International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2017 IJSRCSEIT | Volume 2 | Issue 5 | ISSN : 2456-3307

Hybrid Job-Driven Scheduling for Heterogeneous MapReduce Clusters

J. Sivarani[1], T. Subramanyam[2]

[1]Department of Computer Science, Sri Padmavathi University, Tirupati, India
[2]Asst. Professor, Department of Computer Science Sri Padmavathi University, Tirupati, India

ABSTRACT

It is cost-efficient for a tenant with a limited budget to establish a heterogeneous virtual MapReduce clusters by renting various virtual private servers (VPSs) from a VPS provider. To provide an appropriate scheduling scheme for this type of computing environment, and MapReduce still performs poorly on heterogeneous clusters, we propose in this paper a hybrid job-driven scheduling scheme (JoSS for short) from a tenant perspective. JoSS provide not only job level scheduling, but also Map-task level scheduling and Reduce-task level scheduling; The deployment of MapReduce in data canters and clouds present several challenges, improve data locality for both map-level task and reduce-level task, avoid job starvation and improve job execution performance. Two variations of JoSS-Task and JoSS-Job are further introduced to separately achieve a better map-data locality and a faster task assignment. We conduct extensive experiments to evaluate and compare the two variations (JoSS-T and JoSS-J) with current scheduling algorithms supported by Hadoop. The result shows that the two variations crush the opposite tested algorithms in terms of map and reduce data locality , and network overhead while not acquisition significant overhead. Additionally, the two variations area unit severally appropriate for various MapReduce-workload eventualities and supply the most effective job performance among all tested algorithms.

Keywords : MapReduce, Hadoop, Map-task Scheduling, Reduce-task Scheduling, Heterogeneous virtual MapReduce clusters

I. INTRODUCTION

MAPREDUCE [1] is a really popular paradigm in distributed programming model proposed by Google to process vast amount of data in parallel manner. Due to programming-model simplicity, data distribution, scalability, and fault tolerance, MapReduce and its open-source implementation is known as Hadoop [2], have been employed by many organizations , including Facebook, Amazon, IBM, Twitter, and Yahoo!, to process their business data. MapReduce has also been used to solve diverse applications, such as text tokenization, indexing, search, data mining [3], health care [4], machine learning [5], statistical modelling [6], bio informatics [7], social network [8], and astronomy [9].

MapReduce permits a computer user to define a MapReduce job as a map function and reduce function and provides a runtime system to divide the job into multiple map and reduce tasks and perform these tasks on a MapReduce clusters in parallel. Typically, a MapReduce cluster consists of a collection of commodity machines/nodes located on several racks and many racks and interconnected with one another in a local area network [LAN] [10].During this paper, we call this a conventional MapReduce clusters. As a result of the very fact that building and maintaining a standard MapReduce cluster is costly for a person/company with a restricted budget, an alternative way to establish a heterogeneous virtual MapReduce cluster by either renting a MapReduce framework [11] from a MapReduce services provider. (E.g. Amazon) or dealing multiple virtual private servers (VPSs) from a VPS provider [12]. Each VPS could be a virtual machine with its own OS and disk space. Due to some reasons, such as availability issues of a data center or resource shortage on a popular datacentres, a tenant may rent VPSs from different datacentres operated by a same VPS provider to determine his/her heterogeneous virtual MapReduce cluster [13]. During this paper, we have a tendency to target a heterogeneous virtual MapReduce cluster of this type.

For a person person/company that a build conventional MapReduce clusters map-data locality (when the data is local on the same node as the mapper working on the data) in the cluster is categorized into data locality, node locality and different rack [15]. Since the person/company is aware of physical interconnection and placement among all nodes and racks. However, for a tenant who builds a heterogeneous virtual MapReduce cluster, the tenant only knows each VPS's IP address and each VPS's data center location (E.g. place name). Other information such as physical machine and rack that each VPS belongs to is not released by the provider. Hence from the tenants point the map-data locality in his/her heterogeneous virtual MapReduce cluster can only be categorized into the following three levels:

VPS- locality, it means that a map task and input data are located in the same heterogeneous VPS.
Cen-locality, it means that a map task and its input data are within the data centre, but not at the same heterogeneous VPS.

Off-cenlocality, it means that map task and its input data are located at different datacentres.

Reduce-data locality is rarely designed in a conventional MapReduce[14] cluster since reducing the distance between a reduce task and its input data coming from all the related map tasks in a LAN. But this is achievable in a heterogeneous virtual MapReduce cluster compromising multiple datacentres.

MapReduce in data centres or cloud platforms offers a more cost effective model to implement big data analytics. Hardware heterogeneity occurs because servers are gradually upgraded and replaced in data centres. Interference from multiple tenants sharing the same cloud platform can also cause heterogeneous performance even on homogeneous hardware. The difference in processing capabilities on MapReduce nodes breaks the assumption of homogenous clusters [16] in map design and can result in load imbalance. Which may cause poor performance and low cluster utilization. To improve MapReduce performance in heterogeneous environment, Work has been done to make task scheduling and load balancing heterogeneity aware. Despite these optimizations, most MapReduce implementation such as Hadoop still performs poorly in heterogeneous environment.

The MapReduce distributing model runs on a large data cluster consists of homogenous nodes also assumes the homogeneous workload when making a scheduling decision. MapReduce take care of the details of partitioning the input data, scheduling the program's execution. The MapReduce performance depends on the previous properties which appear obviously in the homogeneous environment. The homogenous environment assumptions have been broken as:

It is not always possible or even desirable to have a big cluster consists of only one type of machine.
It is unsatisfied virtualized data center.

