MapReduce

MapReduce: Simplified Data Processing on Large Clusters

Summary by: Daniel Blossom

November 25th, 2013

Main Idea

- Allow programmers to easily utilize resources on large distributed systems.
- Input for large computations is not always straight forward; to compute the input data becomes overly complex.
- MapReduce simplifies the process by creating an interface that will allow parallelization and distribution over these large computations.
- Inspired by Lisp two functions *map* & *reduce* are implemented; map the operations and reduce values with same key.
- Both functions are written by the user.

Implementation

- Map is a key value pair and Reduce is a key value pair from resulting list; which outputs a list. An example from the paper:
  - Map (key1, value1) → list(key2, value2)
  - Reduce (key2, list(value2) → list(value2)
- User implements map, reduce and mapreduce functions. User invokes MapReduce.
- Program splits input files into *n* pieces. Those pieces are distributed over the cluster of PC's.
- A special copy “master” is created to control the tasks (distribute the work).
- Workers divide up the map and reduce tasks directed by the master.
- Some map pairs are written to the disk in regions. The master will direct the reduce function where these regions are located. Others are written directly to memory.
- Reduce workers retrieve the intermediate data created by the map workers and looks for unique keys. The reduce function appends its finding to the output file or files.

Analysis

- Process of breaking a large problem into smaller parts and then processing those smaller parts. This divide and conquer technique will drastically increase performance in processing large datasets.
- Master needs to keep all the data structure for each map; seems like a lot of storage use.
- Having user define the implementation provides more work on the programmer but allows for more custom-ability to their application.
- Having the master send the map and reduce functions to workers acts like threading, and can assist in graceful failures. The master wil assign the task to another node in the cluster.
- In the papers test environment not one particular machine stood out as overly powerful; which implies a powerful server(s) is not needed.
- To alleviate the issue of stragglers (caused by machines with some sort of processing issue at run-time) a backup task is created, seems like more processing and storage than might be needed. However if it increases the average paying upfront might be a better deal, the authors seem to think it does. In fact he shows us a 44% increase with backup disabled.
- Encapsulation makes implementation “easier.”
- Seems to operate like a workload manager in any operating system but with less fault tolerance. I say less because I cannot get past the single master.

Advantages & Disadvantage

- With one “master” managing the work, makes for a single point of failure, even with its checkpoint state.
- Clustering the work will increase performance.
- Testing library is available, easier to test/debug on one or few machine than thousands.
- Since it sorts the intermediate keys it is guaranteed in order.
- A lot of overhead to implement
- Even with that overhead it does provide a sense of encapsulation to the programmer.
- Not useful for everyday database needs, but could provide useful in processing information pulled from an everyday database.
- Has assisted google in redefining their own indexing, so maybe the same can be done somewhere else.
- Limited network bandwidth needed.
- Tasks stored on local disks; space will always need to be available.

Real World Cases

- Data mining (website specific but idea is larger than this)
  - How many visitors to a website (We all want to know that our site does not suck)
  - What type of queries was made on a website (IE: Product on Amazon).
  - Location of visitors to a particular website (IE: IP suggests more NY than CA visitors, why? Is this a local business?)
- Data sorting
  - Large datasets of millions of records can be sorted very fast.
  - Population of the United States comes in a form of one large unsorted csv file. Breaking this up into smaller individual tasks for cluster computing would take this very slow sorting task and speed it up. Could mean the difference of hundreds of seconds (or translation – hours).
- Queuing mechanisms could benefit from this idea.
  - A group of 20 friends head to Six Flags and all stick together, it would take all day to ride every ride. But if those 20 friends spread out and each rode a few rides (kiddie ones included) they could meet up at the front entrance faster and compare notes about the ride (of course nothing beats riding a roller coaster yourself).
  - Ever head to the buffet at a resort with a group of friends at lunch time? Very limited tables. Usually you travel looking in a group, however a divide and conquer method would work better.
- Financing: With the upcoming holiday season the CEO of Wal-Mart needs to know the sales trends for the past 5 years during the months of November – January and compare those numbers the other months. This information will help upper management strategically layout every store so that maximum sales are utilized, more of some products are ordered for some stores but not others, and where some items should be located. With all the stores and each store having different items sold more in some states these types of queries could take days to compute. With utilizing the power of map / reduce and clustering the work over thousands of PC's, what seems like an impossible task is very doable.