Background and Related Work

Background: Hadoop

Hadoop is an open-source framework that aims to solve problems associated with large scale data storage and processing. The framework itself was based off of two papers released by Google: “The Google File System” in 2003 and “MapReduce: Simplified Data Processing on Large Clusters,” released in 2004. Google itself had been utilizing the described filesystem and MapReduce approach effectively to solve it’s own scalability problems and following the release of the papers, many other companies became interested in emulating Google’s approach.

Apache Hadoop grew out of Apache Nutch, an open source web search engine. Following the publication of the Google papers on their file system and MapReduce algorithm, the creators of Nutch attempted to integrate the ideas put forth by Google into their own project. By early 2006 the Hadoop project was split off from Nutch to become an open source implementation of the Google file system and data processing methods without being tied to any specific project in particular.

Shortly after its release to the open source community, Yahoo! provided a dedicated team to work on the Hadoop project full time, attempting to turn the project into something that could truly run at “internet-scale.” Many other companies quickly joined the fold as the project began to gain traction and perform well on many different benchmarks.

In the time since its creation, the Hadoop framework has been adopted by a who’s who of large tech companies that deal with any sort of scalability issues: Facebook, Amazon, Rackspace, IBM, Oracle, Dell, Intel, Microsoft, Adobe, AOL, EBay, Hulu, Last.fm, LinkedIn, The New York Times, Spotify, Twitter, Yahoo!, and many, many more. It has become the go-to tool for handling big data issues and new applications built upon the framework are under constant development.

Background: HDFS

HDFS is a central component of the Hadoop framework and forms the primary (and default) distributed storage mechanism used by Hadoop applications. HDFS is written in Java and designed to integrate well with the MapReduce paradigm. It is also designed to be fault tolerant, scalable, and easy to balance and expand.

Each HDFS cluster will contain a namenode, whose responsibility it is to keep track of where data blocks are stored. Data itself is stored on datanode members, who coordinate with the central namenode to inform it of their file contents and replication responsibilities.

HDFS is fault tolerant because of it’s ability to mandate replication of data between datanodes such that if any one goes down, the data can always be retrieved from another node. In the interest of data locality, HDFS will attempt at first to co-locate data to a nearby datanode, so that the data can be transferred to it quickly and only a relatively small amount of cluster network bandwidth is consumed. Once it has replicated a data block to another rack-local datanode, HDFS can be configured to also back it up to another datanode not present on the same rack, to make sure that the data is protected even if the entire rack

were to fail. This three-tiered replication is the default methodology employed by HDFS when operating in a distributed mode, though it can be configured higher (and to replicate even to other data centers, etc) or lower, in case resources are limited and the data being stored is not of any tremendous value.

Background: MapReduce

MapReduce is another central component of the Hadoop framework. It traces its origins back to Google's 2004 paper, "MapReduce: Simplified Data Processing on Large Clusters." MapReduce itself is a programming model for processing vast amounts of data through a simple, parallel, distributed algorithm. The MapReduce implementation provided by the Hadoop framework can be utilized by a variety of different programming languages: Java by default, C++ with Hadoop Pipes, and Ruby, Python, BASH, etc. through Hadoop Streaming (communicates over STDOUT).

As the name implies, the MapReduce algorithm consists of two phases: a map phase and a reduce phase. Each of the phases take key-value pairs as input and also output key-value pairs. The specifics and types of these pairs are defined by the programmer.

The Map() function performs filtering and sorting of the input data while the Reduce() function generally performs some form of summary operation across the key-value pairs that were output by the Map() function.

Background: YARN

The Hadoop MapReduce implementation underwent a considerable overhaul in the Hadoop 2 (MapReduce 2.x) releases. In previous versions of MapReduce, a centralized JobTracker node kept track of which nodes (TaskTrackers) had empty Map or Reduce "slots" that were available. In addition to monitoring the available resources in the cluster, the JobTracker was responsible for scheduling and monitoring all jobs that were submitted to the cluster. On many large clusters, the dual roles played by the JobTracker began to test the scalability of the Hadoop architecture.