It does not take the difference of workload characteristics between jobs into account when making a schedule decision from VPS provider.

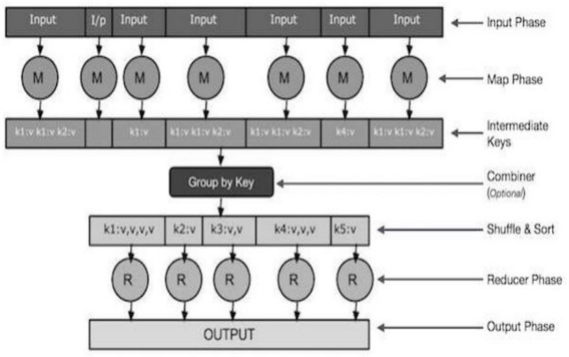
Thus, the need of employing the MapReduce model on a heterogeneous environment becomes necessary for hybrid jobs. The heterogeneity environment affects the performance of the MapReduce algorithms. Many researchers [17], [18], [19], [20] and [21] has discussed how the heterogeneity affects the MapReduce performance and developed algorithms to improve performance in heterogeneous environments.

We propose an appropriate scheduling scheme for a tenant to achieve a high map-and-reduce data locality and improve job performance in his/her heterogeneous virtual MapReduce cluster, so we propose a hybrid job-driven scheduling scheme (JoSS) JoSS classifies MapReduce jobs into either large or small jobs based on each job's input size to the average datacenter scale of the heterogeneous virtual MapReduce cluster, and further classifies small MapReduce clusters jobs into either map-heavy or reduce-heavy based on the ratio between each job's reduce-input size and the job's map-input size. Then JoSS uses a particular scheduling policy to schedule each class of jobs such that the corresponding network traffic generated during job execution can be reduced, and the corresponding job performance can be improved. In addition, we propose two variations of JoSS, named JoSS-T and JoSS-J, to guarantee a fast task assignment and to further increase the VPS-locality, respectively. for heterogeneous clusters by providing scheduling in three levels: job, map task, and reduce task. In this paper JoSS-T refers for the Tasks (both map and reduce) and JoSS-J refers for the job. We implement JoSS-T and JoSS-J in Hadoop-2.7 and conduct extensive experiments to compare them with several known scheduling algorithms supported by Hadoop, including the FIFO algorithm [22], Fair scheduling algorithm [23], and Capacity scheduling algorithm [24]. The experimental results demonstrate that both JoSS-T and JoSS-J outperform the other tested algorithms in terms of map and reduce-data locality, and network overhead without causing too much overhead, regardless of the particular job type and scale.

The contribution of this paper is as follows:
Introduces an overview of the MapReduce model, a brief introduction to Hadoop, and the MapReduce approaches in heterogeneous environment.
JoSS to appropriately schedule MapReduce jobs in a virtual MapReduce cluster by addressing both map-data locality and reduce-data locality from the perspective of a tenant.
By classifying jobs into map-heavy and reduce-heavy jobs and designing the corresponding policies to schedule each class of jobs and scheduling them in a round-robin fashion, JoSS avoids job starvation and improves job performance.
A formal proof is also provided to determine the best threshold for different MapReduce jobs.
Two variations of JoSS (i.e., JoSS-T and JoSS-J) are introduced to respectively achieve two conflicting goals: speeding up task assignment and further increasing the VPS-locality.
MapReduce benchmarks to create two different MapReduce workloads for evaluating and comparing JoSS-T and JoSS-J with three known scheduling algorithms supported by Hadoop. Moreover, a set of metrics showing data-locality, network overhead, job performance, and load balance.
The rest of this paper is organized as follows. Sections2. Survey MapReduce. Sections3. Related work, respectively. Section 4 presents the JoSS and the two variations. Section 5 derives the best threshold to classify map-heavy jobs and reduce-heavy jobs. In Section 6, Extensive experiments are conducted and discussed. Section 7, concludes this paper.

II. MAPREDUCE

The MapReduce [1] contains two important tasks, namely Map and Reduce. The map task is done by means of Mapper Class. The reduce task is done by means of Reducer Class. The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs).The Reduce task takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples. The reduce task is always performed after the map job.



Input Phase, Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key, value pairs. Map, Map is a user-defined function, which takes a series of key-value pairs and processes. Intermediate Keys, The key-value pairs generated by the mapper are known as intermediate keys. Combiner, A combiner is a type of local Reducer that groups similar data from the map phase. Shuffle and Sort, the Reducer task starts with the Shuffle and Sort step.

Reducer, The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Output Phase, In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

III. RELATED WORK

Analysis of structured data has seen tremendous success in the past. However, analysis of large scale unstructured data in the form of video format remains a challenging area. YouTube, a Google company, has over a billion users and generates billions of views. Since YouTube data is getting created in a very huge amount and with an equally great speed, there is a huge demand to store, process and carefully study this large amount of data to make it usable The main objective of this project is to demonstrate by using Hadoop concepts, how data generated from YouTube can be mined and utilized to make targeted, real time and informed decisions. The project utilizes the YouTube Data API [25] (Application Programming Interface) that allows the applications/websites to incorporate functions that are used by YouTube application to fetch and view information. The Google Developers Console is used to generate a unique access key which is further required to fetch YouTube public channel data. Once the API key is generated, based console application is designed to use the YouTube API for fetching video(s) information based on a search criteria. The text file output generated from the console application is then loaded from HDFS.

