HAMR Technical Report: Machine Learning

Naïve Bayes

HAMR vs Hadoop/Mahout

Version 0.3
Nov 10, 2014

hamr-info@hamrtech.com
www.hamrtech.com
302.797.4267

Key Findings: ***HAMR™ Delivers 80X+ Performance Improvement for the Naïve Bayes Benchmark.***

HAMR is significantly more efficient than MapReduce in both memory and processor utilization. Memory efficiency allows far less spilling to disk; direct processing is *one* of the sources of greater processor utilization. These efficiencies translate into speedups of up to 87X versus Mahout/Hadoop. With HAMR, as data size increases, performance improvements accelerate dramatically.

HAMR is able to achieve these improvements with a simple API that does not require the user to write a more complex program. A classification case can be written in a single source file of 250 lines.

Introduction

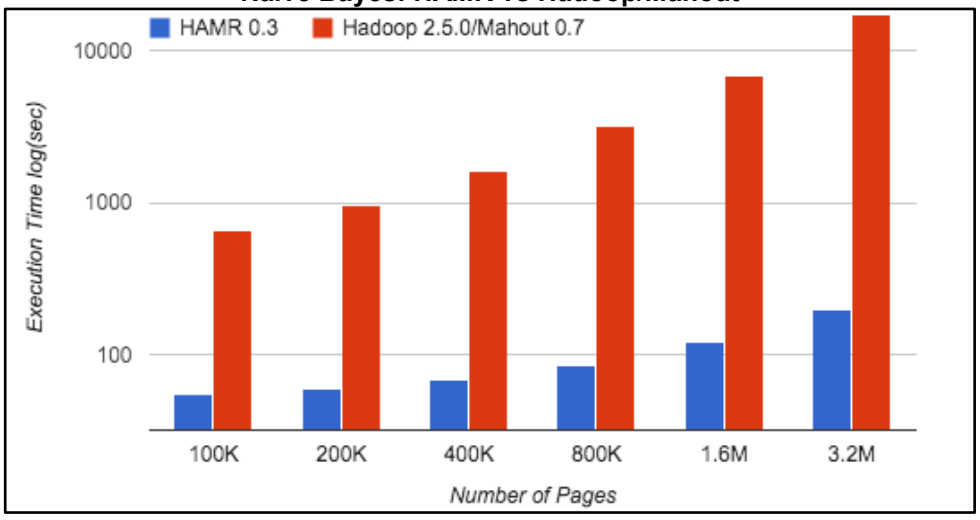
The Naïve Bayes Classifier algorithm uses an extension of Bayes' theorem of conditional probability to categorize text. It assumes that all words in a document appear with independent probability, regardless of the document's category or what other words appear. This is a two-step process. First, the classifier is trained using sample data, and second, the classifier applies this training data to an unknown input. In this document, the performance of the Naïve Bayes benchmark is compared between Apache Hadoop with Mahout versus HAMR 0.3.

There are differences between the HAMR and Mahout implementations of the Naïve Bayes trainer. However, the underlying Naïve Bayes training and classifying algorithm is the same. Both HAMR and Mahout implementations provide correct results.

Results

The following graph presents a performance comparison between the HAMR and Hadoop implementations. Execution time is recorded in seconds, collected as an average of three runs for data sets ranging from 100,000 to 3,200,000 pages (450 MB–14.4 GB) with 50 classes.

