Using H_2O on Hadoop

http://0xdata.com

Document history

Date	Author	Description
2013-May-22	TMK	Initial version.
2013-June-22	TMK	Updated algorithm picture.

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.

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 (3.5 is tested)
- Cloudera CDH4
- MapR 2.1.1

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 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.

For interactive use of H2O, we recommend deploying on a Hadoop cluster dedicated to this purpose. 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.

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 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. (There is no "master" H2O node.)

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 currently has various 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.

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.

A third way is for the user to interact with the H2O console, which provides an R-like command-line environment for working directly with their data.

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.

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.

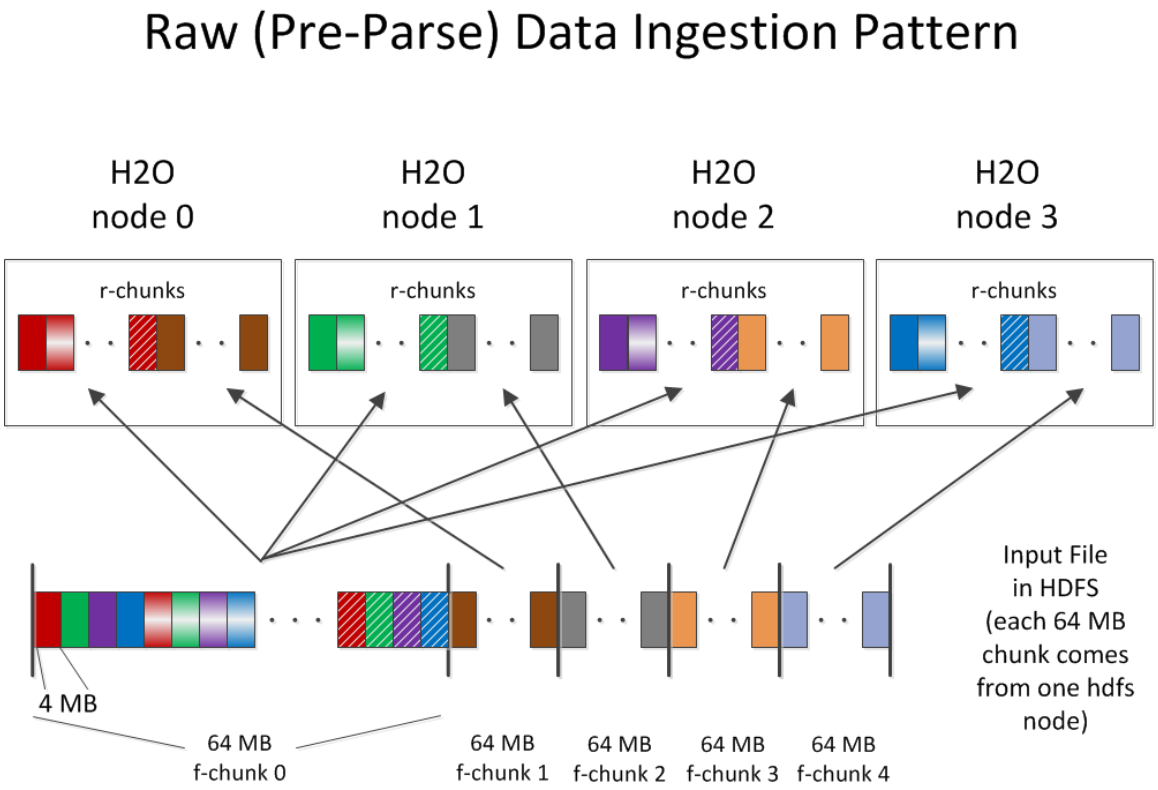
HDFS files are split across HDFS nodes in 64 MB chunks (referred to as file chunks, or f-chunks in the diagram "Raw Data Ingestion Pattern").

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.

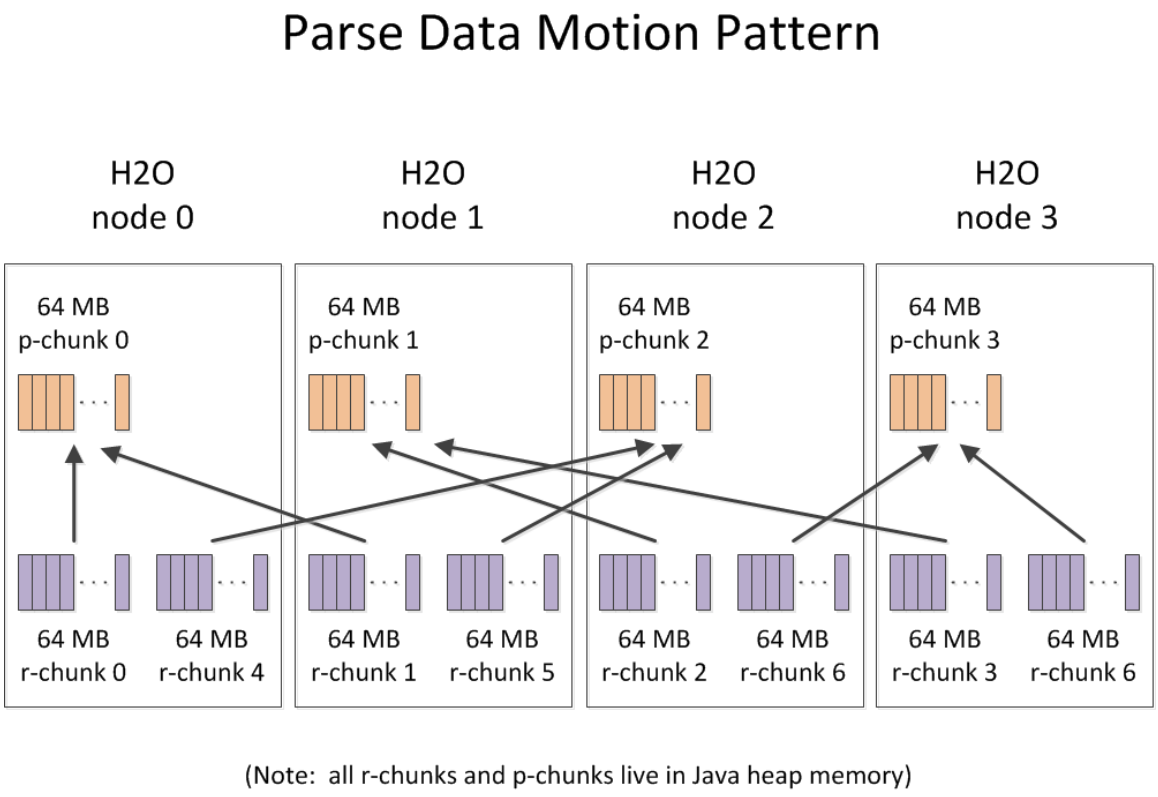
The first 64 MB f-chunk is sprayed across the H2O cluster in 4 MB pieces. This ensures the data is spread across the H2O cluster for small data sets and parallelization is possible even for small data sets. Subsequent 64 MB f-chunks are sprayed across the H2O cluster as whole 64 MB pieces.



