Lab 3 Write Up - Failures

by Zeyue Chen and Jiheng Lu

https://github.com/zeyuec/parallel-computing/tree/master/lab3

1. Solution Overview

Overall, in order to handle the case of failure in Master Node, the system will initialize a Backup Master Node, in order to keep the track of the status of Master. Before delivering jobs to Workers, Master Node will divide the overall job into pieces, store the data required into its own memory, and store the pointers directing to the data into a stack of ids (jobStack) and the pointers directing to the available workers(workerStack). These part of memory will be sent to Backup Master and refreshed over time, including the workerStack and jobStack.

Whenever Master Node receives results from Workers, the partial results will be stored into Collected Result, which will also be refreshed with Backup Master by sending out the part to be updated. Master Node will also put this worker back to the workerStack to make sure it can be used for the future computation.

For the Worker Nodes, they will receive part of data with the job requiring them to complete from Master Node, and they will send back the result to whomever send them the job. Therefore the logic here is simple.

In order to handle failures in worker nodes, the system is designed to keep track of status of all the Worker Nodes (workerStatus), works completed (finishedCount), a stack of work to be completed (jobStack) and a stack of available worker can be used for computation. These information will be kept on the Master Node, which will also be kept refreshed with Backup Master all the time.

2. Part 1: Failures in Worker Nodes

As mentioned in the Overview, in order to handle the case of failures in Worker Nodes, there is a Worker status tracker, called workerStatus, to track all the status of the Worker (whether it is alive), current job id, and last response time.

The Master Node will keep waiting for the response by the Worker Node after sending out the job. After a pre-assigned time, if there is still no response from the Worker, the Worker Node will be treated as dead. This timeout period is currently assigned as 3 seconds, because the averaging task-completion time for this lab is around 1.2-1.5 seconds. This time will need to be assigned prior to the MPI initialization.

After determining a Worker to be dead, the Master will set the status of the Worker in workerStatus to be unavailable, so that Master will not send job to it any more. In addition, the current job the Work was working on would be push back to the jobStack, and assigned to any other available Worker Node. Anything else remains the same.

3. Part 2: Failures in Master Node

In order to handle the failures in Master Node, a Backup Master Node is introduced, and it will be kept refreshed all the time. In other words, the Backup will be synchronized to be the same.

The Backup does not perform too much calculation, but rather it maintains all the data Master Node may have. There is also a timeout period for the Backup Master Node to check the last updating request time from Master Node. For this lab, the period is assigned to be 1 second. As the worker timeout period, it is required to be assigned prior to the MPI initialization.

When the Master Node is actually dead, the Backup will no longer receive any updating request. After the timeout period, the Backup will treat the Master to be dead, and pick up the work from the status of it lastly received from Master. In other words, rolling back to the last time Backup Master Node is refreshed to the same as Master Node, the Backup Master starts to deliver the jobs in the jobStack, and treat itself as the new destination of the completed results.

4. Simple Questions

a. How does the performance of the system scale with the value of the threshold p?

The value of threshold p determines whether the task can be completed significantly, as described in the next section (b). If the task can be completed, for the performance, the p does affect a bit (the more Workers are dead, the slower the completion time would be).

However, the affection is not clear, because if p is smaller than the boundary (making each Worker has close to 50% of the possibility to complete all the jobs), it is more likely most of the Nodes are not dead, and if p is larger than that, almost all the Nodes will be dead before the completion. And, the contingency of how many Nodes will be working and the overall completion time is way too high. Therefore, it is very hard to make a clear statement for the relation between the performance and p value.

b. **What values of p result in a solution that would be reasonable to deploy in practice?**
   For 1000 overall pieces of jobs, the p is reasonable to be assigned as 0.005, i.e. 0.5% of the possibility the Worker Node might be dead/failure occurring. After multiple times of testing, this p value is reasonable to have all the jobs completed.

   We figured out that for each node the possibility of the node to complete all the assigned jobs is:

   P = (1 - p) ^ (n / m) n - number of overall jobs
   m - number of Worker Nodes

   It is found that when P is smaller than 50%, e.g. half possibility to complete all the nodes, the task can be completed. When one Worker is down/dead, the jobs on other Workers will be increased (the jobs initially assigned to the dead Worker will be assigned to other Workers), and therefore the possibility of other Workers to be dead is also increased significantly. If the P is larger than 50%, which means, half of the Nodes will be dead before it completes all of its jobs, then the rest of the Nodes will have much more than 50% possibility to be dead before the completion. The more Nodes dead, the less the rest of the Workers are possible to complete all the jobs.

   Again, for 1000 overall jobs with 8 cores initialized (i.e., 6 Worker Nodes), each Worker has 1000/6 ~= 167 jobs, with P = (1-p)^167 = 0.5. As a result, we get 0.5% as the reasonable number for a p.
   If there are 317 overall jobs(1e15 as the input of factor problem), the reasonable number for a p should be smaller than 1.4%.

c. **Augment your master-worker solution to handle a single failure in the master.**
   The single failure in the Master Node is handled, and the solution is described above in 3 (the Part 2).

5. How to compile the code

Since we're using the thread features in c++11 standard. The code should be compiled with g++ compiler that supports the new standard. If you want to compile that on CentOS, please make sure the g++'s version is above 4.4.