Job-aware Network Scheduling for Hadoop Cluster

Wen Liu[1], Zhigang Wang[1], and Yanming Shen[1]
[1] School of Computer Science and Technology, Dalian University of Technology,
Dalian, Liaoning, P. R. China, 116024
[e-mail: 627952@qq.com; 475920112@qq.com; shen@dlut.edu.cn]
*Corresponding author: Yanming Shen

Received July 4, 2016; revised September 26, 2016; accepted November 22, 2016;
published January 31, 2017

Abstract

In recent years, data centers have become the core infrastructure to deal with big data processing. For these big data applications, network transmission has become one of the most important factors affecting the performance. In order to improve network utilization and reduce job completion time, in this paper, by real-time monitoring from the application layer, we propose job-aware priority scheduling. Our approach takes the correlations of flows in the same job into account, and flows in the same job are assigned the same priority. Therefore, we expect that flows in the same job finish their transmissions at about the same time, avoiding lagging flows. To achieve load balancing, two approaches (*Flow-based* and *Spray*) using *ECMP* (*Equal-Cost multi-path routing*) are presented. We implemented our scheme using NS-2 simulator. In our evaluations, we emulate real network environment by setting background traffic, scheduling delay and link failures. The experimental results show that our approach can enhance the Hadoop job execution efficiency of the shuffle stage, significantly reduce the network transmission time of the highest priority job.

Keywords: Cloud computing, job-aware, priority scheduling, Hadoop, load balancing

Part of the results in this paper appeared in the Proceedings of the 2015 International Conference on Cloud Computing and Big Data (CCBD). This work is supported by the National Natural Science Foundation of China under Grant No. 61173160, and Scientific Research Program of the Higher Education Institution of Xinjiang under Grant No. XJEDU2016I049.

https://doi.org/10.3837/tiis.2017.01.012 **ISSN : 1976-7277**

1. Introduction

With the development of Internet, data has penetrated into every field. In order to deal with massive data, researchers use cluster of machines to parallelly process massive data. Some significant research results are introduced, such as MapReduce [1-3], GFS [4], BigTable [5] etc.

The large scale distributed computing applications will generate large amounts of data transmission in different processing stages. Monitoring data of the Facebook cluster [6] shows that the network transmission time has exceeded 50% of the total required time, and network transmission is becoming the key factor of limiting application performance. In order to improve network efficiency, researchers focus on flow-based network scheduling strategy.

To schedule flows efficiently, first it is necessary to know the application traffic demands[7,8]. Some people leverage the network-level parameters from switches and hosts to obtain that information. For example, Hedera [7], Helios [8] estimate traffic demand based on flow counters on switches. Mahout [9] uses socket buffer usage at end hosts to predict network requirement. However, these approaches can get the network-level information only after a flow starts. Also, due to congestion control and background traffic, flow parameters observed cannot accurately reflect the actual traffic demand. Furthermore, these approaches can only obtain individual flow information without priorities and dependency relationship. In parallel computing, flows are not independent, and are correlated, and information obtained from network layer cannot reflect the relationship among flows and their priorities.

To address these issues, we propose to obtain flow information from application layer. In this way, it can not only ensure the information accuracy, but also can be well combined with the centralized control of the big data application framework. In this paper, we propose the job-aware scheduling approach, and the corresponding method of obtaining flow information from application layer for the shuffle stage in Hadoop. To obtain flow information from Hadoop application layer, we can get flow information in advance, and accurately determine the flow start-end time, and also obtain the relationship among flows. In scheduling, we consider the correlations among flows and propose the job-based priority scheduling.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 presents our flow forecasting method from application layer. Section 4 proposes the network priority scheduling approach. Experiment results and analysis are shown in Section 5. In Section 6 we conclude the paper.

2. Related Work

Alizadeh et.al proposed DCTCP based on the TCP protocol [10], which adds identification field in a packet, and uses the explicit congestion notification (ECN) to quickly notify the sender of the network congestion. Dogar et.al proposed Smart Priority Class in [11], to forecast the network resources requirement when sending a flow. Mahout [9] observed the size of the TCP socket buffer, and determined the size of a TCP flow by setting a threshold. Hedera [7] collected the flow information by observing the occupied bandwidths at a switch.

Some works propose to obtain flow information from the application layer. Orchestra [6] makes the running application actively send the network demand to the scheduling controller by modifying the Hadoop application framework. FlowComb [12] analyzed the map task nodes from Hadoop JobTracker log, and then obtain flow information between map nodes and reduce nodes via web API provided by Hadoop. HadoopWatch [13], by monitoring log files, proposed to obtain flow information before a flow starts. Coflow [14,15] established a logical application layer of the data plane, obtained the demand of the parallel computing model (e.g., [16]), and satisfied this requirement by appropriate scheduling.

To schedule flows, DCTCP [10] determines whether the network is congested from Explicit Congestion Notification (ECN), and then controls the TCP sending rate. Based on DCTCP, D2TCP [17] proposed a deadline-aware congestion avoidance algorithm, which combines the deadline information and the network congestion information to control the TCP transmission rate. PDQ (Preemptive Distributed Quick) [18] uses distributed scheduling approach to complete flow transmission faster, and more flows can satisfy the deadline requirements. These approaches can be applied in small scale networks, and become more complicated in multi-path scenario.

Based on TCP protocol, MP-TCP (multi-path TCP) [19] proposed a multi-path TCP, which divides a TCP connection into multiple TCP connections, and then schedules flows by ECMP protocol (Equal-Cost multi-path routing). In ECMP, the Hash-based path selection may assign multiple flows to the same path, resulting in severe congestion. To address this issue, Advait Dixit et al. proposed the RPS [20] (random packet spraying) packet scheduling approach to achieve a better load-balancing. Hedera uses the Global First Fit algorithm to assign a path satisfying a flow’s bandwidth requirements from all possible multi-paths. Helios [8] and FlowComb [12] use the same approach proposed by Hedera to schedule flows. MicroTE [21] proposed to obtain flow information in real time by OpenFlow [22] protocol, and then doesn’t allocate flows to heavy loaded links to achieve better load-balancing.

