MCSA: a Multi-Criteria Shuffling Algorithm for the MapReduce Framework

Jean-Pierre Corriveau and Leo Lyu
School of Computer Science
Carleton University
Ottawa, Canada
Email: jeanpier@scs.carleton.ca; LeoDuo@cmail.carleton.ca

Riham Elhabyan and Wei Shi
School of Information Technology
Carleton University
Ottawa, Canada
Email:riham.elhabyan@gmail.com; wei.shi@carleton.ca

***Abstract*—During the shuffle stage of the MapReduce framework, a large volume of data may be relocated to the same destination at the same time. This, in turn, may lead to the *network hotspot* problem. On the other hand, it is always more effective to achieve better *data locality* by moving the computation closer to the data than the other way around. However, doing this may result in the *partitioning skew* problem, which is characterized by the unbalanced computational loads between the destinations. Consequently, shuffling algorithms should consider all the following criteria: data locality, partitioning skew, and network hotspot. In order to do so, we introduce MCSA, a Multi-Criteria shuffling algorithm for the MapReduce scheduling stage that rests on three cost functions to accurately reflect the trade-offs between these different criteria. Extensive simulations were conducted and their results show that the MCSA-based scheduler consistently outperforms other schedulers based on these criteria. Furthermore, the MCSA-based scheduler can be easily adjusted to the meet the distinct needs of different customers.**

I. Introduction

Nowadays, the data scale has been growing at an exponential rate due to the rapid development of many data intensive applications such as social media and e-commerce. The term *Big Data* refers to the sheer volume and speed at which the data is being generated through these data-intensive applications. Due to the massive volume of data that is generated, it must be distributed across thousands of machines in order to be processed in a reasonable amount of time. Generally, parallel computing is adopted as a solution and allows the same computations, albeit with different data sets, to be performed concurrently on many machines.

The Hadoop MapReduce framework is implemented on top of the Hadoop File System, *HDFS*. A Hadoop cluster consists of a large number of commodity machines with one node serving as the master node and multiple slave nodes. The master node runs a resource manager, called the *Job Tracker*, which is responsible for scheduling tasks on slave nodes. Each slave node runs a local node manager called *Task Tracker*, which is responsible for launching and allocating resources for each task. Hadoop MapReduce provides a FIFO-based default scheduler [3], which maintains individual tasks in a queue; every task in a job has to wait for its turn. However, this strategy may result in poor resource utilization. MapReduce jobs need to be completed in a timely manner allowing users who are making smaller queries to get results back in a reasonable time. In order to solve the problem of sharing resources fairly among users, Hadoop MapReduce supports several other job schedulers, such as the Capacity scheduler [1], Fair scheduler [2] and Delay Scheduler [12]. Moreover, the Hadoop MapReduce framework provides a *Task Scheduler* interface to allow custom designed schedulers at the task level.

During the scheduling phase, the job scheduler assigns reduce tasks to a set of reduce nodes. This may require multiple intermediate data items, with the same key-value pair, to be relocated/shuffled to this new set of reduce nodes. This stage is known as the *Shuffle* Stage.

The shuffle stage could cause a large volume of data relocation in the network at the same time, which may increase network traffic load. Moreover, this may introduce an additional delay for a job's execution. In the MapReduce framework, it is always more effective to move the computation closer to the data than to move the data closer to the computation. *Data locality* refers to assigning a task on or close to the local machine, i.e., the machine that stores the input data on its local disks. It is always a good practice to maximize the percent of tasks that achieve data locality to improve the overall performance [6]. However, assigning all tasks to local machines may lead to an uneven distribution of tasks among machines: some machines may be heavily congested while others may be idle. Moreover, many MapReduce schedulers suffer from the *partitioning skew* problem. This problem occurs when the computational load is unbalanced and unevenly distributed among reduce tasks, which will result in performance degradation [9], [10]. Therefore, it is an important challenge during the shuffle stage, to find the right balance between data locality and load balancing, while at the same time optimizing the network traffic to avoid congestion.

