CS542200 Parallel Programming
Homework 4: MapReduce

Due: Jan 15 23:59, 2023 (Sunday)

1 Goal

This assignment provides an opportunity for you to practice your parallel programming skills by implementing the scheduling and parallel programming model of the well-known big data processing framework, MapReduce.

2 Requirements

- In this assignment, you are asked to implement a parallel program that mimics the **data locality-aware scheduling policy** and the **functional level programming model** of MapReduce .
- You will implement the parallel program using **MPI and Pthread library**. The jobtracker(scheduler) and tasktrackers(workers) are implemented as MPI processes, and threads are used for executing computing tasks and IO.
- The **jobtracker** is responsible for generating the map tasks, reducing tasks of a MapReduce job and following the data-locality scheduling principle to dispatch tasks on worker nodes for execution.
- Each node runs a **tasktracker** which is responsible for creating and managing a set of **mapper and reducer threads** to execute the receiving map tasks and reduce tasks and outputs the intermediate and final output files.
- **We do NOT consider worker nodes to join, leave or fail during the job execution.**
- You are required to implement MapReduce system architecture, programming model, and scheduling algorithm described in Section 3, 4 and 5, respectively.
- You are required to **implement a WordCount sample code to demonstrate your implementation**.
- All the codes should be compiled into a **single MPI program**, and you should make sure the program terminates properly after all the computing tasks are completed.
- **Performance is not the primary concern in this assignment, but you are still encouraged to improve the code efficiently.**

3 MapReduce System Architecture and Components

- Fig.1 is the system architecture and components that must be implemented in this assignment.
  a. **There is only one jobtracker process in the system, and one tasktracker process on each node.** The number of mapper threads and reducer threads on a node is specified in the job execution command.
  b. A mapper thread is responsible for the data processing of an input file data chunk at a time, and a reducer thread is responsible for the data processing of a set of keys that are hashed to the reducer number according to the partition function.
  c. Each node has only **one reducer thread and (*s*-1) mapper threads**, where ***s* is the number of CPUs allocated on a physical node specified in the execution command**.
  d. The input and output files are stored on the distributed file system (**mimicked by NFS shared folders**), and the intermediate results are written in the local file system (**mimicked by your home directory, so the reducer can directly read its content from the home directory**). To mimic the data locality property in our system, **a configuration file is given to specify the logical mapping between the data chunks and worker nodes**. The file **chunk size is specified as a number of lines**.

