THE CHINESE UNIVERSITY OF HONG KONG

CSCI5570 Lab Report

Jiang Jiaxin
Liang Huichao
Liu Jianlin
March 27, 2020

1 BASELINE

We have implemented a Parameter System with the following function:

1. Configurable worker number in each node
2. GET, ADD interfaces for global parameters
3. BSP, ASP and SSP consistency model

To provide an performance evaluation of our system, we have done two experiments.

1. Train model with fixed worker number but different data sizes. Result can be found in Figure 1.1
2. Train model with fixed data size but different consistency models and worker numbers. Result can be found in Figure 1.2

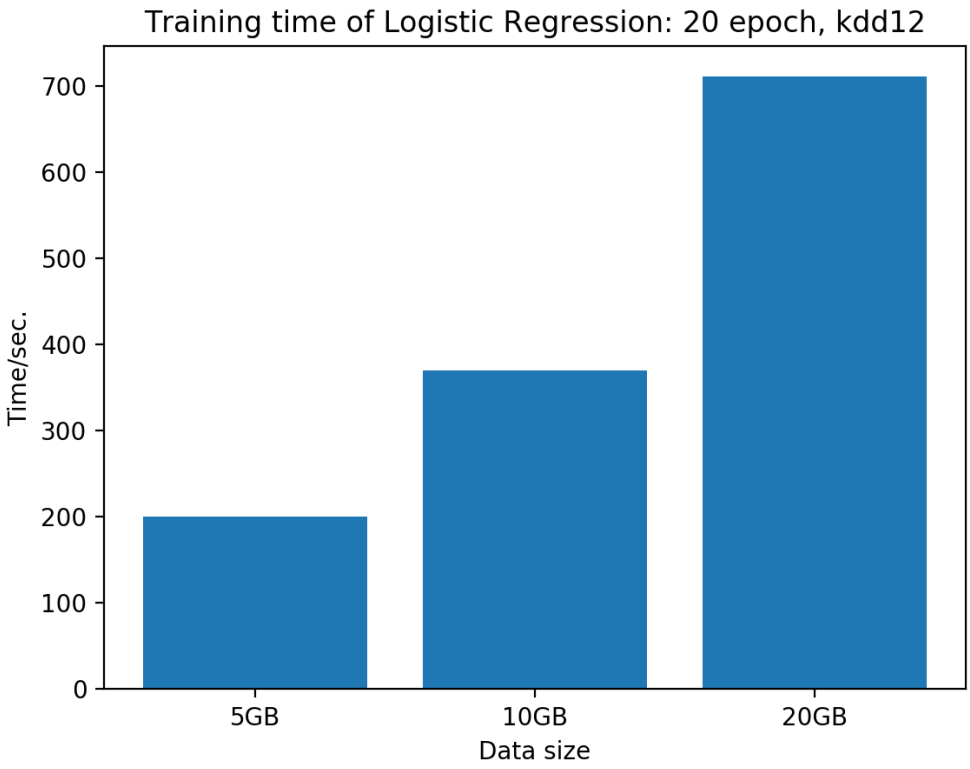

Figure 1.1: Training time with different data size. 120 threads

Figure 1.1 shows the training time of Logistic Regression on different data size. The growth of training time can be represented by a linear trend with the growth of data size on kdd12. Also, it demonstrates that our system could process data at this scale within a reasonable time.

Figure 1.2: Training time with different consistency model and worker numbers

As we can see from the Fig.1.2, the longest time is used to train models on BSP model. SSP model takes shorter time and ASP model is the fastest among three consistency models. Furthermore, training time reduces as the number of worker increases, which fits the feature and goal of our parameter system. Being different from the result of data size, time used on different consistency models is not linear with the number of worker.

You can see the number of worker is fixed in our experiment. Fortunately, the performance can be better with more worker. Our system could process large scale data by increasing the number of worker. The training time will not be very long with enough worker.

2 Additional Feature

Under Bulk Synchronous Parallel (BSP) computational model, all workers have to wait until the slowest worker finishes the computation in each iteration. The straggler problem will affect the performance of the overall system, especially when the number of the workers increases. In such situation, the computation resource will be wasted for most of the workers are idle.

To address the straggler problem, we use a Work Assigner to reschedule the workload among workers(training data) during each iteration. To achieve this goal, we assigned each node its

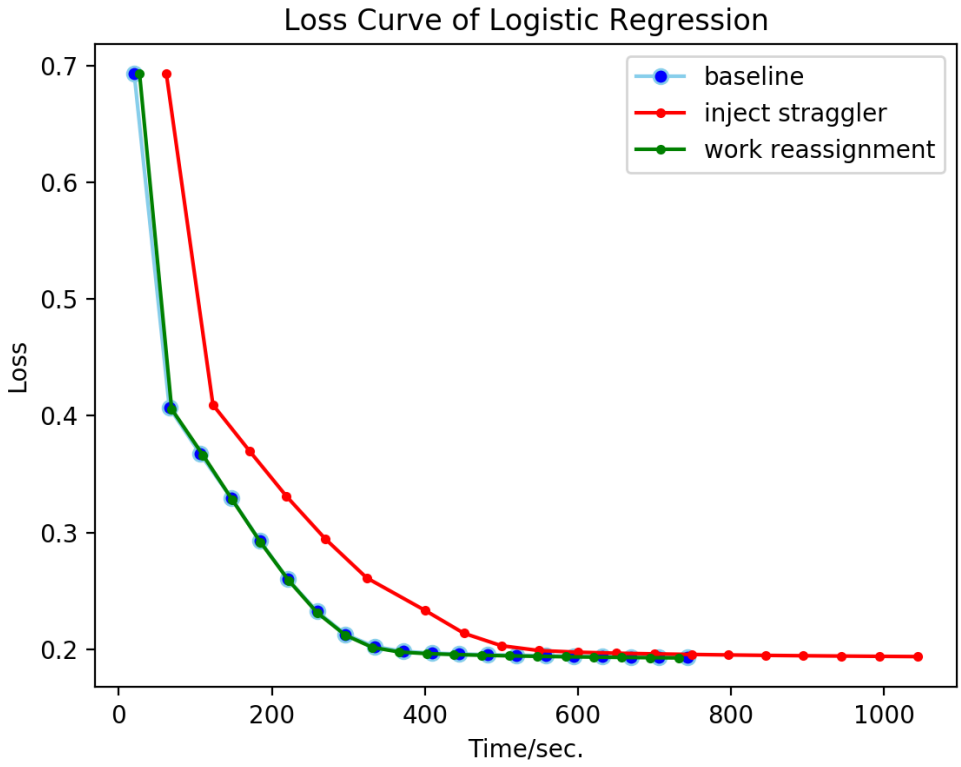

Figure 2.1: The blue line: no straggler, work reassignment disabled.
The red line: injected straggler, work reassignment disabled.
The green line: injected straggler, work reassignment enabled.

next sibling as helpee, so that the helping relation is determined. Notice that we will need to transfer a portion of the training data from straggler thread to the helper thread. To avoid massive data transport between workers, we preload a backup data partition of helpee node in every node. When a straggler worker is detected, its backup node can start work steeling if possible.

To validate the effectiveness of our method, we inject some stragglers by randomly putting threads to sleep and compare the training time with/without work reassignment. We trained a Logistic Regression model on KDD12 dataset. The result is shown in Figure 2.1. As we expected, the overall training time with work reassignment strategy is much lower compare to the opposite, which indicated that the straggler issue had been alleviated.