In addition to separating the old JobTracker's responsibilities into two separate daemons (the new ResourceManager and ApplicationManager), YARN also abstracts the Hadoop architecture to allow many distributed applications other than just MapReduce to take advantage of cluster resources. YARN itself is an acronym for "Yet Another Resource Negotiator" and allows applications to request cluster resources in the form of containers (a specified amount of RAM, CPU) that allows it to act in many ways like a distributed operating system.

When a MapReduce application is now launched, an ApplicationMaster is launched on the machine that submitted the job. The ApplicationMaster is responsible to coordinating with the ResourceManager concerning the application's execution on the cluster. If enough NodeManagers (running on all slave nodes) in the cluster indicate that they have the requisite resources to run the application, the ResourceManager will grant the ApplicationMaster container leases upon which it will attempt to schedule Map/Reduce tasks, optimizing whenever it can for locality to the data to be processed. Overall the

addition of YARN to the Hadoop framework has made it much more flexible and scalable without sacrificing any performance on most benchmarks relative to Hadoop 1.

Background: Hive

Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis. It was initially developed at Facebook but now the main contributors to it also include NetFlix and Amazon.

Hive works by letting the user define a schema with which to process a large data set. This schema was previously (Hadoop 1) defined in HCatalog but now this project has been rolled into Hive. Once the schema is provided and the Hive system can find it in a specified traditional RDBMS, Hive provides an SQL-like query language called HiveQL that allows users inexperienced with the MapReduce/Hadoop framework to interact with the datasets through standard queries that are implicitly converted to MapReduce queries by Hive. Even for users somewhat experienced with MapReduce, writing a custom MapReduce job still requires a fair amount of time. Internally, Hive converts the submitted HiveQL queries into directed acyclic graphs of MapReduce jobs. Unless it's something that will run extremely often or require some very eccentric features of the MapReduce architecture, Hive provides a relatively simple and intuitive way to learn from large aggregate datasets.

Background: Pig

Pig was developed by Yahoo! Research in 2006 as a higher layer of abstraction for processing large datasets. Pig's main component is the Pig Latin language, which lets users easily define a series of transformations applied to input data to produce output. It "feels" more like a traditional programming language with the notable exception of not providing any control flow operations; generally if you want such things you can use a language such as Java for them and then call the Pig Latin statements therein when appropriate (all Hadoop components integrate fairly seamlessly with Java). Like HiveQL, under the covers Pig converts the Pig Latin statements into a series of MapReduce jobs.

What separates Pig from HiveQL is that it is designed to do a lot more than simple queries of the dataset. With few exceptions, if you can do something in MapReduce, there is the ability to do it in Pig with much less code. This includes things such as loading data, sorting, filtering, grouping, joining, moving, etc. While Pig Latin's higher layer of abstraction does impose something of a performance penalty versus regular MapReduce, the gap has been narrowing considerably in recent years.

Background: Mahout

Mahout is an extensive machine learning project from the Apache Software Foundation. It began as a subproject of Apache's Lucene project, which is an open-source text search engine (Nutch grew out it also). It focuses especially on distributed and scalable machine learning applications. It's three primary functions currently are recommender

engines (collaborative filtering), clustering, and classification.

Mahout is written entirely in Java and most algorithms (though not all) are built to utilize a Hadoop cluster. It is under extremely active development and new algorithms and implementations are being added all the time. Though it does require data from Hadoop to be in very particular formats, it provides a extremely useful and powerful tool for learning from the sort of large datasets commonly seen on Hadoop.

Background: Hadoop benchmarks

A widely-publicized early example of Hadoop's power came in 2007, when the New York Times used Hadoop to sort through over four terabytes of archives and convert them to PDF files available over the web. Not long after this example, Hadoop broke the world record in sorting a terabyte of data. Running on 910 commodity nodes, Hadoop was able to sort a terabyte in under 3 and a half minutes. Never a company to rest on its laurels, Google in November of 2008 shattered the terabyte sort record by using its proprietary MapReduce implementation to get it done in a mere 68 seconds. It also successfully sorted a petabyte of data in six hours and two minutes. The current record for terabyte sort on Hadoop appears to belong to MapR with a time of 54 seconds. While gigabyte-sized sorts pale in comparison to these impressive achievements, they allow some degree of standardized benchmarking and do a fine job of demonstrating where your cluster is relative to the large ones seen in industry environments.