Some researchers consider the flow scheduling problem by considering the flow correlation. Orchestra [6] used the centralized scheduling approach, similar to BitTorrent protocol, to broadcast data to their destination. For correlated flows (such as flows of Hadoop shuffle phase), they are assigned a weight and the network resources are allocated according to the weight. This will make the correlated flows be completed at the same time as much as possible. Based on flow characteristics of Bing [10,23,24] and FaceBook [6,25], Barrat [20] considers the task-based flow scheduling.

3. Application Layer Flow Forecasting

3.1 Overview

Distributed computing systems, such as Hadoop, will generate a lot of files during the execution of a job. These files include source files, temporary files, result files and log files. By real-time monitoring the files operation and extracting the key information, we can obtain the flow information generated by the Hadoop operation. Specifically, the Hadoop job log and index file will be stored in a temporary file after the map task ends. From these files, we can obtain information of shuffle phase flows from this map task. If a reduce task needs to read the data from a certain map task, the network transmission parameters will be recorded in the map task log file. Flow information can be obtained by monitoring and analyzing the temporary files and log files of the nodes in the cluster.

Fig. 1 is our architecture of the flow forecasting system. Through monitoring and analyzing the log files, the architecture can forecast network demand of the job in realtime, and transmit the forecasting information to a central node. Based on the flow information collected by the central node, we can do a global scheduling in the network. The flow forecasting system has two core components, the forecasting agent and the centralized controller. In the cluster, the forecasting agent is deployed on each computing node, and the collected information is sent to the centralized controller.

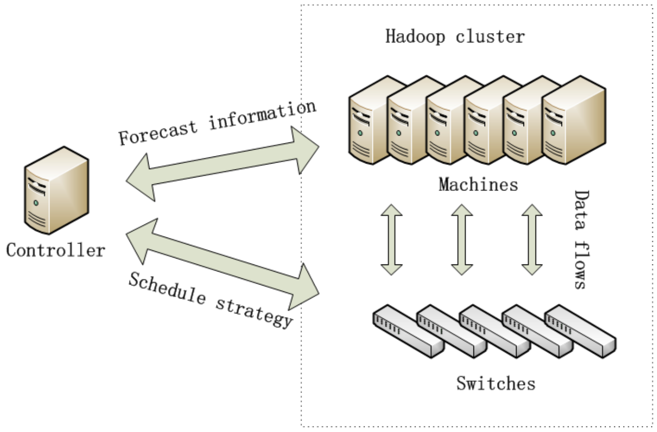


Fig. 1. The architecture of the forecasting system

The forecasting agent: In this paper, we use the file change notification mechanism of the Linux kernel, *Inotify*. When running Hadoop, we monitor the temporary files and log files, extract flow information and send it to the centralized controller in realtime. Because *Inotify* can only monitor the specified directory and cannot monitor its subdirectory, in this paper, we design an approach to monitor the subdirectory according to the structure of Hadoop working directory and the log directory.

The centralized controller: The controller's main function is to collect the forecasting information from all the forecasting agents, and store the forecasting information. The collected forecasting information includes the flow source/destination addresses, the flow size, the start-stop sign of the flow, and the dependencies of flows. By utilizing these information, we can better schedule flows.

3.2 The storage of flow information

In running Hadoop, the structure of the working directory and the log directory are shown in Fig. 2 and Fig. 3.

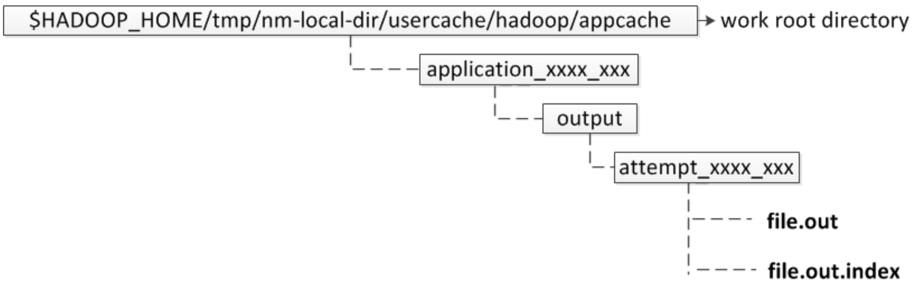


Fig. 2. Hadoop working directory

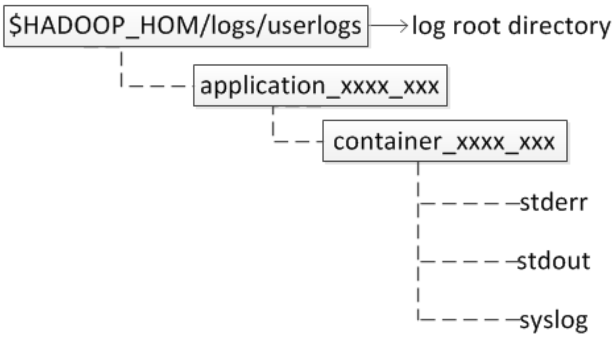


Fig. 3. Hadoop log directory

In the directory name, such as “application_XXXX_XXX”, "XXXX_XXX" represents numbers, where the first part is the job label *JobId* assigned by the Hadoop central control procedure. The *JobId* is globally unique and can be used to distinguish different jobs. Therefore, it can be used to determine dependencies among the flows from jobs. The second part of the directory name is the task ID of this job, and task ID is unique within each job. In the forecasting program, the task ID can be mapped to a flow’s destination address of a reduce task.