The First In First Out(FIFO) scheduling algorithm is a default scheduling algorithm provided by Hadoop , Reduces response time due to speculative execution. Works well in the case of only short jobs. It follows a strict job submission order to schedule every map task of a job and mean while attempt to schedule a map task to an idle node that's near the corresponding map-input block. Uses fixed threshold for selecting tasks to reexecute. Can't identify which tasks to be reexecuted on fast nodes correctly. The FIFO algorithm only focuses on map-task scheduling, rather than reduce-task scheduling. Hence, when FIFO is adopted in a heterogeneous virtual MapReduce cluster, its low reduce-data locality might cause a long job turnaround time. FIFO is used to achieve node locality and rack locality in conventional MapReduce clusters, rather than achieving the VPS-locality and Cen-locality in a heterogeneous virtual MapReduce cluster. Consequently, the map-data locality of FIFO might be low in a heterogeneous virtual MapReduce cluster.

In addition to the FIFO scheduling algorithm, Hadoop also provides the fair scheduling algorithm and the capacity scheduling algorithm. The first fair scheduling is proposed by Facebook to fairly assign computation resources to each job in a cluster such that all jobs shares equal resources overtime. Job weight is not considered for each node. The later, introduced capacity scheduling algorithm by Yahoo!, also allows multiple users to share a MapReduce cluster. It supports multiple queues and allocates a fraction of a cluster's computation resources to each queue, i.e., all jobs submitted to a queue can only access to the resource allocated to the queue. Similar to these two algorithms, JoSS allows multiple jobs to simultaneously share the computation resource of a virtual MapReduce cluster. User needs to know system information and make queue set and queue select group for the job. But different from the two algorithms (fair scheduling and capacity scheduling), JoSS further provides reduce-task scheduling to improve job performance.

We propose JOSS to appropriately schedule Map Reduce jobs in a virtual Map Reduce cluster by addressing both map-data locality and reduce-data locality from the perspective of a tenant. By classifying jobs into map-heavy and reduce-heavy jobs and designing the corresponding policies to schedule each class of job, JOSS increases data locality and improves job performance.

IV. THE PROPOSED SCHEME

In this section, we describe how JoSS schedules MapReduce jobs in a heterogeneous virtual MapReduce cluster consisting of n datacenters, n>1. Let cenc be the cth datacenter supporting the composition of the heterogeneous virtual MapReduce cluster, c=1,2,...,n. Let NVPS be the number of VPSs provided by cenc, NVPSc > 1. Let VPSc be the mth VPS provided by cenc,=1,2,...;NVPS;c. Assume that each VPS has only one map slot and one reduce slot, i.e., at most one map task and one reduce task can be performed by a VPS simultaneously. For each datacenter in cenc, JoSS maintains two permanent queues, denoted by MQc;0 and RQc;0, to respectively put the map tasks and the reduce tasks that are scheduled datacentres to be executed by VPSs at cenc.

Let J be a MapReduce jobs submitted by a user, and D is the input data processed by J. Based on the predefined block size S, D will be divided into m blocks B1,B2,...,Bm where m Let Bi is the i-th block of D, i ¼ 1, 2,...,m . According to the total number of the blocks of the jobs, J is divided into the same number of map tasks.

Let M i be the ith map task that processes B, i=1,2,...,m. Let r be the number of reduce tasks of J, and let Rj be the jth reduce task of J where j=1, 2,...,r and r>1. In the following, a VPS performing a map task is called a mapper, whereas a VPS running a reduce task is called a reducer.

4.1 Job Classification

Before we are introducing the algorithm of JoSS, we first describe how the JoSS classifies different types of jobs and schedules each class of jobs. Let Rreduce and Mmap be the total reduce-input size and the total map-input size of J, respectively. Based on the ratio of Rreduce over Mmap, J can be classified into either a reduce heavy job or a map-heavy job of the all jobs. If J satisfies, implying that the network overhead is dominated by the J's reduce-input data set for the you tube, then J is classified as a reduce-heavy job (RH job for short). Otherwise, J is classified as a map-heavy job (MH job for short). Note that the td is a threshold to determine the classification, td >0: The best value of the td will be derived as the

Rreduce /Mmap> td

In fact, Mmap;the size of Bi, and Rreduce; where FPi is the filtering percentage of Bi showing the ratio of Mi's map-output size of the all jobs over Mi's map-input size of the all job, FPi>0.

In order to reduce and the above classification, we chose six MapReduce benchmarks: Word-Count, Grep, Inverted-Index, Sequence-Count, Self-Join, and Term Vector from PUMA to conduct the experiments on the youtube dataset. The purpose is to study the minimum playing the youtube dataset of the among the filtering-percentage values of all map tasks of a MapReduce job. In the first experiment, we randomly selected you tube dataset from the youtube dataset is generated by the user views based on the admin can be uploaded in the web; from the you tube data set to be the input of each benchmark.

4.2. Scheduling Policies

Policy A :

This policy is only designed for the small RH job. If J is a small RH job, it would be better that each reducer of J is close to the all mappers of J since the reducer can more quickly retrieve its input data from all the mappers from the input data.