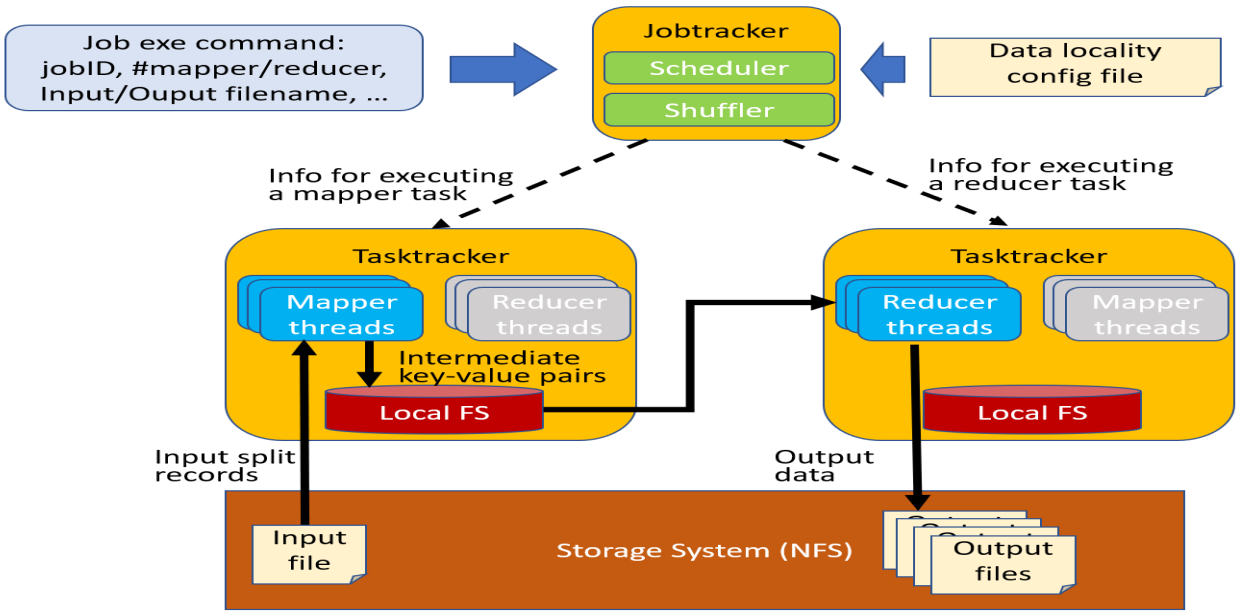


Fig.1: The system architecture of MapReduce

4 MapReduce Programming Model

- You are also required to implement the MapReduce programming model realized by the functions shown in Fig. 2. You can decide the interface and implementation of these functions, but their functionality must match the descriptions below.
  a. Input split function: It reads a data chunk from the input file, and splits it into a set of records. **Each line in the input file is considered as a record. The key of a record is its line number in the input file, and the value of a record is the text in the line.**
  b. Map function: It reads an input key-value pair record and output to a set of intermediate key-value pairs. **You can assume the data type of the input records is int (i.e., line#), and string (i.e., line text), and the data type of the output records is string (i.e.,word), and int (i.e., count).**
  c. Partition function: It is a hash function that maps keys to reducers. The output should be the reducerID, hence **the return value should be bounded by the number of reducers**.
  d. Sort function: It sorts the keys before passing them to the reducers. **The default implementation should follow the ascending order of ASCII code.**
  e. Group function: It contains a comparison function of the keys to determine what are the values that should be grouped together for calling a reduce function. **The default implementation should only group the values with the exact same key together.**
  f. Reduce function: It aggregates the values of a key, and outputs its final key-value pairs. You can assume the data type of the output key-value pair is **string (i.e.,word), and int (i.e., count).**
  g. Output function: It writes all the output key-value pairs of a reduce task thread to an output file stored on NFS. **The default implementation for the output format should be one key per line, and separate the key and values by space.**

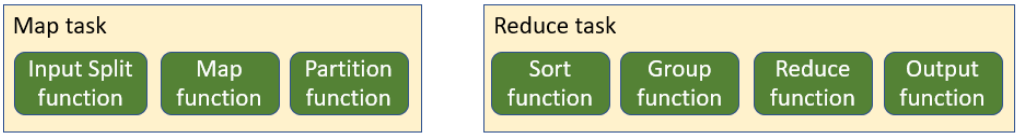


Fig. 2: The required functions implemented by a mapper task and a reduce task.

5 MapReduce Data Locality-Aware Scheduling Algorithm

- MapReduce has a dynamic load-balancing scheduler that dispatches map tasks based on data locality. The reduce task can be randomly scheduled to any available reducer thread.
- Below is the scheduling steps of the MapReduce data locality-aware scheduling algorithm
  1. Read the data locality config file of the input file, which contains each data chunk (chunkID) mapping to the location (nodeID). **If the nodeID is larger than the number of worker nodes, mod the nodeID by the number of worker nodes.**
  2. Generate mapper tasks and insert them into a scheduling queue, where the taskID is the same as the chunkID.
  3. Wait requests from tasktrackers (i.e., nodes), and dispatch mapper tasks to nodes based on the data locality. That is, the tasks are scheduled in a FIFO order, but the task with data locality to the node should be scheduled with higher priority than the task without data locality as shown by the example shown in Fig. 3.
- **If a mapper is reading chunks from a remote location, call sleep for *D* seconds. The value of *D* will be given by the execution command described in the next section.**

