MapReduce

From Wikipedia, the free encyclopedia

MapReduce is a software framework implemented by Google to support parallel computations over large (multiple petabyte[1]) data sets on clusters of commodity computers. This framework is largely taken from map and reduce functions commonly used in functional programming,[2] although the actual semantics of the framework are not the same[3].

MapReduce implementations have been written in C++, Java and other languages.

Contents

- 1 Logical view
  - 1.1 Example
- 2 Dataflow
  - 2.1 Input reader
  - 2.2 Map function
  - 2.3 Partition function
  - 2.4 Comparison function
  - 2.5 Reduce function
  - 2.6 Output writer
- 3 Distribution and reliability
- 4 Uses
- 5 Implementations
- 6 References
- 7 External links
  - 7.1 Papers
  - 7.2 Articles
  - 7.3 Software

Logical view

The *Map* and *Reduce* functions of *MapReduce* are both defined with respect to data structured in (key, value) pairs. *Map* takes one pair of data with a type on a data domain, and returns a list of pairs in a different domain:

```
Map(k1,v1) -> list(k2,v2)
```

The map function is applied in parallel to every item in the input dataset. This produces a list of (k2,v2) pairs for each call. After that, the MapReduce framework collects all pairs with the same key from all lists and groups them together, thus creating one group for each one of the different generated keys.

The *Reduce* function is then applied in parallel to each group, which in turn produces a collection of values in the same domain:

```
Reduce(k2, list (v2)) -> list(v2)
```

Each *Reduce* call typically produces either one (k2,v2) pair or an empty return, though one call is allowed to return more than one pair. The returns of all calls are collected as the desired result list.

Thus the MapReduce framework transform a list of (key, value) pairs into a list of values. This behavior is different from the functional programming map and reduce combination, which accepts a list of arbitrary values and returns one single value that combines *all* the values returned by map.

Example

The canonical example application of MapReduce is a process to count the appearances of each different word in a set of documents:

```
map(String name, String document):
 // key: document name
 // value: document contents
 for each word w in document:
   EmitIntermediate(w, 1);

reduce(String word, Iterator partialCounts):
 // key: a word
 // values: a list of aggregated partial counts
 int result = 0;
 for each v in partialCounts:
   result += ParseInt(v);
 Emit(result);
```

Here, each document is split in words, and each word is counted initially with a "1" value by the *Map* function, using the word as the result key. The framework puts together all the pairs with the same key and feeds them to the same call to *Reduce,* thus this function just needs to sum all of its input values to find the total appearances of that word.

Dataflow

The frozen part of the MapReduce framework is a large distributed sort. The hot spots, which the application defines, are:

- an *input reader*
- a *Map* function
- a *partition* function
- a *compare* function
- a *Reduce* function
- an *output writer*

Input reader

The *input reader* divides the input into 16MB to 128MB splits and the framework assigns one split to each *Map* function. The *input reader* reads the data from stable storage (typically a distributed file system like Google File System) and generates key/value pairs.

A common example will read a directory full of text files and return each line as a record.

Map function

Each *Map* function takes a series of key/value pairs, processes each, and generates zero or more output key/value pairs. The input and output types of the map can be (and often are) different from each other.

If the application is doing a word count, the map function would break the line into words and output the word as the key and 1 as the value.

Partition function

The output of all of the maps is allocated to particular reduces by the applications's *partition* function. The *partition* function is given the key and the number of reduces and returns the index of the desired *reduce*.

A typical default is to hash the key and modulo the number of *reduces*.

Comparison function

The input for each *reduce* is pulled from the machine where the *map* ran and sorted using the application's *comparison* function.

Reduce function

The framework calls the application's *reduce* function once for each unique key in the sorted order. The *reduce* can iterate through the values that are associated with that key and output 0 or more key/value pairs.

In the word count example, the *reduce* function takes the input values, sums them and generates a single output of the word and the final sum.

Output writer

The *Output Writer* writes the output of the reduce to stable storage, usually a distributed file system, such as Google File System.

Distribution and reliability

MapReduce achieves reliability by parceling out a number of operations on the set of data to each node in the network; each node is expected to report back periodically with completed work and status updates. If a node falls silent for longer than that interval, the master node (similar to the master server in the Google File System) records the node as dead, and sends out the node's assigned work to other nodes. Individual operations use atomic operations for naming file outputs as a double check to ensure that there are not parallel conflicting threads running; when files are renamed, it is possible to also copy them to another name in addition to the name of the task (allowing for side-effects).

The reduce operations operate much the same way, but because of their inferior properties with regard to parallel operations, the master node attempts to schedule reduce operations on the same node, or as close as possible to the node holding the data being operated on; this property is desirable for Google as it conserves bandwidth.

Uses

MapReduce is useful in a wide range of applications, including: "distributed grep, distributed sort, web link-graph reversal, term-vector per host, web access log stats, inverted index construction, document clustering, machine learning, statistical machine translation..." Most significantly, when MapReduce was finished, it was used to completely regenerate Google's index of the World Wide Web, and replaced the old *ad hoc* programs that updated the index and ran the various analyses. [4]

MapReduce's stable inputs and outputs are usually stored in a distributed file system. The transient data is usually stored on local disk and fetched remotely by the reduces.