But this also implies that all the mappers of J should be close to each other. The , policy A works as follows. It first chooses cen_w, which is a datacenter having the atleast amount of unprocessed tasks among all the k datacenters in a VPS, Then it schedules all tasks of J to cen_w by putting J's map tasks and J's reduce tasks at the end of the all jobs MQw,o and RQw,o, respectively.

Policy B :
This policy is only designed for the small MH job. If J is a small MH job, it would be better that each mapper of J is close to its input block of the datacenter, and each reducer of J is close to most mappers of J jobs. The, policy B works as follows: It schedules J's all map tasks based on the number of unique input blocks of J held by each datacenter of the VPS. If a datacenter holds more unique blocks of J, more map level tasks of J will be scheduled to the VPSs at this datacenter.

Policy C :

This policy is only designed for the large jobs. If J is in the large job to a virtual MapReduce cluster of the VPS, using one datacenter of the cluster to run all map level tasks of J might need several rounds to finish these map level tasks, implying that the job turnaround time will be prolong. To prevent this from happening, it is the better not to use a single datacenter to run all these map level tasks.

The , as long as J is the large job, JoSS utilizes policy C, which in fact uses the same strategy of policy B to schedule all (Map and Reduce) tasks of J. However, in policy C, all the map level tasks scheduled to cenc will not be put into MQc;0 since MQc;0 is reserved for the only small jobs. Instead, these map level tasks will be put into a new map-task queue created for cenc. Similarly, the reduce level tasks of the large job scheduled to cenc will be put into a new reduce-task queue created for cenc, rather than RQc;0. The purpose is to separate large jobs and small jobs into different queues and allow JoSS to avoid job starvation and also improve the execution performance; And VPS of the both Map data locality and Reduced data locality.

4.3 JOSS And Its Two Variations

JoSS consists of three types of components: input-data classifier, task scheduler, and task assigner. The input-data classifier is designed to classify the input data uploaded by a user into one of the two types: web document and non-web document. The web document refers to a file consisting of a lot of tags and text enclosed in angle brackets. By simply inspecting the first several sentences of a document and without tags, the inputdata classifer can easily know if it is a web document or not. After the classification, the input-data classifier records the type of the input data in the JoSS . Whenever receiving a MapReduce jobs from a user, the task scheduler determines the type of the job and then schedules the job based on either policy A, B, or C. The task assigner then determines how to assign a task to a VPS whenever the VPS has an idle slot.

The algorithm of the task scheduler. Upon receiving J, the task scheduler retrieves J's input data type classified by the input-data classifier and checks whether JoSS has executed J on such input-data type or not by calculating the corresponding hash value and cen1.
comparing the value with H, where H is a set of hash values previously generated and recorded by JoSS. If the hash value is not in H (see line 4), it means that JoSS does not know J's average filtering-percentage value and J's job classification. To obtain the above information, the task scheduler simply appends J's all map tasks and J's all reduce tasks to two queues, denoted by MQ!"!# and RQ!"!#, respectively. This allows the task assigner to use the Hadoop FIFO algorithm [1] to assign these tasks to idle VPSs. Once J is completed, JoSS records the corresponding hash value and averge filtering-percentage value. However, if the hash value is in H (see line 7), it means that JoSS knows the average filtering-percentage value of J. Then the task scheduler schedules J as follows: If J is a small RH job, the abovementioned policy A is used to schedule the tasks of J (please see lines 9 to 12).

Task Scheduler of JoSS

Input: J and input data

Output: task scheduling result

Procedure:

```
1:      Calculate the hash value for J's executable code
        and J's data of input type;
2:      Let H be a set of the hash values
         previously generated by JoSS;
3:      if the hash values are not in H
4:      {
5:              add all map level tasks of the J
                 to the end of MQfifo;
6:              add all reduce level tasks
                of the J to the end of RQfifo;}
7:      else
8:      {
9:              if J is the small RH
                jobs{    // Using the policy A.
10:             Let cenn be the data center having
                 the least unprocessed tasks
cen1,cen2,....cenm;
11:             add all map level tasks of the J
                to the end of MQw,0;
12:             add all reduce level tasks
                of the J to the end of RQw,0;}
13:     else
14:     {
15:             Let Lc is the set of all unique input
                 blocks of held by cenc
16:             where c=1,2,....,n;
17:             let  a=n;          //n is the number
                 of map level tasks of J//
18:             while a>0{// not all map level tasks of
J are scheduled//
19:             Let Ld be the first largest among of all
jobs L1, L2, „„Ln;
20:             Let ILdI is the size of Ld;
21:             Let cend is the related datacenter;
22:         if J is the small MH jobs{          // Using the policy B.
23:             add ILdI all map level tasks of the J to the end of MQd,0;}
24:         else { // J is the large job, so using the policy C//
25:             let v be the total number of map level task queues in cend;
26:             Generate a new map level task queue MQd,v+1;}
27:             add ILdI all map level tasks of the J to the end of MQd,v+1;}
28:         for c=1ton
29:             delete a block from Lc if the block is ib Ld;}
30:         a=a-ILdI;}
31:         Let cene be the data center holding the
             largest number of unique input data blocks of J;
32:                 if J is the small MH jobs{
        // Using the policy B.
33:                 add ILdI all map level tasks of the J to the end of RQc,0;}
34:         else { // J is the large job, so using the policy C//
35:                 let v be the total number of map level task queues in cend;
36:                 Generate a new map level task queue RQc,v+1;}
37:                 add ILdI all map level tasks of the J to the end of RQc,v+1;}}}
```

