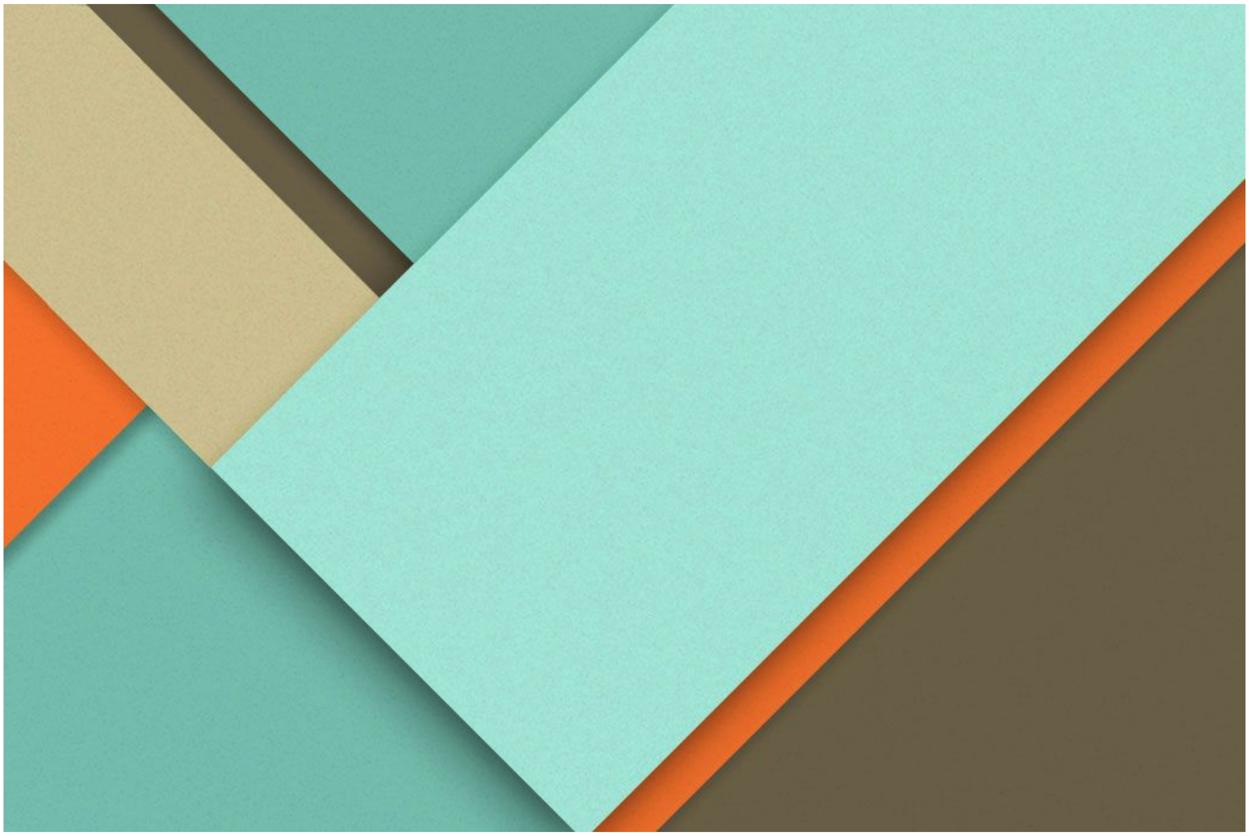


Implementing Distributed MapReduce in Python

10.20.2015

Jiao Wang

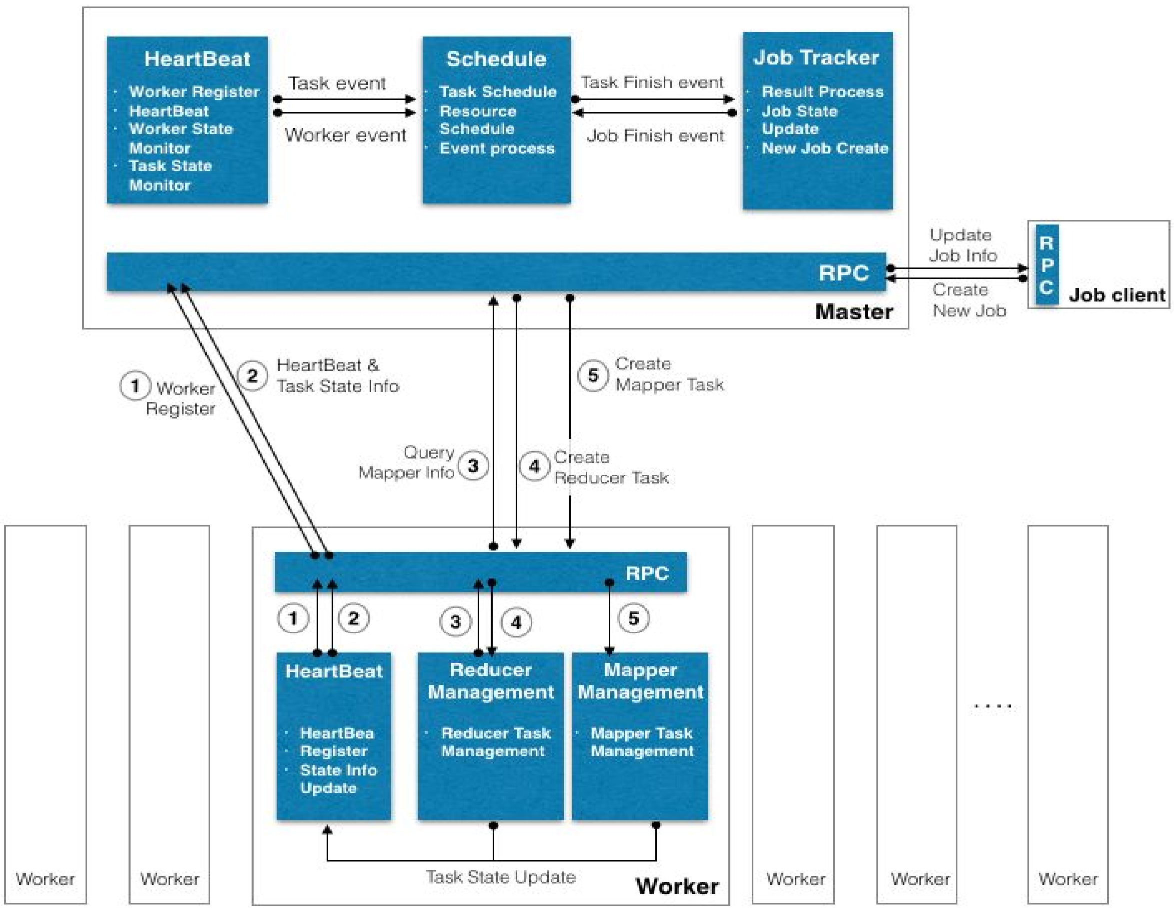
Yuanyuan Zhang

University of San Francisco

Goals

Implement a reliable, distributed MapReduce framework in Python using ZeroRPC and Gevent

Project Design



Master: has 3 three threads, schedule thread, heartbeat thread, Job track thread.

Schedule thread

schedule thread has three kinds of jobs: transaction handling, task schedule and reducer schedule, use gevent to control three jobs running

Data Structures:

Job table:

ID	Job name	splits	num_reducers	infile	outputfile	status	progress

worker table:

ID	Addr	Status	num_heartbeat	num_callback	mapper status	reducer status

mapper status table:

Job ID	Split ID	Task ID	Status	Progress	Finish or not

reducer status table:

Job ID	Partition ID	Task ID	Status	Progress	Finish or not

MapTask table:

Job Id
Task Id
Worker name
Status
Finish or not
Num_reducers
Split Id

Split info
Partitions

Reduce Task table:

Job Id
Task Id
Worker name
Status
Finish or not
Num_mappers
Split Id
Split info
Partition Id
Partitions

Message handling

Scheduler need to handle the transactions from Job client, Heartbeat thread and work RPC request.