The output file of the Hadoop job in the map phase is temporarily stored on the local disk of the corresponding node, where the temporary file "file.out.index" stores the sizes information of flows to be transmitted to each reduce task from this map task. When the file "file.out.index" is created, we can obtain the information of flows from this map task to other reduce tasks. In the shuffle phase, reduce task reads the data from the map task node. For efficiency, each reduce task reads data from at most 5 map nodes at the same time. Therefore, from the "file.out.index" file, we can only obtain the reduce label, and cannot obtain the address of the node running a reduce task. From the Hadoop operational logs, we can obtain the mapping between the reduce task label and its address, and therefore determine the destination address for a flow. The starting and ending time of a flow between a map node and a reduce node can also be obtained by analyzing the corresponding log files.

3.3 Forecasting agent

The forecasting agent is deployed on each processing node in the cluster, running forecasting program, connecting to the central controller, and sending the flow information to the centralized controller in realtime.

The structure of the working directory is shown in **Fig. 2**. We need to monitor the event of the subdirectory creation in the root directory, and also the event of the subdirectory creation in the created subdirectory. For example, when the directory "attempt_xxxx_xxx" is created, by analyzing the name of the directory, we can obtain *JobId* of the job and the task labels. Then, we need to monitor the creation of the file "file.out.index" in the directory of "attempt_xxxx_xxx". This file records the size of the data volume transferred between a map node and a reduce node. When this file is created, according to the file format, we analyze the file, obtain the corresponding values, and send these information together with *JobId* and the node’s address to the controller.

For the log subdirectory, with the same approach, we can monitor the directory of "container_xxxx_xxx" and the modification events of the "syslog" log. For "container_xxxx_xxx" directory, we analyze the name of the directory, and get the task label and job *JobId* which this log belongs to. For "syslog" modification event, based on the keyword, we can obtain the category of this log. If it is a reduce log, the reduce task *JobId*, the task label and the node's address need to be sent to the controller. If it is a map log, we will continue to monitor the modification events of this log, and record the position (*lastItem*) of the last analyzed item of the log. When a modification event is detected, we analyze the content between the last analyzed location and the end of log, and update *lastItem*. If there are log events associated with staring a reduce task, we extract the label of the reduce task, and set the current time as the starting time of the flow between this map node and the reduce node. If there are log events indicating the finish of transmission of a reduce task, we extract the reduce label, and set the current time as the stopping time of the flow. If a log event indicates that this map task has finished all data transmission, we will stop the monitoring for this map log. Everytime detecting a new flow, we will transmit the information of the map task *JobId*, the map task label, map node's address, the start and stop time of the flow, and the reduce task label to the collection node.

In summary, forecasting agents are able to acquire a flow's source address, destination address, the flow size, the flow start and end sign and the job that the flow belongs to. The forecasting information of these network flows is divided into three types of information, as shown in **Table. 1**.

Table 1. The type of information for traffic forecasting

type	source	explanation
1	file.out.index	Source address, size, and reduce task label of a flow
2	reduce log	Reduce task label and the destination address of the reduce node
3	map log	Start and stop time of a flow between a map node and a reduce node

3.4 Network forecasting information collection

The centralized controller is in charge of collecting forecasting information from all forecasting agents, organizing and storing these information according to the source address and destination address of flows.

In our approach, the forecasting agent can only obtain type 1 information in Table 1, i.e., the source address, size and the reduce task label of a flow. The destination address can be obtained by analyzing the reduce logs, which record the mappings between reduce task label and the reduce node's address.

The information collected about a flow also includes which job this flow belongs to. Usually, flows in the same job are correlated. Therefore, we can perform job-aware flow scheduling, where flows in the same job are scheduled together. In this way, we can avoid lagging tasks and reduce the job completion time.

4. Job-based Priority Scheduling

4.1 Overview

In the Hadoop cluster, each reduce task needs to obtain the intermediate results from the map task, which is called the shuffle phase. Because the map and reduce tasks may be assigned to different nodes, intermediate results at map nodes need to be transferred to the reduce node through the network, which will produce a large number of flows. Also, a reduce task can proceed to the next step only after obtaining data from all map tasks. When all reduces are done, the final job results are obtained. If we don't schedule these flows carefully, it is possible that some flows may lag behind and extend the overall job completion time.

Priority scheduling is to assign a priority to each flow. For a high priority flow, it should be allocated more network resources, so that the high priority network flow can be completed earlier. As shown in Fig. 4, on the same link, there are two flows A and B, sending the same amount of data. If fair scheduling is used to allocate the network resources, A and B will complete the transmission at the same time, which is 8, as shown in Fig. 4 1).

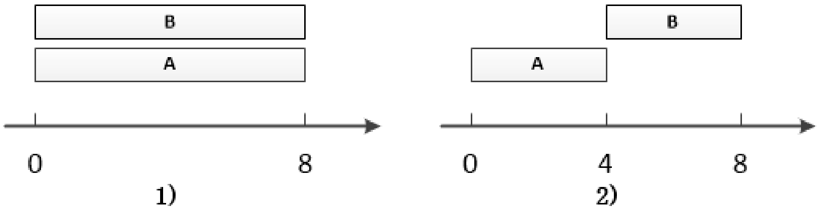


Fig. 4. The example for priority scheduling

If the priority scheduling is used, all network resources are allocated to A first. When flow A is completed, flow B occupies all network resources, as shown in Fig. 4 2). At this time, the completion time of flow A is 4, the completion time of flow B is 8. The average completion time reduces to 6. It can be seen that the priority scheduling approach can reduce the average flow completion time. If flow A has deadline requirements, fair scheduling may not be able to satisfy the requirement.

In this paper, we define the priorities of jobs based on their time of arrivals. This means that an earlier arrived job has a higher priority. Flows in the same job have the same priority. After assigning a priority for a flow, it is important to appropriately allocate network resources to flows with different priorities. Next, we introduce how we do the path selection for a flow and queue management at a switching node.

