Implementation of Task Scheduling algorithm with Deadline Constraints for MapReduce

Abhijith C S – CS11B003

Introduction

These days the requirements for massive data processing are increasing drastically. MapReduce is a distributed programming model for expressing distributed computation on massive amounts of data and an execution framework for large-scale data processing on clusters of commodity servers. The most important advantages of MapReduce is its convenience. Programmer can process massive data without knowing the details of distributed implementation. User can process large scale of data by only providing the Map and Reduce interface.

It is an attempt to implement a task scheduling algorithm with deadline constraints in Hadoop platform. It considers the specified deadlines of a job and tries to make the job be finished before the deadline. The standard implementation could take the node's computing capacity into account for improving the performance – some algorithm classifies the nodes into several levels in heterogeneous clusters. Later part is left as a future scope for research.

Implementation

There are many task scheduling algoriths for Hadoop and MapReduce in particular. The default one being the FIFO implementation – the first come job is served first and next job next. Capacity Scheduler, Fair Scheduler etc., are some other task scheduling algorithmms we have by default. Another algorithm being LATE - Longest Approximate Time to End, as name suggests it considers the approximate time to end for a job and schedules the tasks.

In several practical cases, one might want to restrict some jobs based on the deadlines. A job must have to be executed in certain amount of time, in order to get the desired output. So it is significant to come up with the idea of a task scheduling algorithm which considers the deadline of the task as well into account. FIFO (First In First Out) is the default scheduling algorithm in Hadoop – the job coming in First will be executed First and the next job next in the order. In such a case, say if a job wanted to get finished before some other job in advance in order to get the desired output, it is impossible. Here comes the idea of incoperating another parameter called Deadline, which is basically the worst maximum time under which the task must be finished.

In a way, it is equivalent to a Priority Queue based implementaion and allocation of jobs. In the default scheduler, each job is getting queued in the order they're coming in. The job from the front is pop'ed out and getting executed. Some slots are kept vacant for faulty tasks (for ensuring fault tolerence) and speculative tasks. Here in the new mode of scheduling, we define a parameter called 'Deadline' associated with each job. For the sake of implementation, deadline is considered as a random integer value associated with each job. Smaller the value of the deadline, shorter is the 'life' of the job and it has to be executed at the earliest – ie., it gets the highest priority in the queue. When a new job comes in, its associated dealine parameter is compared with that of other jobs in the queue and put in the jobQueue accordingly. Most prior tasks come in the front of the queue.

Note that, in the implementation of the algorithm an additional parameter 'int deadLineValue' is defined. Value of which is set during run time, just as a random number. Smaller the value, smaller is the time for it to get executed, thus most priority. This parameter is included into the default FIFO scheduling, converting the queue into a priority queue. When a job with more priority comes, put into the front pushing all others back (maintaining a normal standard priority queue).

Observations – The algorithm was run on virtual set up (3 virtual machines running on two Ubuntu Machines) using Oracle VirtualBox. I could observe that, when there were no clashes in the jobs (drawback mentioned below), it ran succssfully as expected.

Drawback – Say the case you have all the nodes running and no empty slots left (the intentionally kept free slots for fault tolerance are also busy). Suddenly a new job enters with a demand of immediate execution – ie., smaller value of 'deadline' parameter, the algorithm neglects it. Which is a major drawback of the model. There has to be some way, the more prior job coming in getting executed on top of others without affecting the flow.

Improvements – In the sample implementation of the algorithm, deadline parameter is set just as a random number during run time. Have to find a way to assign deadline values before hand. Also, some modification to make the algorithm more effecient by avoiding thee drawbacks.

Conclusion

It's just a simple way of including 'deadline' parameter in task scheduling algorithms. There are many other factors to be considered like, the computing power of each node etc. The common assumptions we take in the case of Hadoop Cluster are -

- Nodes can perform work at roughly the same rate.
- Tasks progress at a constant rate throughout time.
- There is no cost to launching a speculative task on a node that would otherwise have an idle slot.
- A task's progress score is representative of fraction of its total work that it has done. Specifically, in a reduce task, the copy, sort and reduce phases each take about 1/3 of the total time.

- Tasks tend to finish in waves.
- Tasks in the same category require roughly the same amount of work.

The above said assumptions are to be analysed and rather than incoperating the deadline just as a parameter of time, computing power of node, progress rate etc., are also to be counted to come up with an effecient scheduling algorithm. This project doesn't cover them all. It suggests huge scope for research in thhe field.

Reference

1. Paper on *MTSD* by Zhuo Tang Sch. of Inf. Sci. & Eng., Hunan Univ., Changsha, China
2. Paper on *Job Scheduling for Multi-User MapReduce Clusters* by M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker and I. Stoica, “,” Technical Report of University of California, Berkeley.
3. http://hadoop.apache.org/mapreduce/docs/r0.21.0/capacity_scheduler.html, 2011.