Figure 1: The algorithm of the task scheduler

Otherwise, it means that *J* is either a small MH job or a large job, and the task scheduler uses lines 14 to 37 to schedule *J*. Recall that policies B and C are used to schedule a small MH job and a large job, respectively. If *J* is a small MH job, the task scheduler directly inserts *J*'s map tasks to the permanent map-task queue of the determined datacenter (see line 22), and also inserts *J*'s reduce tasks to the permanent reduce-task queue of the determined datacenter (see line 33). In other words, no additional queue will be created for any small jobs. The purpose is not to increase the queue management overhead of JoSS. In another case, if *J* is a large job, the task scheduler additionally generates a new map-task queue and a new reduce-task queue to respectively put *J*'s map tasks and *J*'s reduce tasks (see lines 24 to 26 and lines 35 to 37). This will allow the task assigner to properly assign small jobs and large jobs to VPSs.

Task- driven & Task-assigner(TTA)

Input: an idle slot for the all input data VPSc,l

Output: a task assigned to the result as VPSc,l

Procedure:

```
1:      Let Imap and Ired be the two indexes with the same initial value is 0;
2:      while VPSc,l has an ideal slot
3:      {
4:                  Let Nmap be the total number of map level tasks queues in cenc;
5:                  Let Nred be the total number of reduce level tasks queues in cenc;
6:          if the slot is a map{//the idle slot is a map slot;//
7:                  if MQfifo it is not empty{
8:                  Use FIFO to assign the map level task from MQfifo to VPSc,l
9:                  Delete  the task from MQfifo;}
10:                 else{
11                  Imap=Imap mod(Nmap+1);
12:                 To assign the first map level task from MQc,Imap to VPSc,l;
13:                 Delete the task from MQc,Imap;
14:                 Imap++;}}
15:           else{ // the idle slot a reduce slot;//
16:               if RQfifo it is not empty{
17:               Use FIFO to assign the reduce level task from RQfifo to VPSc,l
18:               Delete the task from RQfifo;}
19:           else{
20:               Ired=Ired mod(Nred+1);
21:               To assign the first re level task from MQc,Ired to VPSc,l;
22:               Delete the task from MQc,Ired;
23:               Ired++;}}}
```

Figure 2. The algorithm of task-driven & task assigner (TTA)

Recall that two variations of JoSS (i.e., JoSS-T and JoSSJ) are proposed in this study. The former combines the abovementioned task scheduler and a Task-driven Task Assigner (TTA) to provide a fast task assignment. The latter combines the task scheduler and a Job-driven Task Assigner (JTA) to further improve the VPS-locality. Fig. 5 illustrates how TTA works. Whenever $VPS!,\ell\ell$ has an idle map slot, TTA preferentially assigns a map task from $MQ!"!\#$ to $VPS!,\ell\ell$ based on the Hadoop FIFO algorithm (see lines 7 to 8). The goal is to preferentially execute all newly submitted jobs one by one and obtain their filtering-percentage values to determine their job classifications. However, if $MQ!"!\#$ is empty, TTA assigns one of the first map tasks from all the other map-task queues of $cen!$ in a round-robin fashion (see lines 10 to 13) such that tasks can be assigned quickly and job starvation can be avoided.

Job- driven & Task-assigner(JTA)

Input: an idle slot for the all input data VPSc,l

Output: a task assigned to the result as VPSc,l

Procedure:

```
1:    Let Imap and Ired be the two indexes with the same initial value is 0;
2:    while VPSc,l has an ideal slot
3:    {
4:            Let Nmap be the total number of map level tasks queues in cenc;
5:            Let Nred be the total number of reduce level tasks queues in cenc;
6:        if the slot is a map{//the idle slot is a map slot;//
7:            if MQfifo it is not empty{
8:            Use FIFO to assign the map level task from MQfifo to VPSc,l
9:            Delete the task from MQfifo;}
10:           else{
11            Imap=Imap mod(Nmap+1);
12:           To assign the first map level task from MQc,Imap to VPSc,l;
13:           Delete the task from MQc,Imap;
14:           Imap++;}}
15:           else{ // the idle slot a reduce slot;//
16:               if RQfifo it is not empty{
17:           Use FIFO to assign the reduce level task from RQfifo to VPSc,l
18:           Delete the task from RQfifo;}
19:           else{
20:           Ired=Ired mod(Nred+1);
21:           To assign the first re level task from MQc,Ired to VPSc,l;
22:           Delete the task from MQc,Ired;
23:           Ired++;}}}
```

Figure 3. The algorithm of task-driven & task assigner (TTA)

Similarly, whenever $VPS!,\ell\ell$ has an idle reduce slot, TTA preferentially assigns a reduce task from $RQ!"!\#$ to $VPS!,\ell\ell$ (see lines 16 to 17). Only when $RQ!"!\#$ is empty, TTA assigns one of the first reduce tasks from other reduce-task queues of $cen!$ to $VPS!,\ell\ell$ (see lines 19 to 22). The above algorithm shows the algorithm of JTA, which in fact is very similar to that of TTA.

The only difference is that JTA always uses the Hadoop FIFO algorithm to assign a map task from each map-task queue (please compare line 11 in two variations) so as to further improve the VPS-locality.

V. SELECTING THE BEST THRESHOLD

5.1 The Small Workload

The following metrics are used to evaluate the performance of the five algorithms under the small workload.