A. Our Contributions

We have designed and developed a Multi-Criteria Shuffling Algorithm (MCSA) for the MapReduce Framework. We present it in this paper. The MCSA-based scheduler (hereafter MCSA scheduler) is a locality, skew and network-aware scheduler. The main goal of the MCSA scheduler is to find a schedule that assigns the reduce tasks to their destinations such that the following aspects are optimized: data locality, partitioning skew, and network traffic. First, we present a Linear

Authorized licensed use limited to: Carleton University. Downloaded on June 18,2020 at 01:19:18 UTC from IEEE Xplore. Restrictions apply.

Programming (LP) formulation of the scheduling problem. In this formulation, a cost function is accurately defined for each of the aforementioned three criteria. Then, a final cost function is defined to capture the trade-off between all these criteria. Extensive simulations on arbitrary and tree networks are evaluated and compared against previously proposed schedulers. To do this, several performance metrics are used including the data locality rate, the maximum standard deviation for the node workload, as well as the link traffic. Simulation results show that the MCSA scheduler consistently outperforms the other schedulers in terms of all studied criteria. Furthermore, the MCSA scheduler can be easily adjusted to the meet the distinct needs of different customers.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 describes the MCSA algorithm in details. Section 4 presents the simulations settings and results. Finally, Section 5 concludes this research work.

II. Related Work

The Center-of-Gravity Reduce Scheduler (CoGRS) [7], [8] is a locality-aware, skew-aware reduce task scheduler. It attempts to minimize the network traffic incurred during the scheduling phase by scheduling every reduce task at its center-of-gravity node determined by the network locations. However, the CoGRS scheduler suffers two main problems. Firstly, the reduce slots starvation problem in which the selected target node may not have available slots to process at least one reduce task. Secondly, the network contention incurred by the data transfers is not considered. It is highly possible for the network traffic to have an uneven distribution due to the unpredictable locations of the intermediate files as well as their partition skews. This problem may introduce hotspots in the network, and hence lower the network resource utilization, which may compromise the benefits of the CoGRS scheduler. An illustrative example is shown in Figure 1, where the map results (only keys are shown) are generated at 5 nodes and those with a key of 3 are shuffled to the selected node for the reduce task. In this example, given the shortest path routing, all pairs of key 3 except the one at E are passed over link $(S2, C)$ to node C, which would become a hotspot, thus degrading the network performance. The route over switches $S3$ and $S4$ to node C for key 3 at node B would be a better selection, in spite of extra one hop, to remove or mitigate this phenomenon.

The smart shuffling scheduler [11] was proposed to solve the network hotspot problem. The goals of this scheduler are to minimizing the overall network traffic, achieve workload balancing and eliminate the network hotspot problem.

All the aforementioned schedulers try to optimize the scheduling phase of the MapReduce framework. However, to the best of our knowledge, none of them (except the smart shuffling scheduler [11]) investigate the combined effect of the following aspects of the MapReduce framework: data locality, partitioning skew, and network hotspot. The smart shuffling scheduler [11] investigated these aspects by defining a cost function for each aspect and selecting the set of destination nodes for the reduce tasks that return the minimum values of

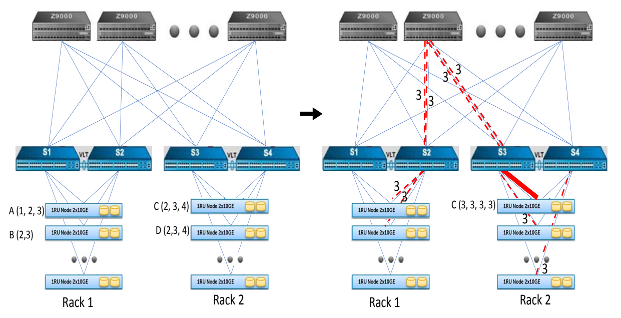


Fig. 1: An example of hotspots in a network. The intermediate key-value pairs are generated by map tasks at 5 nodes that are connected by 4 switches ($S1 - S4$), each only showing the keys in parentheses (left sub-graph). The right sub-graph shows how key 3 is shuffled from nodes A, B, C, D and E to the selected reduce task at C according to the shortest path routing (dashed red lines), and how link $(S2, C)$ becomes a hotspot link (bold red line).

the product of the cost functions. However, the cost functions were not scaled. Hence, the final cost function is more biased towards the data locality aspect. Moreover, given that the final cost function is the product of the cost function of each aspect, the result is even more biased towards the data locality aspect. Motivated by the aforementioned concerns, we propose the MCSA algorithm to develop a locality, skew and network-aware scheduler for the MapReduce framework.