4.2 Fat-Tree topology and path management

Fat-Tree [26,27] topology is a symmetrical multiple stage interconnection with equal division bandwidth, good scalability, and abundant equal-cost paths between two nodes. In order to make full use of the symmetry and multi-path characteristics of Fat-Tree topology, and achieve better load-balancing, we propose two approaches to use the multi-paths between two nodes. One approach is packet-based multi-path load balancing, named *Spray*; the other one is the flow-based load-balancing, named *Flow-based*.

4.2.1 Spray

Spray means packet-based load balancing, where packets in a flow are evenly distributed among all equal cost paths. In this way, each path will receive roughly the same traffic, reducing retransmissions from out-of-sequence packets.

Since Fat-Tree topology is symmetric, in order to ensure uniform distribution of the packets to every path, we adopt destination address based round-robin scheduling approach, where packets having the same destination address are assigned to different equal-cost links. As shown in Fig. 5, to transmit data between node S1 and S2, the packets will be scheduled in the round-robin manner according to their destination addresses. In this example, packets transferred between these two nodes are uniformly assigned to all four equal-cost paths. In order to achieve per-destination round-robin scheduling, we set up a counter at the switching node of the edge layer and agg layer.

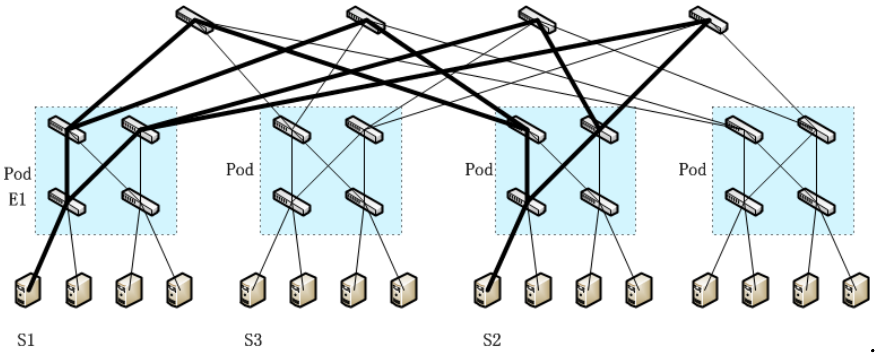


Fig. 5. per-destination round-robin packet *Spray*

However, if we do per-packet round-robin, it may result in different loadings on links. For example, in **Fig. 5**, if node S1 transfers data to node S2 and S3 at the same time, it is possible that packets to node S2 and S3 may arrive interleaved at switching node E1, and flows to S2 or S3 can only utilize two paths among all four paths.

4.2.2 Flow-based path management approach

Different from the *Spray* approach, the *Flow-based* approach assigns each flow to the equal-cost links, and tried to make sure that the utilization on each link is roughly the same. Note that the centralized controller has the information of all flows in shuffle phase, and this provides the possibility of a centralized scheduling. With centralized scheduling, we can choose a suitable path for a flow based on the current flows in the network and link utilization, and therefore avoid assigning big flows to the same link.

Because the path allocation and the flows are dynamic, when a new flow is added, the previous optimal allocation may not be optimal for the current time. If we do dynamic optimal allocation, it will lead to frequent path changes for a flow. This will result in out-of-sequence problem and reduce the throughput. Therefore, in this paper, we don't change the path of a flow once it is allocated.

According to the characteristics of Fat-Tree topology, there are equal-cost multi-paths in the upward path of the switching node in the edge layer and the agg layer. To achieve a better load-balancing, we record the usage of the upward link in each switching node at the edge layer and the agg layer, i.e., the number of flows assigned to each link. When the forecast program detects a new flow, it first obtains the on-path switching node *edgeSw* at the edge layer. Then it determines whether the packets need to be sent to the agg layer. If required, it selects the least loaded link from all equal-cost upward links, distributed flow to this link and obtains the switching node *aggSw* at the agg layer. The same operation is performed at the *aggSw* node. **Fig. 6** shows the flow chart of flow-based path allocation.

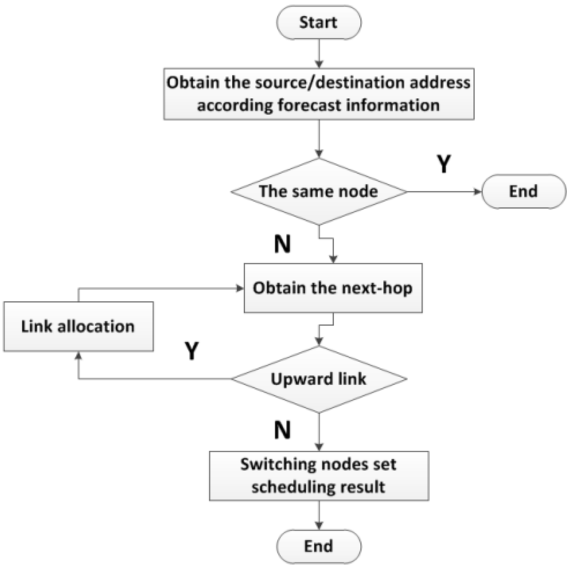

Start
Obtain the source/destination address according forecast information
The same node
Y
End
N
Obtain the next-hop
Link allocation
Y
Upward link
N
Switching nodes set scheduling result
End

Fig. 6. The flow chart of *Flow-based* path allocation

4.3 Queue management

With our proposed priority scheduling, when selecting a packet to send, high priority packet will be selected first. Also, when the queue is full, to accept new high priority packets, we need to drop low priority packets. In this way, we can quickly finish the transmission of high priority packets.

To avoid head of line blocking by low priority packets, we divide a physical queue into multiple logical queues, with each logical queue corresponding to a different priority. An arriving packet will be put at the tail of its corresponding queue. When sending a packet, among all non-empty queues, packet in the highest priority queue will be selected. When the buffer is full, packets in the lowest priority queue will be dropped.

