Rui Huang rhhq7

For homework four, the task was to accelerate the process of building the inverted files using MPI. Firstly, I use master to merge all files to an in-memory data structure called book-type. Then all workers began to work. Master received the result from workers and merged the result to final result. The following part of this report will discuss the details about my implementation, the result and also the analysis of the result.

1. Implementation

   1.1 Read the directory and build a file name list

   In order to build the inverted file, the first thing needed to be done was knowing how many files there were and the name of each file. I use the data structure in Boost library: list<boost::filesystem::path>. I went through the directory, each time I met a file, I put its' path to the list. After I finished, list contained all the files' names.

   1.2 Convert each file to in-memory data structure and merge them

   After I had a list that contained all flies' name, I went through this list and converted each file to an in-memory data structure called book_type. It was a map whose key was string and value was also string. It was a little different from the book_type in the code provided on canvas, which was also a map, whose key was integer and value was string. The reason why I changed it from integer to string was that, in the following code, I would merge each maps to a final big map. Since the key was 1000*chapter plus verse, it might be possible that when I merge maps there were elements

that had the same key. My solution to solve this problem was just convert the integer to string and combine it with the filename. The second string(value) of the map was the filename following by the content of each chapter and verse. When I convert each file to a map, I found an error in the code that provided on canvas, which located in line 182 of the file “sample_file_word_count.cpp”. The error was that in the code, it just used colon to split each line to chapter, verse, and text and stored them to a vector and assumed that the third element of the vector was the entire text. The problem was that, sometimes colon also occurred in text and text was splitted to different parts, which meant the third element of vector only contained part of the text. I solved this problem by combining all the elements from third to the end of vector.

After I converted each file to a map, I merged all maps to a big map. As I mentioned before, all the keys would be different, so there would be no problem to merge all maps to one.

1.3 Assign tasks to workers and merge results.

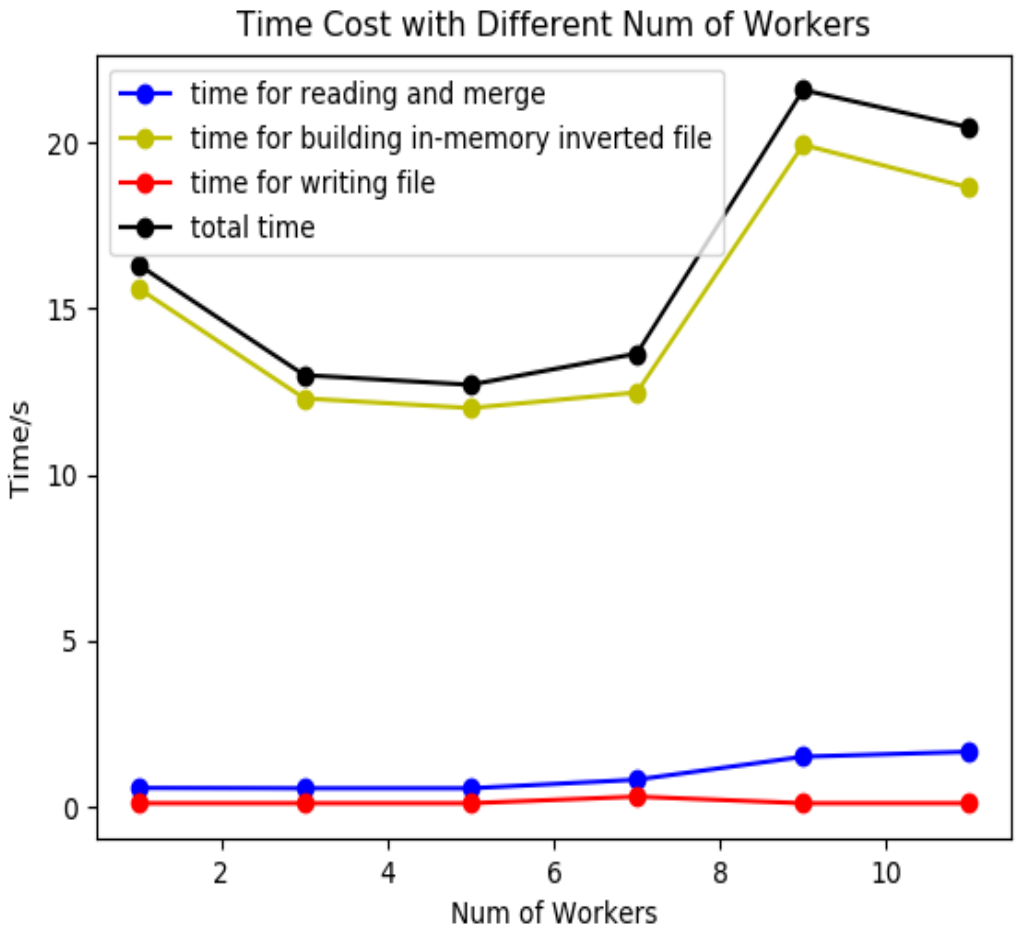
After building the big map, each element of which every line of the origin files corresponding to, master sent a message to each worker. From the message and also the message tag, worker could know the book name, chapter, and verse of this message. When a worker finished the task, it sent a message to master and master sent a new task to the worker. This process repeated until master went through all the elements in the big map.

Rui Huang rhhq7

The format of the message master received is: word1=file,chp:vsX||word2=file,chp:vsx. Then master split this message(string) to different and merge it to final result.

2. Result

I used 1 3 5 7 9 workers to finish this task (without considering the master). Following is the image of the experiment result.



Rui Huang rhhq7

3. Analysis

When number of worker increased from 1 to 5, the time to build the in-memory inverted file went down, and when it increased from 5 to 9, the time increased too and finally, when it increased from 9 to 11, the time decreased again.

The reason why the time went down when number of worker increased from 1 to 5 was very obvious: the more worker, the less task each worker need to do, the less time it cost.

There might be several reasons why the time went up when number of worker increased from 5 to 9. The most important one I think was that if there were more workers, there would be more cost for master to communicate with workers.

There might be another better algorithm to invert file: after building a list that containing all the file name in the directory, master send the name to workers, and each time master receive a message from a worker, master send a new file name to that worker. Repeating this procedure until master go through all the element in the list. When the worker receives a message, which contain the name of the file that needed to be processed, worker begin to work on it. After finishing, worker send the result to master. In this way, if there are 66 files, master need to communicate with workers 66 times totally. In my implement, if there are , for example 10000 lines in 66 files totally, master need to communicate with workers 10000 times, which is much time costly.