III. The Scheduler Design

The MCSA shuffling algorithm starts by electing *Candidate Groups* for the reduce tasks. Each Candidate Group has a number of candidates destination nodes to process a reduce task of a specific job registered on the Job Tracker. We consider two facts when electing the candidate groups. The first one is that processing a reduce task RT at one of its feeding nodes is a good strategy to maximize the data locality [7]. The second one is that feeding nodes may not have an available spot for processing, a fact that was ignored by [7].

In this paper, we elect the candidate groups as follows: For each Reduce Task RT_x, we rank the feeding nodes in terms of the estimated intermediate data size. If one of the feeding nodes has no available slots for processing that specifically reduce task, then it is replaced with the nearest neighbor node that has an available slot for processing that reduce task. Then the first m nodes are chosen as candidate destinations for RT_x, where m is the number of Reduce Tasks to be processed. Each candidate group is evaluated according to three cost functions. A cost function for each of the following attribute: data locality, node workload, and network hotspot is defined. This cost function calculates the cost, for each attribute, of assigning each Reduce Task to a candidate destination node. The final cost function is then calculated as the weighted-sum of the cost functions for all the attributes. In order to improve the MapReduce framework performance, we want to

Authorized licensed use limited to: Carleton University. Downloaded on June 18,2020 at 01:19:18 UTC from IEEE Xplore. Restrictions apply.

place Reduce Tasks on the nodes such that the final cost is minimized.

In this paper, we assume that the network topology is presented as a graph $G(V,E)$ that comprises a set of V nodes and of E edges, where $|E| = e$ is the number of edges and $|V| = n$ is the number of nodes in G. A *Task Tracker* TT is available on each $u \in V$. Let $(TT_1, TT_2, \cdots, TT_\Delta)$ represent a list of nodes/Task Trackers that have at least one available slot that can process a reduce task. Table I presents the main notations used in this paper.

Symbol	**Definition**
m	number of Reduce Tasks
RT_i	the ith Reduce Task
Z_i	number of feeding nodes for RT_i
RT_iF_y	the yth feeding node of RT_i
$\omega(RT_iF_y)$	the estimated size of the intermediate data on RT_iF_y
D_{RT_i}	the candidate destination node of the ith Reduce Task
$\psi(RT_xF_y, D_{RT_x})$	shortest path between RT_xF_y and D_{RT_x}
$d(\psi(RT_xF_y, DRT_x))$	number of hops in $\psi(RT_xF_y, D_{RT_x})$
α	average weight of any given link
$Ł(RT_x)$	sum of the partition sizes of all feeding nodes of RT_x
β	average workload per node

TABLE I: Notations

a) Link Hotspot Elimination Cost Function: In order to minimize the network traffic, a packet of intermediate data should be sent from its feeding node to the *destination node* using the shortest path. In this paper, a link $\mathcal{L}$'s weight is 1 if $\mathcal{L}$ is on such shortest path. It should also be noted that $\mathcal{L}$ may not only be on several intermediate data relocation paths of a specific reduce task but also on the intermediate data relocation paths of multiple reduce tasks. Then, the actual weight of link $\mathcal{L}$ is the total number of times all intermediate data relocation paths of all reduce tasks that pass through $\mathcal{L}$. Then, the Link Weight of a link $\mathcal{L} = (u,v) \in E$ is calculated as follows:

$$W(u,v) = \sum_{x=1}^{m}\sum_{y=1}^{\Delta}\begin{cases} 1 & \text{if } (u,v) \in \psi(RT_xF_y, D_{RT_x}) \\ 0 & \text{otherwise} \end{cases}$$