1. worker RPC request

   build worker information record if receive RPC request

2. receive job from client

   when receive the job request from client, it will collect all the needed information, then create task table, write job info into to table,split file and prepare to assign map and reduce task.

3. from heartbeat

- if map task finish notification

  update task job table

  inform reducer collect data and update reducer table

  assign new task to mapper if job waiting lists is not null and update worker table and task table

- if reducer job finish notification

  update reducer table

  if there is job in job waiting list, assign new job to available reducer and update all related table

- if all reducer finish

  trigger job finish transactions, collect all reducers' result and inform client

- if worker down

  walkthrough Maptask table, clear the all the worker's map tasks, add to these tasks map task queue

  walkthrough Reducer table, clear the all the worker's reduce jobs, add these jobs to reduce job queue

Map task schedule

walkthrough task table:

- if current task is waiting to assign:

  query worker table to check if there is availbale map can do this task, if yes, assign task and update related table,if not, continue walking

- if current task is in progress, continue next
- if current task is in progress, continue next

Reduce job schedule

walkthrough reduce job table

- if status is finished , next
- if status in processing

inform reducer to grab map immediate result data

- if status is unassigned

  query worker table,check if there is available reduce can handle the job

  if yes, create reduce process ,assign job, update related table

- if all the reduce job finished , triggle client job finishing transaction

Heartbeat thread

Heartbeat thread has two main jobs: monitor task status and monitor work status. All the communication between master and worker is reached by heartbeat. If it received any transaction happened in workers, it will notice schedule thread to handle. The benefit of this kind design is reducing the coupling between schedule thread and heartbeat thread.And also keep the data safe, since all the data structures are maintained by scheduler.

Data structure:

1. worker and tasks monitor

   worker #id

   Status : live or not

   MapSlot : task id task status

   ReduceSlot: task id task status

Basic algorithm:

while :

read worker's data and build status monitor record

send heartbeat to all worker which already check in

if receive the response from worker

update status monitor record if needed

if there is task finish notification from work node

notice scheduler

if can't receive response in limit time

mark this work node down ,inform scheduler and update worker status

Job Tracker Thread

Job tracker thread mainly has three kind job, create jobs , update Job information when received the notification and collect reducers' results.

Work Node failure handling process

When master receive worker down message, it find out all the map tasks and reduce jobs which are already done by the worker or not done but already assigned to the down worker.Then mark their status to unassigned. Then these map tasks or job will be reassigned to others available worker.

File split algorithm

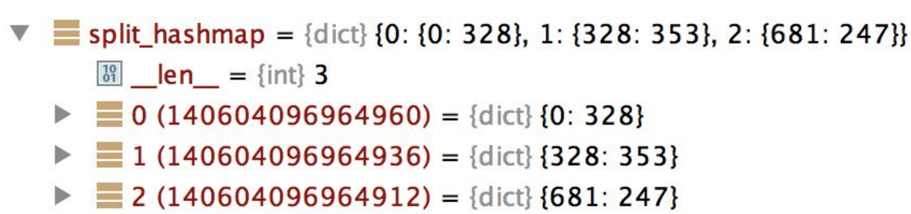
when master receive the job from client, it need to divide job into small jobs, then assign to work nodes.Basic idea of our implementation is: start from 0th bytes in the file, then according to split size recursively generate the {key:value} pairs. Key is the offset, value is the size each work node to handle. However, most time we need to adjust the size slightly. E.g. word count, if we only divided by split size, somethime we may split one word into 2 pieces. so in this case we will continue to read until current line end. And also in AscII hamming decoding or fix and so on, we need to make sure adjust the split size as a multiple of twelve,otherwise we cannot have correct result.