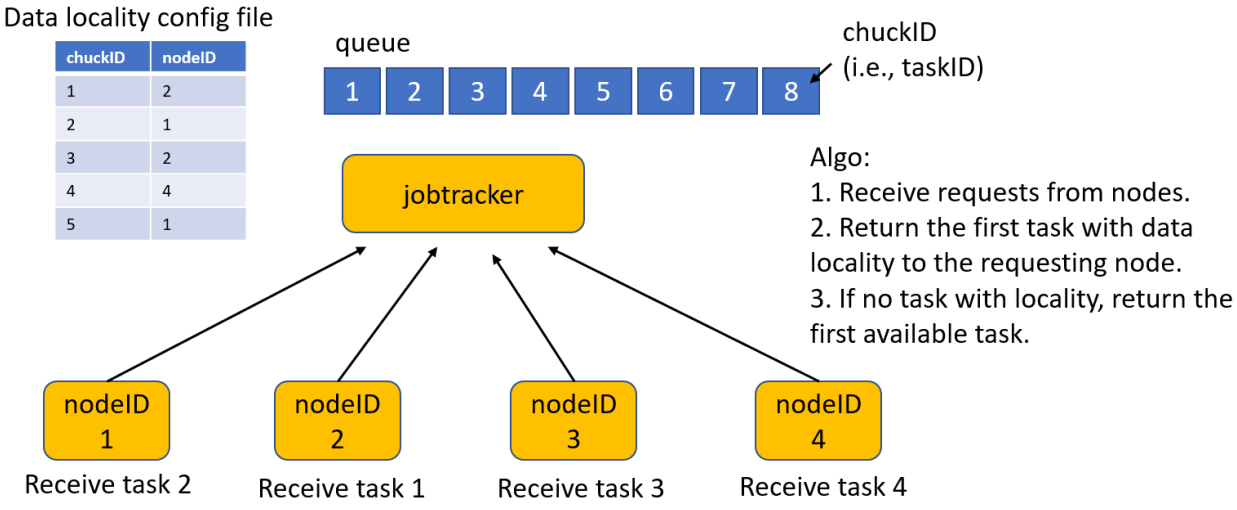


Fig. 3: Data locality-aware scheduling algorithm.

6 RUN YOUR PROGRAMS

- **Command line specification**

```
srun -N<NODES> -c<CPUS> ./mapreduce JOB_NAME NUM_REDUCER DELAY INPUT_FILENAME CHUNK_SIZE LOCALITY_CONFIG_FILENAME OUTPUT_DIR
```

  - NODES: Number of nodes, specified by TA. One of the nodes will act as the jobtracker, and the rest of the nodes will act as the tasktrackers.
  - CPUS: Number of CPUs allocated on a node, specified by TA. Each tasktracker will create CPUS-1 mapper threads for processing mapper tasks, and one reducer thread for processing reducer tasks. More threads can be created by a tasktracker or a jobtracker as needed. Hence, the total number of threads on a node can exceed the number of CPUs allocated on a node.
  - JOB_NAME: The name of the job. It is used to generate the output filename.
  - NUM_REDUCER: The number of reducer tasks of the MapReduce program. The number of mapper tasks will be the same as the number of data chunks of the input file specified in the file locality configure file.
  - DELAY: The **sleeping time in seconds** for reading a remote data chunk in a mapper task.
  - INPUT_FILENAME: The path of the input file. Your program should read the data chunks and file content from this file.
  - CHUNK_SIZE: The number of lines for a data chunk from the input file. Hence, the amount of data read from an input file by the MapReduce program should be the chunk size specified in the command line times the number of data chunks specified in the data locality configuration file, NOT necessarily the whole input file.
  - LOCALITY_CONFIG_FILENAME: The path of the configuration file that contains a list of mapping between the chunkID and nodeID(i.e., tasktrackerID). The file should be loaded into memory by the jobtracker for scheduling.

- OUTPUT_DIR: The pathname of the output directory. All the output files from your program (described below) should be stored under this directory.

- **Output files**

  You are requested to generate the following output files for correctness check.
  - Program output files from each reducer
    - Filename: <job_name>-<reduce_task_ID>.out
    - Content: The wordcount results. E.g., cat 5\ndog 10\n
  - Log file from the jobtracker(i.e., scheduler)
    - Filename: <job_name>-log.out
    - Content: You must log the scheduling event in the format specified below

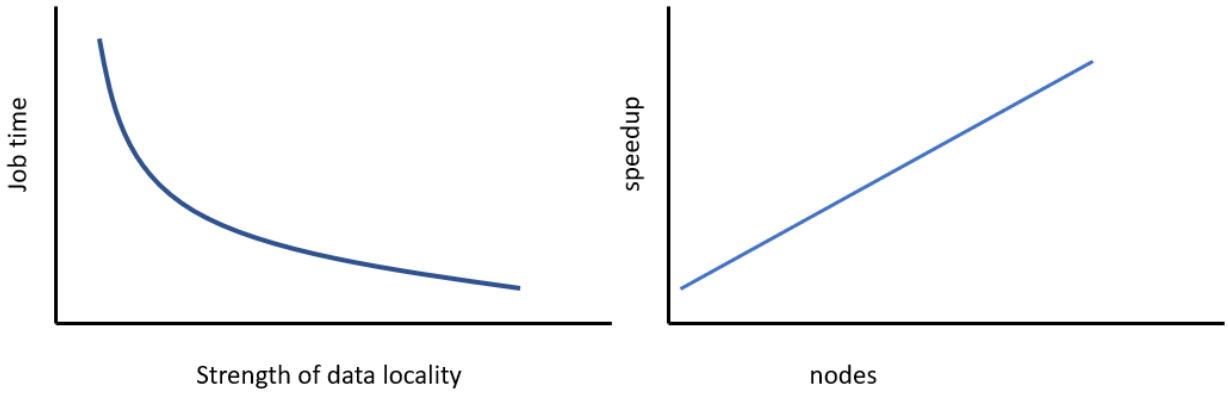
<Time: Unix time format>, <Event: "Start_Job">,<List of Input Arguments:JOB_NAME, NODES, CPUS, NUM_REDUCER, DELAY, INPUT_FILENAME, CHUNK_SIZE, LOCALITY_CONFIG_FILENAME, OUTPUT_DIR>
<Time: Unix time format>, <Event: "Dispatch_MapTask">,<Mapper task ID: INT>, <Mapper ID: INT>
<Time: Unix time format>, <Event: "Complete_MapTask">,<Mapper task ID: INT>, <Mapper Exe Time: seconds>
<Time: Unix time format>, <Event: "Start_Shuffle">,<Total Num of Intermediate K-V pairs: INT>
<Time: Unix time format>, <Event: "Finish_Shuffle">,<Total Shuffle Time: Unix time format>
<Time: Unix time format>, <Event: "Dispatch_ReduceTask">,<Reduce task ID: INT>, <Reducer ID: INT>
<Time: Unix time format>, <Event: "Complete_ReduceTask">,<Reduce task ID: INT>, <Mapper Exe Time: seconds>
<Time: Unix time format>, <Event: "Finish_Job">,<Total Execution Time: seconds>