Accordingly, α is calculated as follows:

$$\alpha = \sum_{x=1}^{m} Z_x/e$$

The link hotspot elimination cost function, A, is calculated as follows:

$$A = \max(|W(u,v) - \alpha|) \quad (for\ all\ (u,v) \in E)$$

b) Node Workload Cost Function: Any node u can be the destination node of several reduce tasks, depending on the available number of reduce slots that it has. Hence, the *workload* of certain node u, $Ł(u)$, is the total size of intermediate data of all reduce tasks that has node u as its destination node and is calculated as follows:

$$Ł(u) = \sum_{x=1}^{m}\sum_{y=1}^{\Delta}\begin{cases} \omega(RT_xF_y) & \text{if } u \text{ is the } D_{RT_x} \text{ of } RT_x \\ 0 & \text{otherwise} \end{cases}$$

Then, β, is calculated as follows:

$$\beta = \sum_{x=1}^{m} Ł(RT_x)/\Delta$$

The node workload cost function, B, is calculated as follows:

$$B = \max(|Ł(u) - \beta|) \quad (for\ all\ u \in V)$$

c) Data Locality Cost Function: The data locality cost function is calculated using the following formula:

$$C = \sum_{x=1}^{m}\sum_{y=1}^{\Delta} d(\psi(RT_xF_y, D_{RT_x})) \times \omega(RT_xF_y)$$

Ultimately, the goal of the proposed scheduler is to choose a list of destination nodes for the reduce tasks that gives the minimum value for:

$$a \times A + b \times B + c \times C$$

It should be noted that each cost function is scaled to produce results in a set of values in the range $[0.0, 1.0]$. Changing the coefficients a, b and c will allow us to change the weights of each aspect that changes the overall performance accordingly.

Then, the value of A, B, and C are calculated for each selected candidate group. The Floyd-Warshall algorithm [5] was used to calculate the shortest path from a feeding node of a reduce task to its destination node.

IV. Simulations and Results

For this paper, we have performed a simulation study to investigate the performance of the proposed scheduler. Moreover, we have compared the performance of our proposed scheduler with the *Smart Shuffling Scheduler* (SS) [11], the *Center-of-Gravity Reduce Task Scheduler* (CoGRS) [7], and a *Random Scheduler* (RS). The RS scheduler randomly assigns nodes as the destination nodes of the reduce tasks.

With respect to the topology of the nodes, the simulations were performed on two groups of networks: arbitrary networks and tree networks. For each group, we varied the number of nodes from 100 to 250. The arbitrary network topologies were generated using the Erdös-Rényi's model [4], in which value p represents the probability of choosing an edge. In our simulations, we chose $p = 0.05$, 0.06, 0.07 and 0.08 in order to randomly create arbitrary networks that are not so dense that they would resemble complete networks.

We have performed two sets of the experiments for each network group: a) We compared the performance of the proposed scheduler against the SS scheduler, the CoGRS scheduler, and the RS scheduler. In order to do that, we used $a = 1$, $b = 1$, and $c = 1$ as the coefficients of A, B and C, assuming all three criteria are equally important; b) The second set of experiments studied the performance of the MCSA scheduler under different criteria selections.

For both sets of experiments, the number of map tasks m is between 80% to 90% of the entire network size and the number of reduce tasks is between 80% to 90% of the number of map tasks. The number of feeding nodes ranges between 50% and 60% of the entire network size. Furthermore, the following

Authorized licensed use limited to: Carleton University. Downloaded on June 18,2020 at 01:19:18 UTC from IEEE Xplore. Restrictions apply.

performance metrics were used to measure the performance of the aforementioned schedulers:

Data Locality Rate (DLR) is used to measure the data locality for each scheduler. The DLR metric is defined as the ratio between the number of Reduce Tasks that are executed on one of its feeding nodes and the total number of all Reduce Tasks. High *DLR* values imply better performance in terms of the data locality. *Maximum Standard Deviation of the node Workload* (STD-Workload) is used to measure the partitioning skew of the nodes. Smaller *STD-Workload* values imply better performance in terms of the partitioning skew problem. *Maximum Standard Deviation of the Link traffic* (STD-Link) is used to measure the network traffic. Smaller *STD-Link* values imply better performance in terms of the network hotspot problem.