Implementations

- The Google MapReduce framework is implemented in C++ with interfaces in Python and Java.
- The Hadoop project [1] (http://wiki.apache.org/lucene-hadoop/) is a free open source Java MapReduce implementation.
- Phoenix [2] (http://csl.stanford.edu/~christos/sw/phoenix/) is a shared-memory implementation of MapReduce implemented in C.
- MapReduce has also been implemented for the Cell Broadband Engine, also in C. [3] (http://sourceforge.net/projects/mapreduce-cell)
- MapReduce has been implemented on NVIDIA GPUs (Graphics Processors) using CUDA [4] (http://www.cse.ust.hk/gpuqp/Mars.html) .
- Qt Concurrent [5] (http://labs.trolltech.com/page/Projects/Threads/QtConcurrent) is a simplified version of the framework, implemented in C++, used for distributing a task between multiple processor cores.
- CouchDB [6] (http://incubator.apache.org/couchdb/docs/overview.html) uses a MapReduce framework for defining views over distributed documents
- Skynet [7] (http://skynet.rubyforge.org/) is an open source Ruby implementation of Google's MapReduce framework

References

1. ^ http://news.cnet.com/8301-10784_3-9955184-7.html
2. ^ "Our abstraction is inspired by the map and reduce primitives present in Lisp and many other functional languages." -"MapReduce: Simplified Data Processing on Large Clusters" (http://labs.google.com/papers/mapreduce.html) , by Jeffrey Dean and Sanjay Ghemawat; from Google Labs
3. ^ "Google's MapReduce Programming Model -- Revisited" (http://www.cs.vu.nl/~ralf/MapReduce/paper.pdf) — paper by Ralf Lammel; from Microsoft
4. ^ "As of October, Google was running about 3,000 computing jobs per day through MapReduce, representing thousands of machine-days, according to a presentation by Dean. Among other things, these batch routines analyze the latest Web pages and update Google's indexes." "How Google Works" (http://www.baselinemag.com/article2/0,1540,1985048,00.asp)

- Dean, Jeffrey & Ghemawat, Sanjay (2004). "MapReduce: Simplified Data Processing on Large Clusters" (http://labs.google.com/papers/mapreduce.html) . Retrieved Apr. 6, 2005.

External links

Papers

- "MapReduce: Simplified Data Processing on Large Clusters" (http://labs.google.com/papers/mapreduce.html) — paper by Jeffrey Dean and Sanjay Ghemawat; from Google Labs
- "Interpreting the Data: Parallel Analysis with Sawzall" (http://labs.google.com/papers/sawzall.html) — paper by Rob Pike, Sean Dorward, Robert Griesemer, Sean Quinlan; from Google Labs
- "Evaluating MapReduce for Multi-core and Multiprocessor Systems" (http://csl.stanford.edu/%7Echristos/publications/2007.cmp_mapreduce.hpca.pdf) — paper by Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, and Christos Kozyrakis; from Stanford University
- "MapReduce for the Cell B.E. Architecture" (http://pages.cs.wisc.edu/~dekruijf/docs/mapreduce-cell.pdf) — paper by Marc de Kruijf and Karthikeyan Sankaralingam; from University of Wisconsin-Madison
- "Mars: A MapReduce Framework on Graphics Processors" (http://www.cse.ust.hk/catalac/users/saven/GPGPU/MapReduce/PACT08/171.pdf) — paper by Bingsheng He, Wenbin Fang, Qiong Luo, Naga K. Govindaraju, Tuyong Wang; from Hong Kong University of Science and Technology; published in Proc. PACT 2008. It presents the design and implementation of MapReduce on graphics processors.

- "Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters" (http://portal.acm.org/citation.cfm?doid=1247480.1247602) — paper by Hung-Chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D. Stott Parker; from Yahoo and UCLA; published in Proc. of ACM SIGMOD, pp. 1029--1040, 2007. (This paper shows how to extend MapReduce for relational data processing.)
- FLuX: the Fault-tolerant (http://citeseer.ist.psu.edu/647742.html) , Load Balancing (http://citeseer.ist.psu.edu/546646.html) eXchange operator from UC Berkeley provides an integration of partitioned parallelism with process pairs. This results in a more pipelined approach than Google's MapReduce with instantaneous failover, but with additional implementation cost.

Articles

- "How Google Works - Reducing Complexity" (http://www.baselinemag.com/article2/0,1540,1985048,00.asp) — article from Baseline magazine
- "Can Your Programming Language Do This?" (http://www.joelonsoftware.com/items/2006/08/01.html) — article from the *Joel on Software* weblog
- Nutch MapReduce (http://weblogs.java.net/blog/tomwhite/archive/2005/09/mapreduce.html) — article about MapReduce in Nutch from Tom White's weblog
- Cat MapReduce (http://code.google.com/p/cat-language/wiki/MapReduce) — article about MapReduce in Cat from the Cat project wiki.
- "Simple Map Reduce in Ruby" (http://multipart-mixed.com/software/simple_mapreduce_in_ruby.html) - article about using SimpleMapReduce on Ruby's Rinda which uses DrbRuby
- "MapReduce: A major step backwards" (http://www.databasecolumn.com/2008/01/mapreduce-a-major-step-back.html) - column about advances in database technology compared to MapReduce.
- Relational Database Experts Jump The MapReduce Shark (http://typicalprogrammer.com/?p=16) - addresses a misconception in the comparison at the previous article.

Software

- Hadoop — open source MapReduce implementation from Apache
- IBM MapReduce Tools for Eclipse (http://www.alphaworks.ibm.com/tech/mapreducetools) — a plug-in that supports the creation of MapReduce applications within Eclipse.
- QtConcurrent (http://labs.trolltech.com/page/Projects/Threads/QtConcurrent) Open Source C++ MapReduce (non-distributed) implementation from Trolltech
- Skynet (http://skynet.rubyforge.org) Ruby Map/Reduce Framework