4.4 Fault tolerance

In Fat-Tree, the link failure between the processing node and the edge switch can only be recovered by replacing the corresponding equipment. Therefore, we only discuss the link failure between the agg layer and the core layer, and between the edge layer and the agg layer. When a link fails, flows on this link need to be re-routed. Also, we should avoid assigning new flows to this link. Next, we discuss agg-core link failures and edge-agg link failures respectively.

4.4.1 The agg-core link failure

When the link <*lSrc*-*lDst*> (*lSrc* means the node in the agg layer, and *lDst* in the core layer) between the agg layer and the core layer fails, the *lSrc* node knows the pod tag of the failed link, denoted as pod (*lSrc*). In the *lSrc*-*lDst* direction, the affected flows are those assigned on link <*lSrc*-*lDst*> by the *lSrc* node. Because there are multi-paths on the upward link, we can reassign existing flows on the failed link to other upward links. Also, new flows needs to avoid being allocated to the link <*lSrc*-*lDst*>. In the *lDst*-*lSrc* direction, the flows on the failed link are transmitted from other pods to the processing node within the pod (*lSrc*). Since the downward link is unique, the *lDst* node cannot reallocate flows to other paths. However, according to the Fat-Tree topology, there is a core switch between any two nodes in the agg layer, and the flows can be rerouted by other core switches.

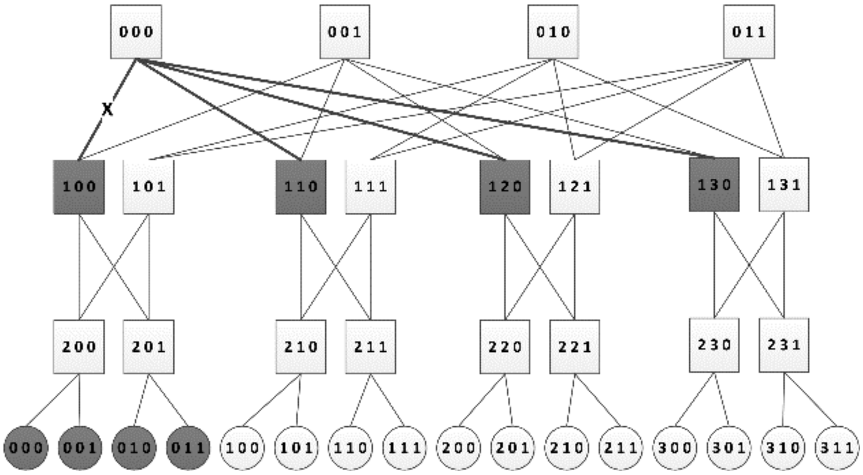


Fig. 7. The agg-core link failure

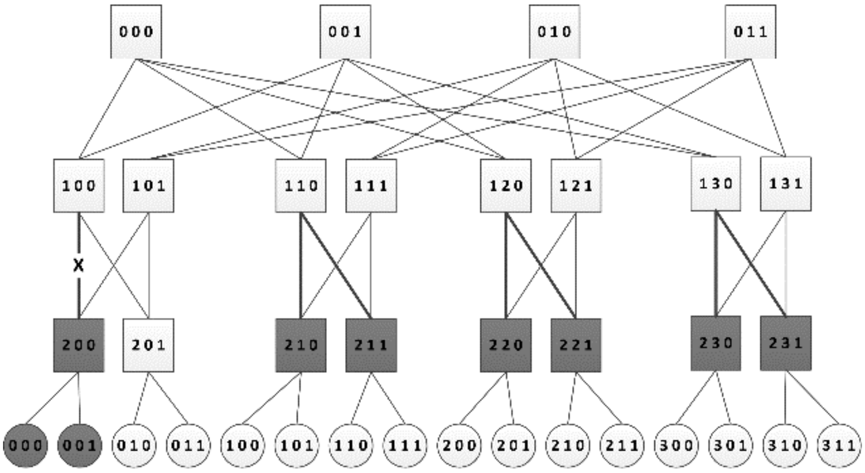


Fig. 8. The edge-agg link failure

As shown in **Fig. 7**, when the link between the agg layer (node 100) and the core layer (node 000) fails, the affected links are thick solid lines, the affected processing nodes and switching nodes are shaded.

4.4.2 The edge-agg link failure

As shown in **Fig. 8**, when the link between the edge layer and the agg layer is disconnected, the affected links are marked with thick solid lines, the affected processing nodes and switching nodes are shaded. Similarly, when the link <*lSrc*-*lDst*> (*lSrc* means the node in the edge layer, and *lDst* in the agg layer) fails, at the *lSrc* node, all upward flows assigned to the failed link need to be reallocated, and new flows on the *lSrc* node need to avoid being allocated to the link <*lSrc*-*lDst*>. In the *lDst*-*lSrc* direction, since the downward link is unique, the *lDst* node cannot reallocate flows. The flows need to be rerouted by source nodes in other pods.

5. Experiment

5.1 Experimental setup

In our experiments, we simulate flows in Hadoop shuffle phase using NS-2. With the same simulation setting, we compare our job-aware priority scheduling with the non-priority scheduling. In the network topology, each link bandwidth is set to 100Mb/s, the buffer capacity at a port is set to 100 packets, and the flow size is 20MB. In each experiment, we vary the number of jobs, the number of map tasks and reduce tasks, and the way of spreading packets, the results are averaged over 10 runs. In the experiment, the number of jobs is varied from 2 to 8. In the case of priority scheduling, each job is assigned a different priority. For a job, we change the number of map tasks and reduce tasks within this job, and they are set as: 6 map tasks - 2 reduce tasks, 8 map tasks - 3 reduce tasks, and 10 map tasks - 4 reduce tasks. For each setting, the nodes running map tasks and reduce tasks are the same. Therefore, flows generated in each setting are the same. To emulate the real network scenario, in our experiments, we also consider background traffic, the delay of transmitting flow information to a switch and link failures.

