MLFQS for Big Data Scheduling

Eric Feldman

Abstract

In today's world, many companies rely on big data processing to provide their services. Increasing the speed at which these services are provided can directly lead to profits. An important part of big data processing systems is scheduling. Currently these systems use simple schedulers such as first in first out scheduling, which can have job starvation problems. The optimal scheduler is known as shortest job first; however, it requires that you know the length of the jobs a priori. This is a difficult task that can not be done in reality. This paper presents the idea of using a multi-level feedback queue scheduler, a common scheduler used within operating systems, to estimate shortest job first. A simulation replicating key parts of a big data environment was created. The simulation uses all three schedulers and compares their average task throughput.

1. Introduction

The system that is replicated in this paper is the Hadoop system [1], which is the open source implementation of Google's MapReduce [2]. MapReduce is a programming model for processing big data, in which users implement a *map* and *reduce* function. The map function takes in a key/value pair and generates a set of intermediate key/value pairs. The reduce function merges the intermediate values with the same key. The Hadoop system will parallelize and execute these functions on a large cluster of commodity machines automatically and without any input from the user. The system is responsible for many things, but this paper will focus on the scheduling of tasks within the system.

To understand how scheduling works within the Hadoop system, it is important to understand the execution of a Hadoop job.

Execution Model:

1. First, the input files are split into M pieces, and multiple copies of the program are copied onto a cluster of machines.

2. One of the machines is chosen as the master and assigns work to the rest of the machines called workers. The master assigns each worker map and reduce tasks.

3. Once the system is initialized, the map phase starts. Workers are assigned map tasks in which they read key/value pairs from the input data pieces. The workers run the user defined map function, which produces intermediate key/value pairs. These intermediate values are buffered in memory and periodically written to disk. Once written to disk, the worker will inform the master that the job is completed and where the intermediate key/value pairs are stored on the worker's local disk.

4. Once all map workers are finished, the master will inform the reduce workers about the locations of the intermediate key/value pairs. The reduce workers will acquire and read the data and sort the intermediate keys such that all occurrences of the same key are grouped together.

5. The reduce workers then run the user defined reduce function on the sorted intermediate key/value pairs and outputs user defined data. Once all reduce tasks have completed, the master wakes up the user program and returns.

The key take-away from this is that the map and reduce phases happen in batches. No reduce task can start running until all of the map tasks have been completed. If a reduce task were to start running before all map tasks were completed, it is possible that the reduce tasks will not have all of the intermediate key/value pairs necessary and will therefore report incorrect results. One of the problems with batch like programming model which is mentioned in the MapReduce paper is that sometimes one of the workers in the cluster will be significantly slower than the rest. Even though most of the tasks within the batch have finished, the next phase cannot start. These machines are labelled stragglers. There are many reasons for a machine to straggle behind, but the one mentioned in the MapReduce paper is that a machine may have a faulty or slow disk, and since there is a lot reading and writing to disk, this task takes much longer than the rest.

Scheduling:

Within the Hadoop system, the master node is responsible for keeping track of and assigning tasks to workers on the cluster. Creating a scheduler algorithm that makes decreases the total length of a job is an important task. Doing so can decrease the amount of time it takes for a company to provide their service. The default Hadoop scheduler (FIFO), the optimal scheduler (SJF), and the proposed scheduler (MLFQS) will now be discussed.

FIFO:

The default scheduler the Hadoop system uses is a first in first out (FIFO) scheduler. This is a simple scheduler that runs tasks on a first come first serve basis. When tasks enter the system, they are put at the end of a queue. When a new task is ready to run, the master will pull from the queue and run that task to completion. This is a simple scheduler and it is easy to implement, but it has its drawbacks. If longer tasks are scheduled first, it will delay all of the tasks behind it from starting. This will make the execution time of the batch much longer.

SJF:

The provably optimal scheduler in terms of throughput is the shortest job first (SJF) scheduler. This scheduler chooses the task with the shortest length to run next. This eliminates the problem mentioned with the FIFO scheduler. Longer tasks cannot prevent other tasks from starting. While this scheduler is the optimal scheduler in terms of throughput, it assumes that the system will know the lengths of the tasks prior to running the them. Of course, in reality, this information is not known and is difficult to predict.