1. Map-data locality, which can be divided into VPS locality rate, Cen-locality rate, and off-cen, Note that the values of the above three rates range from 0 to 1. A value of one is desirable for both the VPS-locality rate and the Cen-locality rate, but a value of zero is desirable for the off-Cen rate.

2. Reduce-data locality rate, which is defined as the percentage of input data that a reducer can obtain from its local datacenter. The value ranges from 0 to 1. A value of one is desirable. **3. Inter-datacenter network traffic (INT for short),** which is the total inter-datacenter network traffic generated during the execution of the workload. A small value of INT is desirable.

4. Job turnaround time (JTT for short), which starts when a job is submitted to the cluster and finishes when the job is completed. A short JTT is desirable.

5. VPS load, which shows the average number of map tasks executed by each VPS and the corresponding standard deviation. With this metric, we can know the load balance among VPSs. A small standard deviation is desirable.

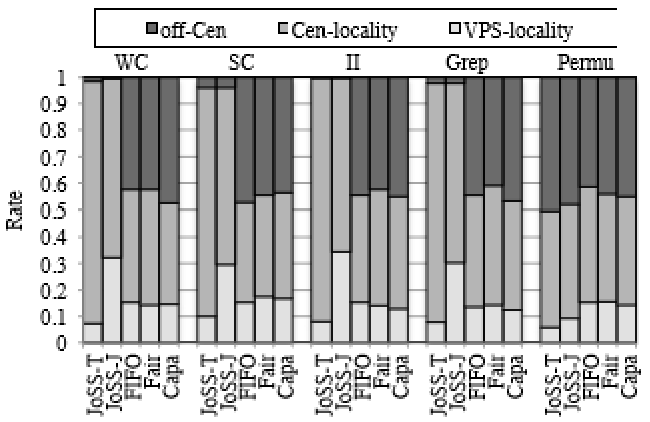


Figure 4. The map-data locality results of the five tested algorithms under the small workload

Even though JoSS-T and JoSS-J had similar off-Cen result, the latter provided a higher VPS-locality rate since it employs the JTA to further increase the VPS locality.

This property also makes the VPS-locality rate of JoSS-J higher than those of the other algorithms when the executed jobs are small MH jobs. The reduce-data locality results of all algorithms. Since JoSS-T and JoSS-J employ the same reduce-task scheduling approach, they have a very similar reduce-data locality rate in every benchmark. In addition, it is clear that JoSS-T and JoSS-J provided a higher reduce data locality rate than the other three algorithms, especially when RH jobs were executed. The reason is the same, i.e., JoSS-T and JoSS-J always use policy A (which favors reduce-data locality) to schedule small RH jobs.

5.2 The Mixed Workload

We evaluated how the five algorithms perform when they execute the mixed workload. Similar to the metrics used earlier, the map-data locality, reduce-data locality, INT, and VPS load were also used to evaluate the five algorithms. However, JTT was not considered in this experiment since the input sizes processed by the jobs in the mixed workload were different, which makes this metric meaningless. Hence, we further used the following metrics to better measure these algorithms:

Workload turnaround time (WTT for short), which is the total time required by the cluster to complete the entire mixed workload.

Cumulative job completion rate during the execution of the mixed workload.

The map-data locality results of all algorithms under the mixed workload. Among all algorithms, JoSS-T caused the lowest VPS-locality rate, regardless of job type. The reason is obvious, i.e., JoSS-T uses TTA to quickly assign a task to an idle VPS, rather than increasing the VPS locality. On the other hand, by comparing, we can see that the VPS-locality rates of the other four algorithms on the mixed workload increased. This is because each VPS held more input blocks of large jobs and therefore improved the VPS-locality rate. This property also causes that JoSS-J was not always better than those of the other three algorithms in terms of VPS-locality. Nevertheless, for all tested MH jobs (i.e., WC, SC, II, and Grep jobs), JoSS-T and JoSS-J had similar off-Cen rates, which were still much lower than those of the other three algorithms.

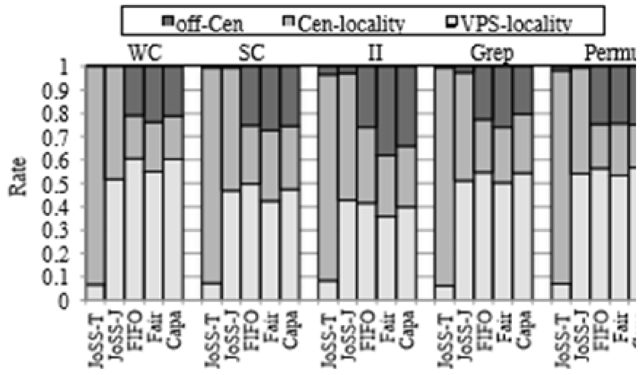


Figure 5. The map-data locality results of the five tested algorithms or the mixed workload

Since JoSS-T and JoSS-J had good data-locality performances, they dramatically reduced the inter-datacenter network traffic for retrieving map-input data and reduce-input data during the execution of the mixed workload.

5.3 Scheduling Overhead

We evaluate the overhead caused by each tested algorithm. The CPU idle rate and memory load of the Hadoop master server when the five algorithms separately executed the mixed workload. It is clear that both JoSS-T and JoSS-J did not significantly increase the CPU and memory load of the master server compared with the other algorithms. In addition, we further evaluated the extra storage space consumed by JoSS-T and JoSS-J to store all necessary information about every newly executed job, including the corresponding hash value and average filtering-percentage value.