5.2 Experimental results and analysis

5.2.1 Experimental results

In the Fat-tree network topology, there are multiple shortest paths between two nodes. To schedule flows over multiple paths, we use the *Spray* and *Flow-based* approaches presented in Section 4.2. **Fig. 9** and **Fig. 10** show the average job completion time with *Spray* and *Flow-based* respectively. We can see with each different simulation setting, compared with the non-priority scheduling, our job-aware priority scheduling can reduce the average job completion time.

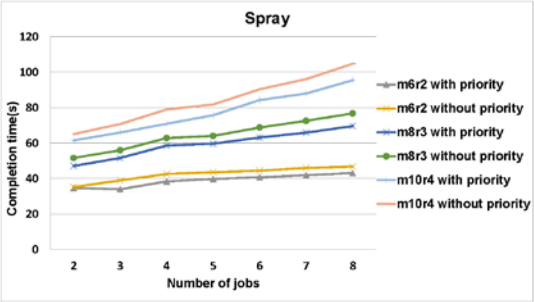


Fig. 9. Average completion time (*Spray-based*)

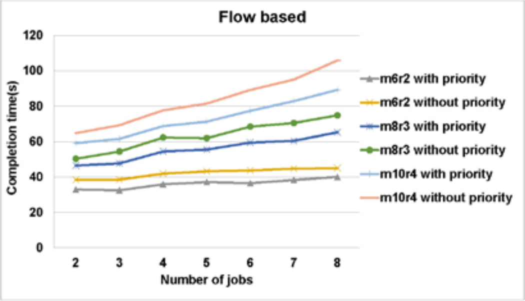
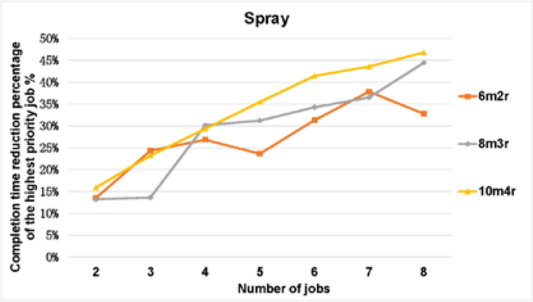


Fig. 10. Average completion time (*Flow-based*)

Our proposed job-aware priority scheduling gives a higher priority for higher priority jobs, and therefore, the completion time for higher priority jobs will be reduced. **Fig. 11** and **Fig. 12** verify this conclusion. From the figures, we see that our job-aware priority scheduling can reduce the highest priority job completion time. When there are more jobs (means the network is more congested), with job-aware priority scheduling, the highest priority flows can obtain more network resources to finish their transmission, and therefore we can achieve a higher reduction ratio.



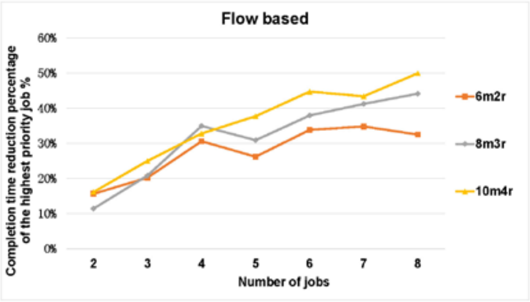
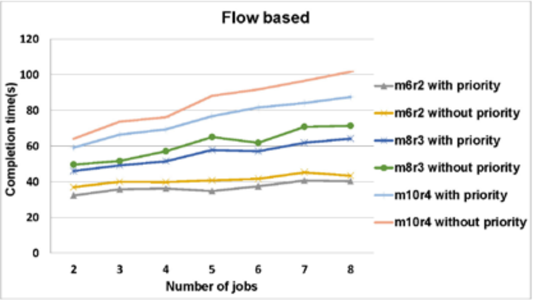


Fig. 11. Completion time reduction ratio for the highest priority job (*Spray-based*)

Fig. 12. Completion time reduction ratio for the highest priority job (*Flow-based*)

5.2.2 Experiment results under background traffic

For background traffic, we use the Poisson distribution, Normal distribution, and Exponential distribution to simulate the background traffic in the real network. Under different background traffic distributions, the results have the same trend. Fig. 13 shows the result of using the Poisson distribution for background traffic and *Flow-based* scheduling to spread packets. The figure shows that job-aware priority scheduling can reduce the average job completion time for background traffic scenario.



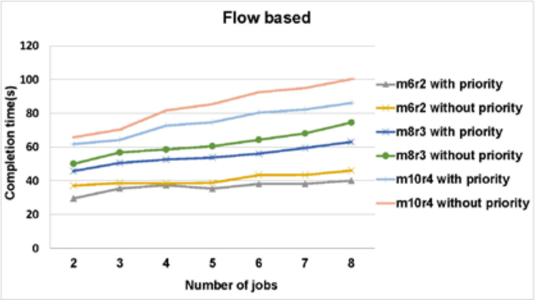


Fig. 13. Average completion time (Poisson)

Fig. 14. Average completion time (unknown flow)

5.2.3 Experiment results with delay

For our job-aware priority scheduling, the flow information is collected at executing nodes and then transmitted to a centralized controller. There is a non-negligible delay in this process. Therefore, in this section, we evaluate the performance with this delay. We model the delay with a uniform distribution in [10ms, 500ms]. Under the delay scenario, it is possible that a flow's priority is unknown when we schedule it, and we need to assign a priority to this unknown flow. In our simulation, we investigate three approaches for assigning the priority, that is, the highest priority, the lowest priority, and the middle priority.

