned to Hadoop.

For interactive use of H2O, the user may create what appears like a very long running batch job, where the H2O cluster stays up for an extended period of time, and resources on the Hadoop node are also claimed for an extended period of time (i.e., until the cluster dissolves).

At the present time, 0xdata recommends batch mode as the most straightforward integration path with Hadoop.

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 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). Ideally do this in a way such that only one H2O lands on each Hadoop node, and all mappers can be started simultaneously.
2. No work may be sent to the H2O nodes until they find each other and form a cluster. (Effectively, this means pause 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. (Nodes are symmetric.)

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 instance. The Java heap is specified via Xmx, and the user must plan for this memory to be fully utilized.
Network I/O	An H2O node generally does network I/O to read in the initial data set. H2O nodes also communicate (potentially heavily) during the parse step, and during an algorithm job (e.g. GLM) running on top of H2O's MapReduce.
Disk I/O	An H2O node uses local disk to temporarily spill (otherwise known as swap) data to free up space in the Java heap.
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 currently has two options for interacting with H2O. The first 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.

[Talk about R here?]

The second way for the user to interact with H2O is to use scripts which talk to the H2O embedded web server via the REST API. The REST API accepts HTTP requests and returns JSON-formatted responses.

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 Locality)

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.

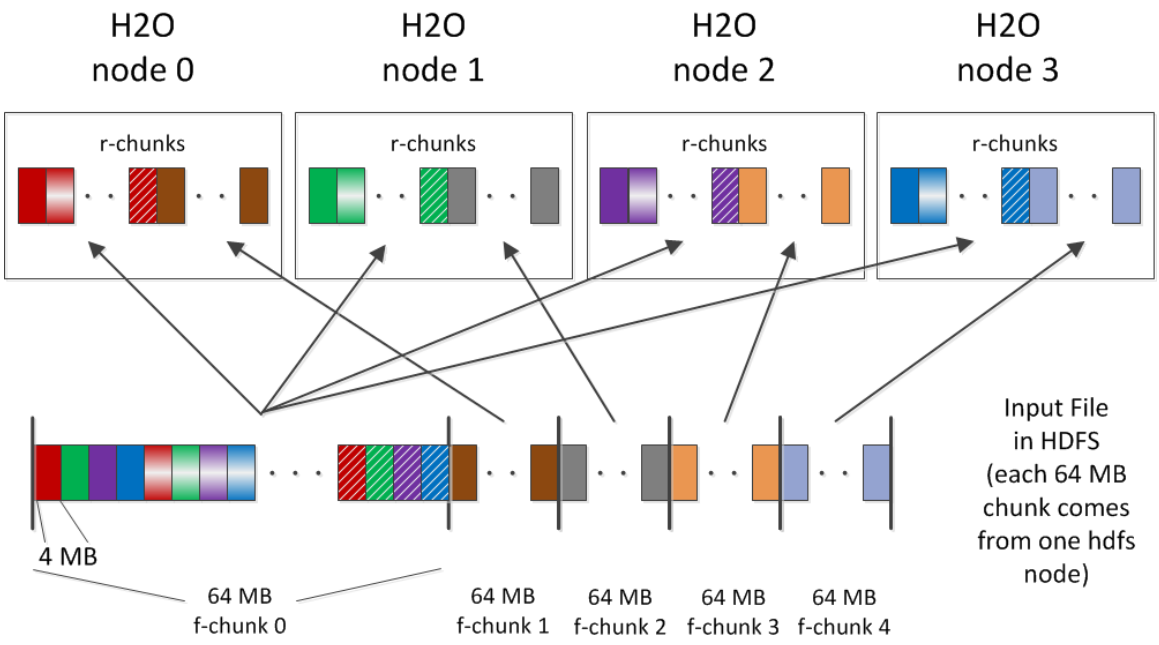
HDFS files are split across HDFS nodes in 64 MB chunks (referred to as file chunks, or f-chunks in the diagram "Raw Data Ingestion Pattern").

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. Plan for the entire input file to be transferred across the network (albeit in parallel pieces).

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.

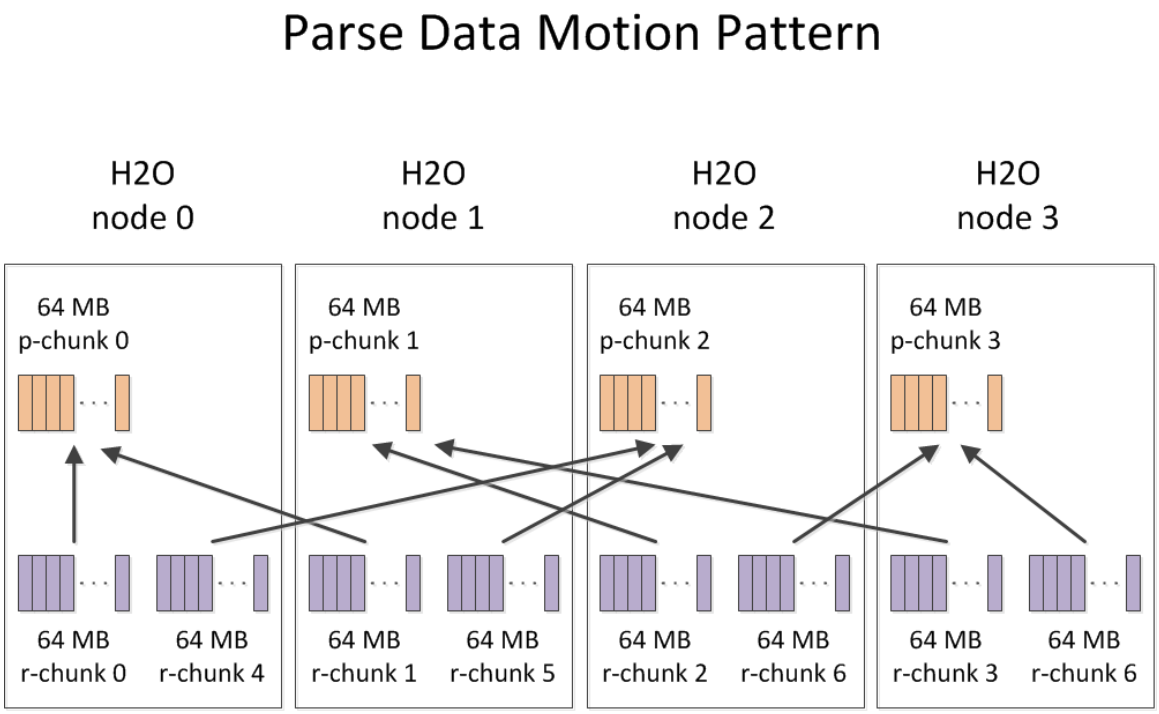
Raw (Pre-Parse) Data Ingestion Pattern



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 usually reduces the overall in-memory data size because the HEX storage format is typically more efficient than the text input data. Note that (as shown in the diagram) the parse process usually 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, for some algorithms it does not need to move again (as with GLM, for example), and no additional data I/O is required.



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

How MapReduce Works on H2O

MapReduce in H2O is independent of MapReduce in Hadoop. 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 cloud 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 through a web browser or through the REST API.

Examples for Starting and Monitoring a Batch H2O Job

[Need an example.]

FAQ

[See if any additional Q/A from AMEX call were unanswered above.]