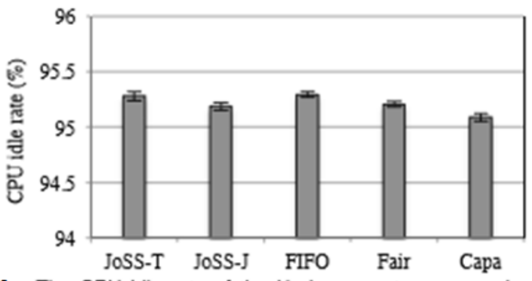


Fig. 6 The CPU idle rate of the Hadoop master server when the five algorithms are individually used to execute the mixed workload.

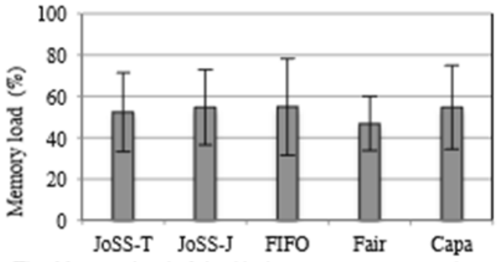
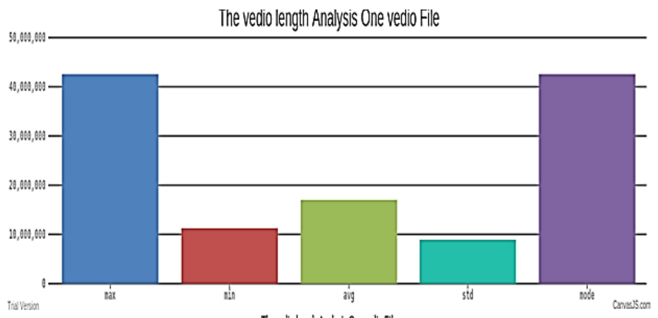


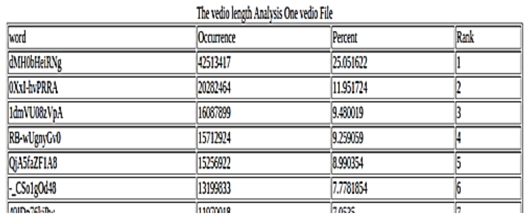
Fig 7 . The Memory load of the Hadoop master server when the five algorithms are individually used to execute the mixed workload.

In our experiments, each such a record is about 20 bytes. Hence, the total storage consumption is proportional to the number of the newly executed jobs. Based on the above analyses, it is clear that JoSS-T and JoSS-J do not incur significant computation overhead memory overhead and storage overhead to the Hadoop master server.

VI. PERFORMANCE EVALUATION AND COMPARISON

We evaluate and compare JoSS-T and JoSS-J with three scheduling algorithms provided by Hadoop, including the FIFO algorithm (FIFO for short), the fair scheduling algorithm (Fair for short), and the capacity algorithm (Capa for short). We established a virtual MapReduce cluster by renting 31 VPSs from Linode [12], which is a privately owned VPS provider based in New Jersey. One VPS acts as the Hadoop master server and is located at a datacenter in Dallas. The remaining 30 VPSs act as slaves. Among them, 15 VPSs are located at a datacenter in Dallas and the other 15 VPSs are located at a datacenter in Atlanta. Each VPS runs Ubuntu 10.04 with two CPU cores, 2 GB RAM, and 48 GB SSD storage space. Each VPS has a map slot and a reduce slot. We use Hadoop MRv1, which is widely adopted in production settings [28], as the implementation of MapReduce, and modify the source code of Hadoop-0.20.2 to realize JoSS-T and JoSS-J. To study how different MapReduce jobs with different filtering-percentage values influence the performances of the five tested algorithms, we chose the following five MapReduce benchmarks to conduct our experiments. The first four jobs are from the MapReduce benchmark suite called PUMA [29], and the corresponding input data are web documents chosen from [30]. The last one job is created by ourselves, and its input data is a set of you tube TXT files.





The vedio length Analysis One vedio File

word	Occurrence	Percent	Rank
dMH0bHeiRNg	42513417	25.051622	1
0Xxl-hvPRRA	20282464	11.951724	2
1dmVU08zVpA	16087899	9.480019	3
RB-wUgnyGv0	15712924	9.259059	4
QjA5faZF1A8	15256922	8.990354	5
-_CSo1gOd48	13199833	7.7781854	6
[illegible]	11070018	7.0535	7

Word-Count (WC for short), which counts the occurrence of each word in data files.

1) Sequence-Count (SC for short), which generates a count of all unique sets of three consecutive words in data files.
2) Inverted-Index (II for short), which takes a list of data files as input and generates word-to-file indexing.
3) Grep, which searches for a pattern in data files.
4) Permu, which generates the permutation for three consecutive DNA sequences in DNA data files. Consequently, not all tested MapReduce benchmarks will be classified as the same job type by JoSS-T and JoSS-J. Some of them will be classified as MH jobs, and the others will be classified as RH jobs.

We used the above five benchmarks to create a small workload and a mixed workload, and used the two workloads to evaluate the performances of the five algorithms.

