Paper


Summary

MapReduce (MR) is a programming model designed for parallel processing of large datasets. It provides a simple interface to the user that processes large datasets in parallel in a distributed system. The key benefit that promoted its widespread is that it significantly reduces the processing time of the continuously growing large datasets, while hiding all the details from the user. The drop in computer prices, the spread of publically available clouds, and the availability of open-source implementations, definitely pushed it spread even further. Hadoop, an open-source MR implementation, is widely deployed in the clouds. However, Hadoop assumes that all the nodes are homogeneous while in fact they are heterogeneous; the heterogeneity could result from a cloud that is composed from different machines having different capabilities or from virtualized machines competing on these physical machines. This heterogeneity could degrade the performance of MR in the clouds. So the authors of this paper proposed a modification to Hadoop that can improve MR performance in the clouds. The authors are trying to solve some inefficiencies of Hadoop’s speculative task scheduling: (i) it selects the tasks that have the lowest progress for speculation, and (ii) when scheduling a speculative task, it assumes that they are all equal. These design choices were based on the assumption that all nodes are equal while this assumption does not hold in the heterogeneous environment of the clouds. Progress alone is not an enough indication of speculative tasks; nodes have different progress rates and it is incorrect to prefer new but fast tasks over old and slow tasks. So the solution presented in this paper is basically ranking the tasks based on their estimated remaining time, and then scheduling the tasks on fast nodes. The intuition behind using estimated remaining time is that the scheduler should prefer tasks that are expected to end further in the future. They also used the task’s progress rate over its lifetime in order to estimate the remaining time. In addition, to reduce resources costs, they used a cap to limit the number of speculative tasks running at a time, and a threshold to prevent speculating fast tasks. They called their algorithm: Longest Approximate Time to End (LATE). According to their statement, LATE can reduce Hadoop’s response time by a factor of 2.

Critique

The authors did a good job in explaining the assumptions that underlie Hadoop’s design choices, and how these assumptions don’t hold when it is deployed in the heterogeneous environment of the clouds, causing performance degradation. However, they used one of these assumptions in their algorithm. The estimated remaining time for a task was based on its progress and progress rate. They assumed constant progress rate per task, which may not hold. Using estimated remaining times is their main idea in the paper, yet they used one of the main invalidated assumptions. This caused some incorrect estimated times, but they claimed that it is infrequent in typical MR jobs, thus not impacting its performance. They proposed a solution regarding estimating the remaining time for reduce tasks. Their solution was to use a per-phase progress rate to estimate the remaining time of these tasks. This may solve part of the problem where the reduce tasks have different time requirements per phase by their nature. But they again assume constant progress rate within each phase and that they are relatively similar between
the different nodes. In addition, although the authors were against Hadoop’s progress scores because it divides the score equally between the three phases of the reduce tasks, they used it in their estimations.

As a possible extension, I suggest two fixes that may (or may not; needs further development and testing) improve the performance. First, to improve the progress score calculation, I suggest using the user’s knowledge about his/her own developed map and reduce functions. When the user submits these codes through the MR interface, the user can also include an estimated time complexity per record read. Then MR uses this information to proportionally divide the scores for the different tasks. Second, I suggest dropping the usage of the progress rate in the ranking procedure. I think that the ranking value should be proportional to the amount of progress remaining \((1 - \text{progress score})\) and how long have the task been running \((T)\). Thus the ranking value could be \((T.(1-\text{progressScore}))\). So the older they are and the less progress they made, the more preferable they are for speculative execution. It is a simpler heuristic but may work.

For their evaluation, they did the experiments on different multiple environment settings. I’m not sure if that is suitable. But the scale was a little small in three out of four environments (shown in Table 1). I think it should have been larger. In the scheduling experiments in Section 5.2, they performed 5-7 runs per experiment and the results had high variance. I think that the number of runs is small and they should have increased it to compensate for the high variance. A possible reason for limiting the scale and number of runs is the cost. I don’t think it is wrong to control some testing environments to see how the algorithm performs in some settings, but I’m not sure if it is too controlled. The performance results comparing LATE, Hadoop, and Hadoop with disabled speculation, using three different jobs are not the same and are not relatively proportional. So how can they generalize their performance result and claim that LATE reduced Hadoop’s response time by a factor of 2? I also think that they should have included a comparison of what they consider typical MR jobs and non-typical MR Jobs and look at performance differences.