1) Performance of the MCSA scheduler against other schedulers: The goal of this set of experiments is to study the performance of the MCSA scheduler against the SS scheduler, the CoGRS scheduler, and the RS scheduler. For this set of experiments, we set $a = 1$, $b = 1$, and $c = 1$ and hence we consider all three criteria simultaneously. Figure 2 shows the DLR, STD-Workload, and STD-Link values of all schedulers under comparison, for arbitrary and tree networks.

Figures 2(a) and 2(b) show that the MCSA scheduler consistently outperforms the other schedulers in terms of the data locality criteria, even though we are considering all three criteria equally. As we pointed out earlier, this is due to the proposed candidate selection method, which ranks and hence favors the feeding nodes that have higher intermediate data as candidate destinations for the reduce tasks. These results are also consistent with Figures 3(e) and 3(f). Figures 2(c) and 2(d) show that the MCSA scheduler consistently outperforms the other schedulers in terms of the partitioning skew criteria. Moreover, the MCSA scheduler has significantly better performance as the network scales up in tree networks. Figures 2(e) and 2(f) show that the MCSA scheduler consistently outperforms the other schedulers in terms of the network hotspot criteria. The MCSA scheduler shows more balanced network traffic in comparison to the other schedulers.

It was also noted that the SS scheduler is the second best in terms of the DLR, in all the tree network topologies and in most of the arbitrary networks. This is due to its definition of the final cost, which is more biased towards the data locality criteria. However, a close look at Figures 2(c) and 2(d) reveals that the SS scheduler falls behind the other schedulers in most of the arbitrary and tree networks.

As the network size scales up, the COGV scheduler falls behind the other schedulers in terms of the partitioning skew and network traffic criteria. This result is consistent with Figures 2(a) and 2(b) that show that the COGV scheduler suffers from poor performance in terms of the DLR. This means that a large volume of data is relocated, which causes poor performance in terms of network traffic and may cause an unbalanced workload.

It can be concluded that the MCSA scheduler finds the best trade-offs for all the criteria under consideration.

2) Performance of the MCSA scheduler under different criteria selections: The goal of this set of experiments is to study the performance of the MCSA scheduler in two cases. The first one assumes that all three criteria are equally important and the second one assumes that we are interested in one criterion at a time. In order to consider one specific criterion at a time and ignore the others, we set the weight coefficient of this criterion to 1 and the weight coefficients of the other criteria to 0. For example, if we want to study the performance of the MCSA scheduler in terms of the data locality criterion only, we set the corresponding weight coefficient c to 1 and the other weight coefficients $a = b = 0$. Figure 3 shows the DLR, STD-Workload, and STD-Link values of the MCSA scheduler, for arbitrary and tree networks, when considering different criteria.

In terms of the data locality criterion: Figures 3(e) and 3(f) show that the MCSA scheduler has a high DLR, even when we are considering the other criteria equally at the same time. This is due to the proposed candidate selection method that ranks and hence favors the feeding nodes that have higher intermediate data as candidate destinations for the reduce tasks. Figures 3(e) and 3(f) show that for both types of networks, all the reduce tasks were executed at their corresponding feeding nodes.