Fig. 14 shows the result where with *Flow-based* packet spreading, unknown priority flows are assigned the highest priority and the middle priority. We can see that our job-aware priority scheduling can still reduce the average job completion time, and especially reduce the completion time of the highest priority job. But if the unknown packets are assigned the lowest priority, although it can reduce the completion time of the highest priority job, the average overall completion time may increase. As shown in Fig. 15, with *Flow-based* path management, when the number of flows is small, the average job completion time is slightly higher than non-priority scheduling. With the *Spray* path management, as shown in **Fig. 16**, only under the case of 10 map tasks and 4 reduce tasks, our job-aware priority scheduling can reduce the average completion time. The reason is that if we assign the lowest priority to the unknown packets, this will lead to packet retransmissions and increase the job completion time. Therefore, if there is delay of obtaining the flow priority information, we should assign the unknown flows a higher priority.

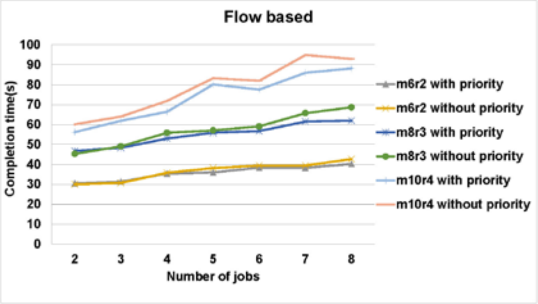


Fig. 15. The average job completion time when assigning the lowest priority to unknown flows (*Flow-based*)

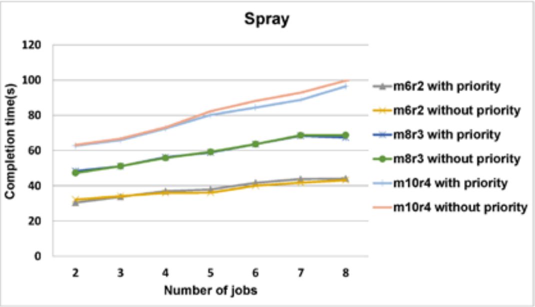


Fig. 16. The average job completion time when assigning the lowest priority to unknown flows (*Spray-based*)

5.2.4 Experiment results under link failure

In this section, we evaluate the performance under link failures. In the experiment, after 10 seconds, we fail a link, and the link is off throughout the experiment.

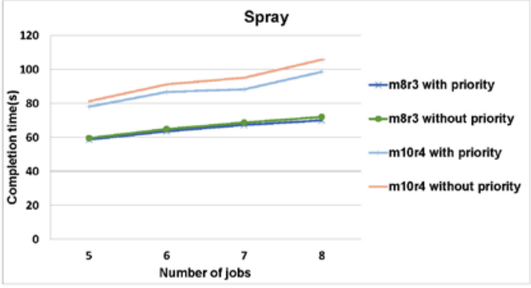


Fig. 17. Average job completion time for edge-agg link failure (*Spray-based*)

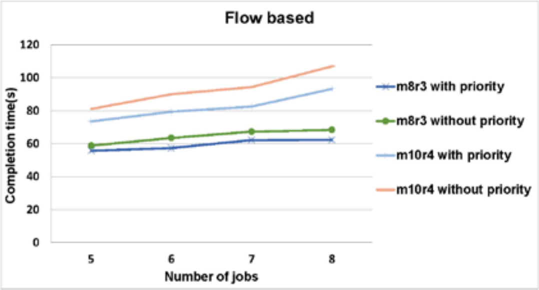


Fig. 18. Average job completion time for edge-agg link failure (*Flow-based*)

Note that for link failures, there are two different kinds of links, agg-core links, and edge-agg links. Experimental results show the same trend for these two kinds of link failures, and therefore, we only show the results for edge-agg link failure. **Fig. 17** and **Fig. 18** show the case of edge-agg link failure. From the figure, we can see that dynamic fault-tolerance approach can correctly reassign flows on the failed link, and our job-aware priority scheduling can reduce the average job completion time. But for *Spray* path management, if the number of flows is few, the performance gain is not significant. This is because for *Spray* path management, it requires a similar cost for multi-paths, to avoid out-of-sequence problem. Therefore, when a link fails, the Fat-Tree topology is no longer symmetric, leading to different bandwidths and delays for these originally equal-cost multi-paths. This will result in packet retransmission, weakening the gain of the job-aware priority scheduling.

6. Conclusions

In this paper, we study the flow scheduling problem in the shuffle stage of a Hadoop job. First, based on the monitoring and analysis of Hadoop temporary files and log files, we obtain the flow information and their dependence in the shuffle phase. Then we send these information to a centralized controller to obtain a global view of flows in the shuffle phase. Based on these collected information, we propose the job based priority scheduling, where flows belonging to the same job have the same priority. Under the Fat-Tree topology, we present two approaches, *Flow-based* and *Spray*, which make use of the equal-cost multi-path, to achieve the network load balancing. In the experimental settings, background traffic, scheduling delay and link failure are introduced to simulate the real environment. The results show that the job-aware priority scheduling approach can reduce the average job completion time in shuffle phase, and can significantly reduce the highest priority job completion time.

References

[1] Dean J, Ghemawat S, “MapReduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, January, 2008. Article (CrossRef Link)

[2] Morton K, Balazinska M, Grossman D, “ParaTimer: a progress indicator for MapReduce DAGs,” in *Proc. of 29th ACM SIGMOD International Conference on Management of data*, pp.507-518, June 6-11, 2010. Article (CrossRef Link)

[3] Ferreira Cordeiro R L, Traina Junior C, Machado Traina A J, et al, “Clustering very large multi-dimensional datasets with mapreduce,” in *Proc of 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp.690-698, August 21-24, 2011. Article (CrossRef Link)

[4] Ghemawat, S., H. Gobioff, and S. Leung, ”File and storage systems: The Google File System,” in *Proc. of 19th ACM Symposium on Operating Systems Principles Bolton Landing*, pp.29-43, October 19-22, 2003. Article (CrossRef Link)

[5] Chang F, Dean J, Ghemawat S, et al, “Bigtable: a distributed storage system for structured data,” *Acm Transactions on Computer Systems*, vol. 26, no. 2, pp. 205-218, June, 2008. Article (CrossRef Link)