MLFQS:

The proposed scheduler to fix this problem of unknown job length is the multi-level feedback queue scheduler (MLFQS). It was created for operating systems to try and estimate the SJF scheduler. However, before explaining how a multi-level feedback queue scheduler works, it is important to understand how a round robin scheduler works. A round robin scheduler will pull from the front of the queue and allow the task to run for a given time slice. When the time slice is over, the scheduler will stop running the task, reschedule to the end of the queue, and then pull the next task. This allows for all jobs within the system to run and prevents any task from starving the other.

The MLFQS scheduler dynamically characterizes tasks into long and short jobs based on the amount of time the task is running on the CPU. Within an operating system, the longer jobs are those that are CPU bound, and the shorter jobs are those that are I/O bound. The scheduler makes use of multiple queues,

and each queue has a priority attached to it. The scheduler uses a round robin scheduler within the highest non-empty priority queue. The scheduler then makes decisions based off of the following rules:

1. When a new job enters the system, the job is placed at the end of the highest non-empty queue.
2. When the current job is running, three different decisions can be made.
    a. If the job finishes within the time slice, it leaves the system.
    b. If the job relinquishes control of the CPU before the time slice completes, it is rescheduled to the end of its current queue.
    c. If the job runs for the entire time slice, it is rescheduled in the queue one priority level down.

This scheduler is designed to push longer CPU bound jobs into the lower queues and run the shorter I/O bound jobs first. The Hadoop system would like to do the opposite. Ideally, the stragglers in the system run last, and do not prevent the other jobs from running. Therefore, the proposed scheduler will switch choices 2a. and 2b. If the job relinquishes control of the CPU before the time slice completes, it is rescheduled in the queue one priority level down. If the job runs for the entire time slice, it is rescheduled at the end of its current queue. This will keep with shorter CPU bound jobs at the higher priorities and the longer I/O bound jobs in the lower priorities.

2. Related Work

Work has been done to compare different schedulers for MapReduce jobs [3]. This work mainly involves comparing the current schedulers that Hadoop has to offer. These schedulers are broken down into two types, static and dynamic. Static schedulers are those that schedule the job to the processor before the job begins to run, while dynamic schedulers are those that schedule the job during run time. Dynamic schedulers allow for preemption within the system. The Hadoop system supports three kinds of schedulers. A FIFO scheduler, a fair scheduler, and a capacity scheduler. The fair and capacity scheduler are both dynamic schedulers that allow for preemption. They were both designed to account for multiple users using the same cluster. When using the fair scheduler or the capacity scheduler, the system will give equal amounts of resources to every user on the cluster. If there is an imbalance in resources, the scheduler will preemptively redistribute the resources. While this scheduler is resource aware, it does not try to solve the same problems as the MLFQS does. Each user will still be using FIFO scheduling within their allocation of the cluster. This means that longer tasks within one user's job can still starve that user's shorter jobs.

In a paper about the different types of schedulers proposed for the MapReduce model [4], the idea of a resource aware scheduler is mentioned. This scheduler is used to solve a similar problem as the MLFQS scheduler. A resource aware scheduler is better able to dynamically schedule jobs based off resource consumption. This scheduler uses two resource metrics, "free slot filtering" and "dynamic free slot advertisement" to schedule the jobs. Each node will advertise their resource availability, and the scheduler assigns tasks to computational slots on each node. While this scheduler will improve the resource utilization in the cluster, support will be needed to manage the network bottlenecks that come with advertisements.

3. Design

To prove that a MLFQS would improve the throughput of MapReduce jobs, a simple simulation of the Hadoop system was created in Java. For this simulation, the three schedulers mentioned were implemented (FIFO, SJF, MLFQS). All three schedulers will be compared with various workloads. For this simulation a *task* is the individual map or reduce task, and a *job* is made up of tasks, which simulates the entire map or reduce phase.

To simulate the MapReduce execution model, jobs are created and run in batches. Meaning, all tasks from a certain job must be completed before the next job can start running. Each task represents either a map or reduce task. These are fake tasks that do not actually do any work. Rather, each task has a predetermined arrival time and length. When the system decides it is time for the task to be run, the task will sit in a while loop for the length of the job. When finished, the task will record its throughput so that the system can take measurements of each scheduler. In order to mimic the stragglers in the Hadoop system, the simulation creates many short jobs and a couple of longer jobs. For the MLFQS scheduler, the tasks will relinquish control of the CPU according to a certain probability, depending on the length of the job. This simulation does not run tasks in parallel like the Hadoop system would. However, this is not a problem, since the principles behind the schedulers still holds regardless of whether one task or an arbitrary number of tasks are running at the same time.