- Example:

```
1454946120000,Start_Job,Job1,4,8,1,10,/hw4/test/book.txt,100,/hw4/test/book_locality.mat,./output_dir
1454946140000,Dispatch_MapTask,1,1
1454946150000,Dispatch_MapTask,2,3
1454946160000,Dispatch_MapTask,4,2
1454946170000,Dispatch_MapTask,3,4
1454946180000,Complete_MapTask,1,10
1454946190000,Complete_MapTask,2,3
1454946200000,Complete_MapTask,4,11
1454946210000,Complete_MapTask,3,2
1454946210000,Start_Shuffle,144000
1454946210000,Finish_Shuffle,100
1454946220000,Dispatch_ReduceTask,1,3
1454946320000,Complete_ReduceTask,1,34
1454946320000,Finish_Job,288
```

7 REPORT

1) List the highlights of your implementation.
   - **You should speak these highlights during the demo to show your novelty and efforts.**
2) Detail your implementation of the whole MapReduce program.
   - You shouldn't explain your code line-by-line.
   - **You should discuss what are the challenges in your implementation, and how you solve them.** For instance, how you properly determine the termination condition in your problem, or how you implemented the data locality-aware scheduling algorithm, etc.
3) Conduct experiments to **show the performance impact of data locality**, and the scalability.
   - You can generate your own data locality configuration file to adjust the strength of data locality. For instance, you may reduce the strength of data locality by limiting the number of nodes for storing the data chunks.



4) **Experience & conclusion**
   - What have you learned from this homework?
   - Feedback (optional)

8 GRADING

1. **[60%] Correctness**
   - A number of test cases will be used to verify the correctness of your program before the demo.
     - Most of them are just changing the input parameters in the execution command by giving different sizes of input file, different number of mappers, reducers, and nodes, etc. Noted, the number of mapper tasks may exceed the number of mapper threads.
     - The correctness of data locality scheduling will be checked.
     - The program must be terminated properly after job execution.
     - The output file format and results must be correct.
   - You will also be asked to **modify the partition/sort/group functions during the demo** for correctness check.
     - E.g., sort key in descending order instead of ascending order, or group by the first charter instead of the whole word, etc.
2. **[20%] Demo**
   - A demo session will be held remotely. You'll be asked questions about the homework.
   - You will be given a chance to highlight the novelty of your implementation.
3. **[20%] Report**
   - Grading is based on your evaluation, discussion and writing. If you want to get more points, emphasize more on how good your implementation is in terms of design or programming techniques, or present more experimental results to demonstrate the characteristics of a MapReduce program.

9 SUBMISSION

Upload the files below to eeclass. (**DO NOT COMPRESS THEM**)

- *.cc, *.h
- Makefile **(required)**
- hw4_{student_ID}.pdf

10 Final Notes

- Resources are provided under `/home/pp22/share/hw4/`:
  - `testcases/` - sample test cases
  - `threadpool/` (you can choose not to use)
- Contact TA via pp@lsalab.cs.nthu.edu.tw or eeclass if you find any problems with the homework specification, example source code or the test cases.
- You are allowed to discuss and exchange ideas with others, but you are required to write the code on your own. You'll get **0 points** if we found you cheating.
- You need to make sure that your program works. Your points will be **discounted** if any error like a compile error occurs.
- No judge and scoreboard in this homework.