VII. CONCLUSIONS

In this paper, we have introduced JoSS for scheduling MapReduce jobs in a virtual MapReduce cluster consisting of a set of VPSs rented from a VPS provider. Different from current MapReduce scheduling algorithms, JoSS takes both the map data locality and reduce-data locality of a virtual MapReduce cluster into consideration. JoSS classifies jobs into three job types, i.e., small map-heavy job, small reduce-heavy job, and large job, and introduced appropriate policies to schedule each type of job. In addition, the two variations of JoSS (i.e., JoSS-T and JoSS-J) are further introduced to respectively achieve a fast task assignment an improve the VPS-locality.

The extensive experimental results demonstrate that both JoSS-T and JoSS-J provide a better map-data locality, achieve a higher reduce-data locality, and cause much less inter-datacenter network traffic as compared with current scheduling algorithms employed by Hadoop. The experimental results also show that when the jobs of a MapReduce workload are all small to the underlying virtual MapReduce cluster, employing JoSS-T is more suitable than the other algorithms since JoSS-T provides the shortest job turnaround time. On the other hand, when the jobs of a MapReduce workload are not all small to the virtual MapReduce cluster, adopting JoSS-J is more appropriate because it leads to the shortest workload turnaround time. In addition, the two variations of JoSS have a comparable load balance and do not impose a significant overhead on the Hadoop master server compared with the other algorithms.

VIII. REFERENCES

[1]. Durga solutions by mapreduce " https://www.youtube.com/watchv=6oemzejdmp8 "

[2]. Hadoop, http://hadoop.apache.org (dec. 3, 2014)

[3]. S. Chen and s. Schlosser, "map-reduce meets wider varieties of applications," technical report irp-tr-08-05, intel research, 2008.

[4]. B. White, t. Yeh, j. Lin, and l. Davis, "web-scale computer vision using mapreduce for multimedia data mining," in proceedings of the tenth international workshop on multimedia data mining, pp. 1-10. Acm, july 2010.

[5]. A. Matsunaga, m. Tsugawa, and j. Fortes, "cloudblast: combining mapreduce and virtualization on distributed resources for bioinformatics applications," in ieee fourth

international conference on escience, pp. 222-229, december 2008.

[6]. X-rime. Http://xrime.sourceforge.net/ (dec. 3, 2014)

[7]. K. Wiley, a. Connolly, j. Gardner, s. Krughoff, m. Balazinska, b. Howe, y. Kwon, and y. Bu, "astronomy in the cloud: using mapreduce for image co-addition," astronomy, 123(901), pp. 366-380, 2011.

[8]. Disco, http://discoproject.org (dec. 3, 2014)

[9]. Gridgain, http://www.gridgain.com (dec. 3, 2014)

[10]. David d. Clark, member, ieee, kenneth t. Pogran, member, ieee, and david p. Wed " an introduction to local area networks" https://groups.csail.mit.edu/ana/publications/pubpdfs/an%20introduction%20to%20local%20area%20networks.pdf

[11]. Vidyullatha Pellakuri1 , Dr.D. Rajeswara Rao2" Hadoop Mapreduce Framework in Big Data Analytics " http://ijcttjournal.org/Volume8/number-3/IJCTT-V8P121.pdf

[12]. Abdullah Almurayh" Virtual Private Server" in 2010 http://cs.uccs.edu/~cs526/studentproj/projS2010/aalmuray/doc/Almurayh_VPS.pdf

[13]. " Improving Performance of Heterogeneous MapReduce Clusters with Adaptive Task Tuning" http://ieeexplore.ieee.org/document/7523426/

[14]. " Optimal MapReduce Job Scheduling algorithm across Cloud Federation " http://csce.ucmss.com/books/LFS/CSREA2017/PDP3681.pdf

[15]. Zhenhua Guo, Geoffrey Fox, Mo Zhou " Investigation of Data Locality in MapReduce " https://pdfs.semanticscholar.org/48b5/568d8cec22d167c88d10a4de01f48a4740d0.pdf

[16]. " Self-Adjusting Slot Configurations for Homogeneous and Heterogeneous Hadoop Clusters" http://ieeexplore.ieee.org/document/7065298/

[17]. Z. Guo, G. Fox, and M. Zhou, "Investigation of data locality in mapreduce," In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012), pp. 419-426, May 2012.

[18]. C. He, Y. Lu, and D. Swanson, "Matchmaking: A new mapreduce scheduling technique," In 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom 2011), pp. 40-47, November 2011. [16] T

[19]. T. White, "Hadoop: the definitive guide," O'Reilly Media, Yahoo! Press, June 5, 2009. [

[20]. M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," In Proceedings of the 5th European conference on Computer systems, pp. 265-278. ACM, April 2010, http://dx.doi.org/10.1145/1755913.1755940

[21]. J. Jin, J. Luo, A. Song, F. Dong, and R. Xiong, "BAR: an efficient data locality driven task scheduling algorithm for cloud computing," In 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2011), pp. 295-304, May 2011.

[22]. "Hadoop MapReduce Scheduling Algorithms - A Survey" http://www.ijcsmc.com/docs/papers/December2015/V4I12201548.pdf

[23]. Fair Scheduler Guide, http://archive.cloudera.com/cdh/3/hadoop0.20.2+737/fair_scheduler.html (Dec. 3, 2014)

[24]. Capacity Scheduler Guide, http://archive.cloudera.com/cdh/3/hadoop0.20.2+737/capacity_scheduler.html (Dec. 3, 2014)

[25]. "https://www.youtube.com/watchv=AcUauzCn7RE" youtube API extract data from youtube.