Another idea was considered to test whether a MLFQS scheduler would improve the throughput of MapReduce jobs. Hadoop is open source, and therefore it is possible to add the scheduler to the Hadoop source code. However, only one person worked on this project and it was decided that this would be too much work for one person to finish in a reasonable amount of time.

4. Evaluation

The throughput of the different schedulers was measured and recorded. The throughput of each job was measured as the time of completion minus the time of arrival of each job. The average throughput of each task was recorded for each scheduler. With this definition of throughput, a lower throughput is better. It is a measure of how long the task had to wait on the tasks before it. The results of a smaller workload, with 2 jobs and 10 tasks each can be found in figure 1. The results of a larger workload, with 2 jobs and 100 tasks each can be found in figure 2.

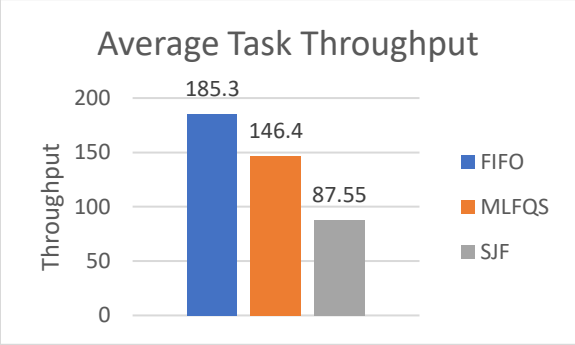

Figure 1. 2 jobs, 10 tasks per job

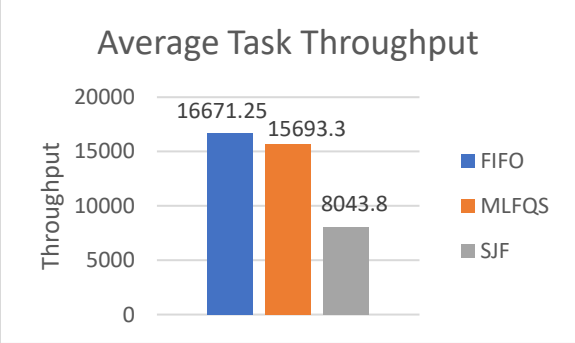

Figure 2. 2 jobs, 100 tasks per job

From the figures, it is clear that the MLFQS scheduler is an improvement on the FIFO scheduler. This is a promising first step in proving that the MLFQS scheduler will make

MapReduce jobs faster. The next step is to take this idea and use it in a real system. As mentioned earlier, the main bottleneck for implementing this in a real system will be network bandwidth. If this is too much of an issue, perhaps modifications can be made to this scheduler such that the main principles still hold. Instead of sending feedback after every time slice, feedback can be sent less often. This will still allow the main principles of the scheduler to hold, without needing to send data very often. Or perhaps tasks can be clumped together on a single machine, and the MLFQS scheduler can be run amongst those tasks. This will avoid the need to send data across the network.

5. Conclusion

The multi-level feedback queue scheduler has worked for operating systems, and it has been shown that it is a viable scheduler for big data processing as well. The next steps include implementing this into the Hadoop system and deploying it on a cluster. The one drawback to doing so may be the network bottlenecks that dynamic resource aware schedulers have. Special care may need to be taken to make sure that the messages passed between the workers and master do not create too much congestion in the network. If network bottlenecks are too big of problem, then versions of the MLFQS may be used to still implement the key principles behind the scheduler.

References

[1] Corparation, A. S. (n.d.). *Apache Hadoop*. Retrieved from https://hadoop.apache.org/

[2] Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*.

[3] team, D. (2018, Semtember 15). *Hadoop Schedulers Tutorial - Job Scheduling in Hadoop*. Retrieved from https://data-flair.training/blogs/hadoop-schedulers/

[4] Usama, M., Liu, M., & Chen, M. (2017). Job schedulers for Big data processing in Hadoop environment: testing real-life schedulers using benchmark programs. *Digital Communications and Netwroks*.