input :

split size and filename

output:

{0:{offset:size},...i:{offset:size}}, k means the ith divided chuck according to the split size



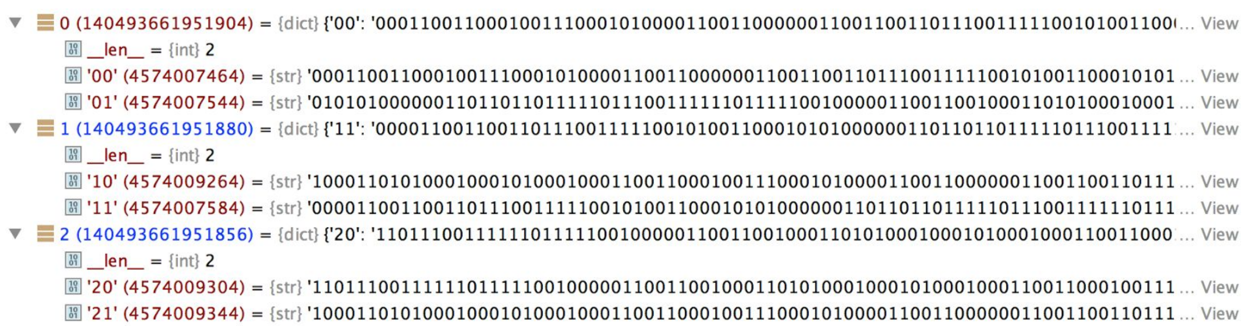
The above figure is the split result of word sort , split size is 256.

Immediate map results partition algorithm

when map phase finish, we need to switch partition status which will divide the immediate result of map into pieces for the continuing reduce phase. Here our algorithm is dividing by number of reducers, each map immediate result will be divided into num of reducers pieces, the out is also in the format of map. Value is the data which reduce need to handle. Key (xx) is composed of two meaningful numbers , first one is the split_id which mean it comes from which split chuck, and second one partition_id which means ith part of the immediate result. The key here is very important, because it is the critical information for keeping result data in order when we collect data from each work node and also very important after node failure, we need to figure out which part job need to redo.

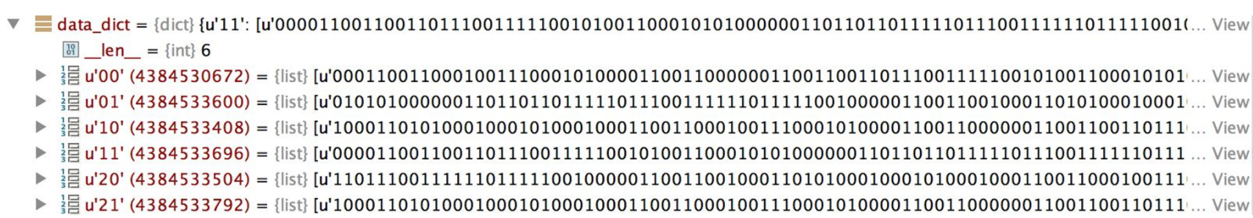
Take hamming encode as an example: 3 maps 2 reducers

the out level 0, 1 ,2 is the split_id, in our implementation each split has a map,so it also can be considered as ith map, so here, 00 means first partition of first split. 11 means second partition of second split and so on.



here we still can collect data in order if we want, just in the order 0th,1th,2th map

However, 00 ,10,20 will be assigned to reducer 0 , and 01,11,21 will be assigned to reducer 1, in that case, we will hard to figure out the order when we collect data from reduce if we didn't have these key. The below diagram will show how these key help to keep data in order.



This is the status that our collector gather all the data from 2 reducers. Still with the keys, so we just need to sort the keys, then get the data in the order of the key.

References

http://the-paper-trail.org/blog/the-elephant-was-a-trojan-horse-on-the-death-of-map-reduce-at-google

http://cs.gmu.edu/~menasce/papers/CMG2013-Shouvik-Menasce-Final.pdf