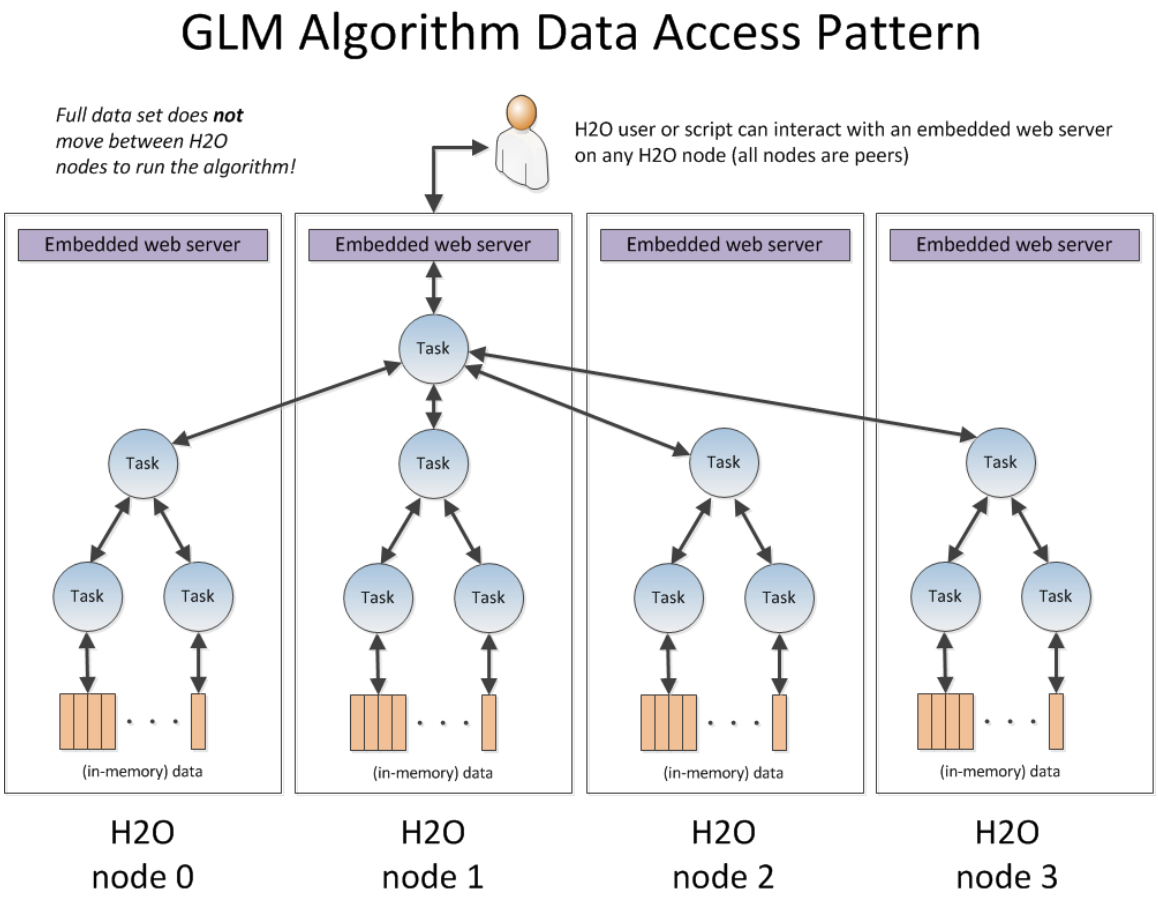
After ingestion, the parse process occurs (see "Parse Data Motion Pattern" diagram). Parsing converts 64 MB in-memory r-chunks (raw unparsed data) into 64 MB in-memory p-chunks (parsed data, which is in the HEX format). Parsing may reduce the overall in-memory data size because the HEX storage format 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.



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



Output from H2O

Output from H2O jobs is written to HDFS.

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 through a web browser or through the REST API.