ACCELERATING MAPREDUCE WITH DISTRIBUTED CACHE

MapReduce is a partition-based parallel programming model and framework enabling easy development of scalable parallel programs on clusters of commodity machines. In order to make time-intensive applications benefit from MapReduce on small scale clusters, this paper proposes a new method to improve the performance of MapReduce by using distributed memory cache as a high speed access between map tasks and reduce tasks. Map outputs sent to the distributed memory cache can be gotten by reduce tasks as soon as possible. Experiment results show that our prototype's performance is much better than that of the original on small scale clusters

Hadoop runs several map and reduce tasks concurrently on every node. Each node tells the Job Tracker when it has empty task slots. The scheduler then assigns it tasks. To help task scheduling and monitoring, Hadoop describes task progress using a progress score between 0 and 1. For a map task, the progress score is the fraction of input data processed. Reduce task is divided into three phases, each accounting for 1/3 of the score:

- The shuffle phase, when the task fetches map outputs.
- The sort phase, when map outputs are sorted and merged by key.
- The reduce phase, when a user-defined reduce function is applied to the list of map outputs with each key.

In each phase, the score equals to the fraction of data processed.

Primary goals of this prototype are:

- Stay close to the original
- For small scale clusters
- Retain fault tolerance
- Local decision making

Distributed memory cache is a sensible solution to meeting these requirements

- no central coordinator
- a uniform global namespace
- low-latency, high-throughput data access
- the ability to deal with concurrent access
- large storage capability
- scalable architecture.

We use memcached to provide low-latency, high throughput access to map outputs in Hadoop. Use of memcached proceeds in two steps

- First, if there are idle slots in memcached, some output of a map task is loaded into memcached as a pair. The key is made up of the map task's ID and its target reduce task's ID. And the output is set as value.
- Once a reduce task is spawned by a TaskTracker, it gets all the finished map tasks' information from the TaskTracker. Then it checks whether the outputs for it are in memcached. If the answer is yes, it gets them from the memcached servers. Then it shuffles the remaining from the TaskTrackers on which the map tasks ran.