[6] Chowdhury M, Zaharia M, Ma J, et al, “Managing data transfers in computer clusters with orchestra,” *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 98-109, August, 2011. Article (CrossRef Link)

[7] Al-Fares, Mohammad, et al, “Hedera: dynamic flow scheduling for data center networks,” in *Proc. of 7th USENIX Symposium on Networked Systems Design and Implementation*, pp. 281-296, April 28-30, 2010. Article (CrossRef Link)

[8] Farrington N, Porter G, Radhakrishnan S, et al, “Helios: a hybrid electrical/optical switch architecture for modular data centers,” *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 339-350, August, 2011. Article (CrossRef Link)

[9] Curtis A R, Kim W, Yalagandula P, “Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection,” in *Proc. of 30th IEEE International Conference on Computer Communications*, pp.1629-1637, April 11-15, 2011. Article (CrossRef Link)

[10] Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., & Prabhakar, B., et al, “Data center tcp (DCTCP),” *ACM Sigcomm Computer Communication Review*, vol. 40, no. 4, pp. 63-74, August, 2011. Article (CrossRef Link)

[11] Dogar F R, Karagiannis T, Ballani H, et al, “Decentralized task-aware scheduling for data center networks,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 431-442, August, 2014. Article (CrossRef Link)

[12] Das, A., Lumezanu, C., Zhang, Y., Singh, V., Jiang, G., & Yu, C, "Transparent and flexible network management for big data processing in the cloud," in *Proc. of 5th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'13*, pp. 37-52, June 25-26, 2013. Article (CrossRef Link)

[13] Peng Y, Chen K, Wang G, et al, "Hadoopwatch: A first step towards comprehensive traffic forecasting in cloud computing," in *Proc. of 33th IEEE International Conference on Computer Communications*, pp. 19-27, April 27-May 2, 2014. Article (CrossRef Link)

[14] Chowdhury M, Stoica I, Chowdhury M, et al, "Coflow: An Application Layer Abstraction for Cluster Networking," *ACM Hotnets*, pp. 1-6, August 7, 2012. Article (CrossRef Link)

[15] Chowdhury M, Stoica I, " Coflow: A networking abstraction for cluster applications," in *Proc. of 11th ACM Workshop on Hot Topics in Networks*, pp. 31-36, October 29-30, 2012. Article (CrossRef Link)

[16] Isard M, Budiu M, Yu Y, et al, "Dryad: distributed data-parallel programs from sequential building blocks," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, pp. 59-72, March, 2007. Article (CrossRef Link)

[17] Vamanan B, Hasan J, Vijaykumar T N, "Deadline-aware datacenter tcp (D2TCP)," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 115-126, August, 2012. Article (CrossRef Link)

[18] Hong C Y, Caesar M, Godfrey P, "Finishing flows quickly with preemptive scheduling," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 127-138, August, 2012. Article (CrossRef Link)

[19] Ford A, Raiciu C, Handley M, et al, "TCP Extensions for Multi-path Operation with Multiple Addresses: draft-ietf-mptcp-multiaddressed-03," *Roke Manor*, March 2011. Article (CrossRef Link)

[20] Dixit A, Prakash P, Hu Y C, et al, "On the impact of packet spraying in data center networks," in *Proc. of 32nd IEEE International Conference on Computer Communications*, pp. 2130-2138, April 14-19, 2013. Article (CrossRef Link)

[21] Benson T, Anand A, Akella A, et al, "MicroTE: Fine grained traffic engineering for data centers," in *Proc. of 17th International Conference on Emerging Networking Experiments and Technologies*, pp. 1-12, December 6-9, 2011. Article (CrossRef Link)

[22] McKeown N, Anderson T, Balakrishnan H, et al, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, April, 2012. Article (CrossRef Link)

[23] Jalaparti V, Bodik P, Kandula S, et al, "Speeding up distributed request-response workflows," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 219-230, August, 2013. Article (CrossRef Link)

[24] Ananthanarayanan G, Kandula S, Greenberg A G, et al, "Reining in the Outliers in Map-Reduce Clusters using Mantri," in *Proc. of 9th USENIX Symposium on Operating Systems Design and Implementation* , pp. 24-31, October 4-6, 2010. Article (CrossRef Link)

[25] Nishtala R, Fugal H, Grimm S, et al, "Scaling memcache at facebook," in *Proc. of 10th USENIX Symposium on Networked Systems Design and Implementation*, pp. 385-398, April 2-5, 2013. Article (CrossRef Link)

[26] Al-Fares M, Loukissas A, Vahdat A, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 63-74, August, 2008. Article (CrossRef Link)

[27] Niranjan Mysore R, Pamboris A, Farrington N, et al, "Portland: a scalable fault-tolerant layer 2 data center network fabric," A*CM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 39-50, August, 2009. Article (CrossRef Link)

Wen Liu received his B.S. degree in Computer Science from Xinjiang Normal University, Urumqi, in 2004, and his M.S. degree in Computer Science from Dalian University of Technology, Dalian, in 2009. He is currently a Ph.D. student of Computer Science at Dalian University of Technology. His research interests include database, stream data processing, and cloud computing.

Zhigang Wang received both his B.S. and M.S. degrees from School of Computer Science and Technology, Dalian University of Technology, in 2013 and 2016 respectively. His research interests include cloud computing and parallel data processing.

Yanming Shen received his B.S. degree in Automation from Tsinghua University, Beijing, in 2000, and his Ph.D. degree in Electrical Engineering from the NYU Polytechnic School of Engineering, Brooklyn, in 2007. He is a professor with the School of Computer Science and Technology, Dalian University of Technology, Dalian. His general research interests include packet switch design, data center networks, cloud computing, and distributed systems. He is a recipient of the 2011 Best Paper Awards for Multimedia Communications (awardedby IEEE Communications Society).