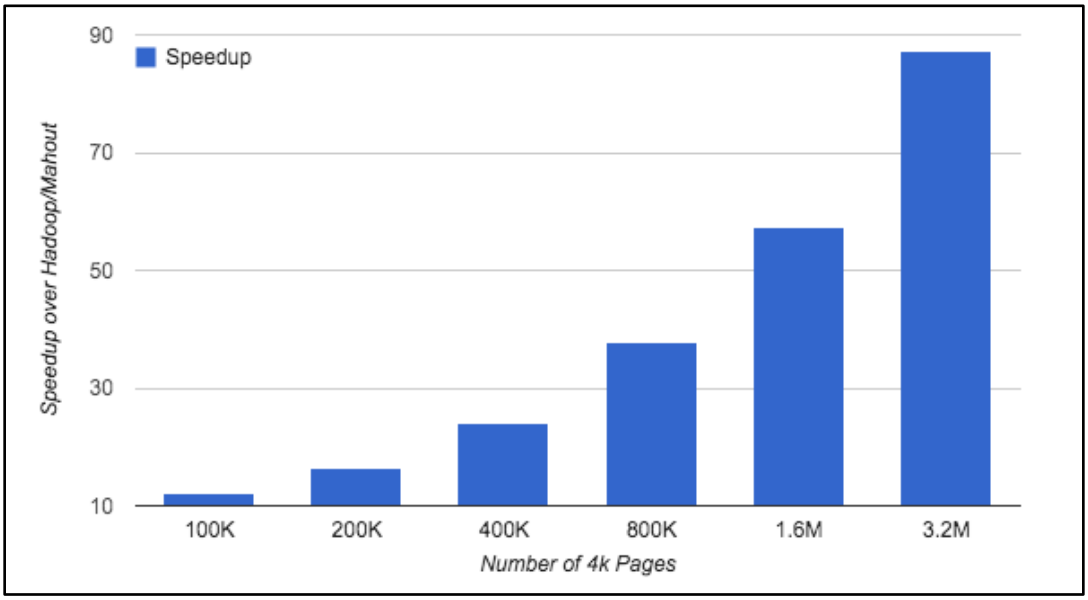
Naïve Bayes: HAMR vs Hadoop/Mahout



For data sets from 100,000 to 3,200,000 pages HAMR clearly outperforms Mahout/Hadoop. Note that Mahout spends the majority of its time in the seq2sparse processes, which vectorize the documents based on different features. The features are calculated according to term frequency–inverse document frequency (tfidf). After the feature selection and vectorization process the input data to Mahout nbtrain is very small. Therefore, it takes a very small portion of time to do model training.

The next diagram summarizes HAMR's speedup over Hadoop and Mahout. This diagram highlights the dramatic acceleration of the performance improvement as data size increases.

Naïve Bayes: HAMR Speed Improvement Over Hadoop/Mahout[1]



	Minimum	Maximum	Average
Speedup over Hadoop	12.20	87.11	39.18

Improvement Sources

HAMR's unique KeyValueStore components allow for far less spilling to disk than the Hadoop/Mahout implementation. Mahout's Naïve Bayes trainer produces twelve files, several of them split among partitions (*.part0000, *.part0001, etc), only two of which are used by the Mahout classifier. HAMR's trainer produces only two files, split in the same manner. This extra work that Hadoop does may account for some of HAMR's speedup. Additionally, HAMR processes documents directly, while Mahout pre-processes the documents into sparse vectors with the seq2sparse tool, which accounts for the majority of Hadoop's processing time.

HAMR Support for Naïve Bayes

The enterprise edition of HAMR includes a set of components that implement the Naïve Bayes text classification benchmark. This provides three classes that are used in training and running the classifier.

[1] (Hadoop with Mahout running time) / (HAMR running time)

These classes are wrappers around a series of Flowlets[2] and can be placed into a HAMR Workflow like any normal Flowlet. Internally, these classes use several specialized KeyValueStores. A KeyValueStore (KVS) is a node in the workflow graph that is a reliable, distributed data structure for storing key/value pairs. It has ports for connecting to Flowlets and Resources. This allows the training data to be maintained in memory without having to save to disk.

The Naïve Bayes trainer is treated like a Flowlet with a single input. KV pairs on this input have a word as the key and its category as the value. The trainer can then be supplied to a classifier, which is also treated like a Flowlet with a single input. The trainer can also write its contents to disk for later use. The training process is shown in Figure 1.

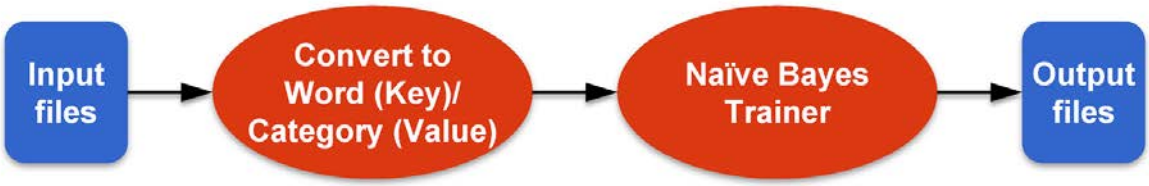


Figure 1: Training the Naïve Bayes classifier

Classifiers use data produced by trainers to classify unknown documents. They can take the data straight from a trainer or read the data from a file previously produced by a trainer. A helper class exists for reading the training data files. The process for classifying an unknown document in the same HAMR instance as a Naïve Bayes Trainer, but separate Workflow, is shown in Figure 2.