Related Work: Hadoop Tuning

Much prior work has been done in the field of Hadoop tuning. Given the nature of the Hadoop platform, customizing a cluster based on the actual hardware and networking resources available can provide tremendous performance benefits. For example, adjusting the io.sort.mb property to ensure that Map outputs can reside in memory rather than be spilled to disk (then having to be subsequently pulled back to memory for input to Reduce() and spilled back to disk) can achieve huge performance gains. Since network bandwidth tends to be the most precious resource in actual industry Hadoop clusters, compressing the output of Map and Reduce stages can also provide large increases in performance. Many of these basic tweaks have been described in many of the papers I've listed yet what is lacking to this point are many papers or guidelines concerning the fine tuning and optimization of Hadoop clusters running the new YARN architecture.

In addition to the absence of information regarding optimizations for the new YARN architecture, there is also not a large amount of information pertaining to running Hadoop clusters on somewhat "less than" commodity hardware. Currently in industry, a "commodity" Hadoop node in term of hardware would be somewhere along the lines of 2+ quad-core 2.5+ GHz CPUs with 32-256 GB ECC RAM, and 4+ 1TB+ SATA disks. So while I certainly did benefit from the related works I've referenced, not a lot of attention has been focused on running Hadoop clusters efficiently on hardware slightly out of date, a situation that might be common among academic and/or recreational Hadoop enthusiasts.

Related Work: Using Mahout for clustering, classification of text data

Mahout excels at text-based machine learning applications, and many have been performed before. Of note is one that uses Wikipedia's massive dump data to classify their latest articles in various categories. While this project did not incorporate any features related to interacting with multiple massive RDBMS systems (the dump data is in standard XML form), the techniques and features of Mahout utilized are roughly similar to the analysis I perform on the Leipzig Corpora collection, which to my knowledge has never been done before. And again, my analysis is performed on hardware considerably less powerful than that utilized by the authors of that particular paper.

References

S. Ghemawat, H. Gobioff and S. Leung. "The Google file system," In Proc. of ACM Symposium on Operating Systems Principles, Lake George, NY, (Oct 2003): pp 29–43.

J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters." Communications of The ACM 51-1 (2008): 107–113.

Derek GottFrid, "Self-service, Prorated Super Computing Fun!" 1 November 2007, http://http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/ .
"Sorting 1PB with Mapreduce," 21 November 2008, http://http://googleblog.blogspot.com/2008/11/sorting-1pb-with-mapreduce.html.

Herodotou, Herodotos. "Hadoop performance models." *arXiv preprint arXiv:1106.0940* (2011).

Joshi, Shrinivas B. "Apache hadoop performance-tuning methodologies and best practices." In *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering*, pp. 241-242. ACM, 2012.

Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz and Ion Stoica, "Improving MapReduce Performance in Heterogeneous Environments," University of California, Berkeley. Accessed 8 Feb 2014 from https://www.usenix.org/legacy/events/osdi08/tech/full_papers/zaharia/zaharia.pdf.

C. Olston, B. Reed, U. Srivastava, R. Kumar and A. Tomkins, "Pig Latin: A Not-So-Foreign Language for Data Processing."ACM SIGMOD 2008, June 2008.

A. Pavlo et. al. "A Comparison of Approaches to Large-Scale Data Analysis." In Proc. of ACM SIGMOD, 2009.

“MapR and Google Compute Engine Set New World Record for Hadoop TeraSort,” 24 October 2012, http://www.businesswire.com/news/home/20121024005285/en/MapR-Google-Compute-Engine-Set-World-Record.

Anil, Robin, Ted Dunning, and Ellen Friedman. *Mahout in action*. Manning, 2011.

Esteves, Rui Maximo, and Chunming Rong. "Using Mahout for clustering Wikipedia's latest articles: a comparison between K-means and fuzzy C-means in the cloud." *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. IEEE, 2011.

Biemann, Chris, et al. "The Leipzig Corpora Collection-monolingual corpora of standard size." *Proceedings of Corpus Linguistic 2007* (2007).

White, Tom. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

Esteves, Rui Maximo, Rui Pais, and Chunming Rong. "K-means clustering in the cloud--a Mahout test." In *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*, pp. 514-519. IEEE, 2011.