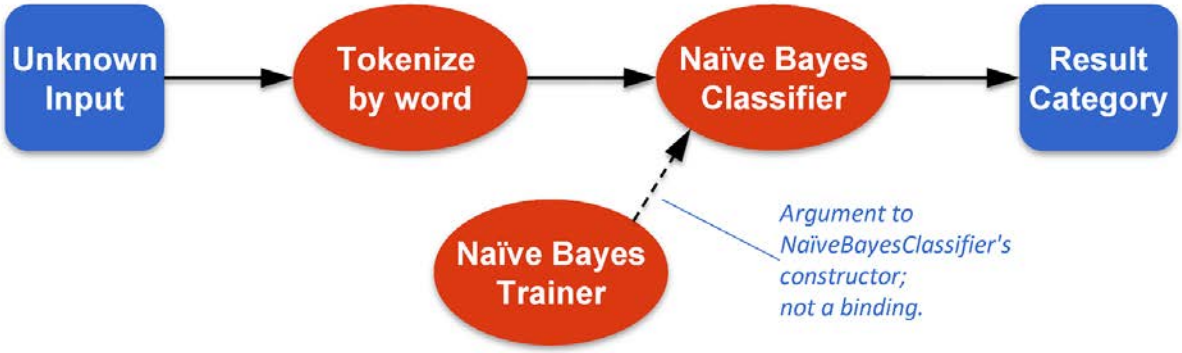


Figure 2: Running the Naïve Bayes classifier

Conclusion

HAMR™ Delivers 80X+ Performance Improvement for the Naïve Bayes Benchmark.

HAMR is significantly more efficient than Hadoop in both memory and processor utilization. Memory efficiency allows far less spilling to disk; direct processing is *one* of the sources of greater processor utilization. These efficiencies translate into speedups of up to 87X versus Hadoop.

With HAMR, as data size increases, performance improvements accelerate dramatically.

HAMR is able to achieve these improvements with a simple API that does not require the user to write a more complex program. A classification case can be written in a single source file of 250 lines.

[2] Flowlet is the base class from which the majority of nodes in a workflow will be composed. Flowlets have associated Bindable instances through which they can bind with other flowlets to establish edges for passing data through the workflow.

Testbed Description

We compared our HAMR implementation of the Naïve Bayes benchmark to the Hadoop/Mahout implementation and its input generator, provided in the Intel HiBench Benchmarking Suite running Hadoop 2.5.0.

The input data is automatically generated documents whose words follow the zipfian distribution using the default linux file /usr/share/dict/linux.words. This includes only the training portion of the classifier.

HAMR and Hadoop read the input files from and write the results to the HDFS file system. We configure Hadoop and HDFS according to the convention set by Hortonworks (http://goo.gl/4XcCbY). The HDFS block replication is set to 3 with a block size of 128MB. While these parameters are tweakable for a given benchmark, we used a configuration representative of standard clusters.

The testing environment is a four node cluster with the specifications listed below. The benchmark was executed on the same cluster, with the same number of compute nodes, and with the same input data for Hadoop and HAMR.

- System: 4 Node Dell PowerEdge R420 2 x Xeon® E5-2450 32 GB 1600 MHz DDR3, 4 x SATA 2 (3 GB/s)
- Network: 4 x QDR Infiniband
- OS: CentOS release 6.5
- Java: JDK 1.7.0 u55

HAMR™ is the next generation in-memory real-time software appliance enabling out of the box Big Data analytics. While maintaining the best aspects of MapReduce found in Apache Hadoop 2.0, HAMRTech has developed a novel Big Data programming model and runtime inspired by MIT dataflow. HAMR leverages innovative Flowlets™ and Key/Value Stores to reduce idle time and costly spilling of data to disk, saving both time and energy. Furthermore, HAMR provides access to all enterprise data, and supports batch, streaming, and real-time analytics, making HAMR a fast, efficient, and more complete Big Data solution.