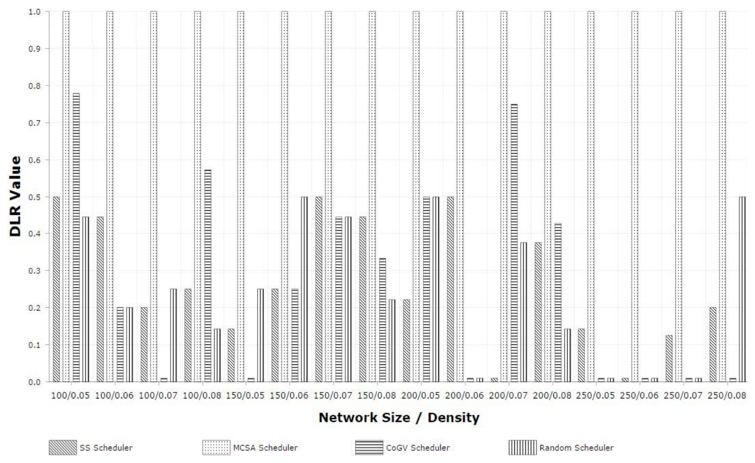
In terms of the partitioning skew criterion: Figures 3(c) and 3(d) show that the MCSA scheduler has better performance, in terms of the node workload balance, when considering the partitioning skew criteria by itself. Taking into consideration all three criteria at the same time results in a higher STD-Workload because more emphasis is put on the other criteria. Moreover, in arbitrary networks, the MCSA scheduler shows better performance when the network size scales up.

In terms of the network hotspot criterion: Figures 3(a) and 3(b) show that the MCSA scheduler has better performance, in terms of the link traffic, when only considering the network hotspot criterion. Taking into consideration all three criteria at the same time results in higher STD-Link because more emphasis is put on the other criteria. Moreover, the MCSA scheduler shows better performance when the network size scales up. As a result of these findings, the MCSA scheduler can be easily adjusted to meet the distinct needs of different customers. The weight coefficients a, b and c values can be easily tuned to the customer preference or interest. Customers who want to consider only one criterion will assign it a weight coefficient of 1 and of 0 for the other criteria. Moreover, different weights can be assigned to a, b and c to find the desired trade-off between these criteria.

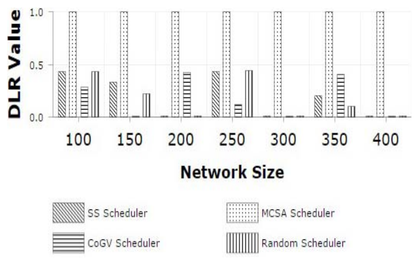
V. Conclusion

In this paper, we have designed and developed a multi-criteria shuffling algorithm that can be adopted by different MapReduce schedulers to improve the shuffling stage by optimizing the following criteria: data locality, partitioning skew, and network hotspot. In order to achieve this, we defined three cost functions to accurately reflect the trade-offs between the different criteria. Our extensive simulations show that the

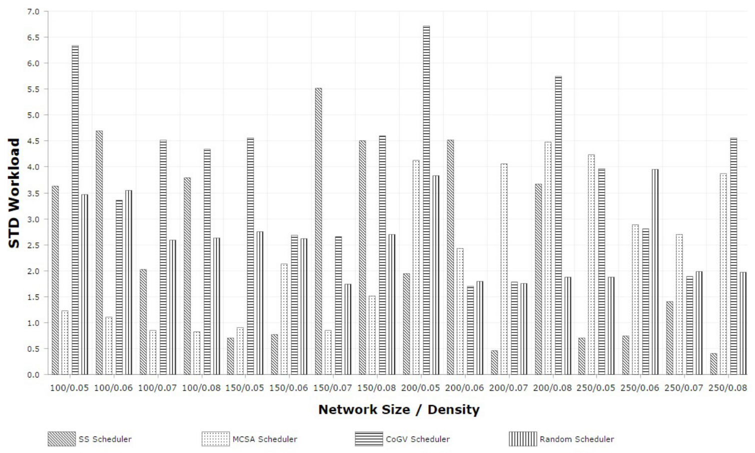
Authorized licensed use limited to: Carleton University. Downloaded on June 18,2020 at 01:19:18 UTC from IEEE Xplore. Restrictions apply.



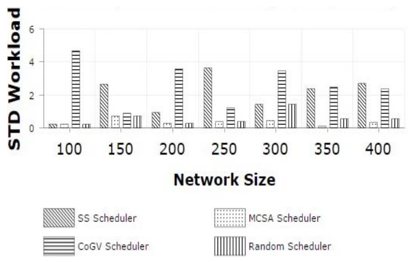
(a) DLR comparison in arbitrary networks



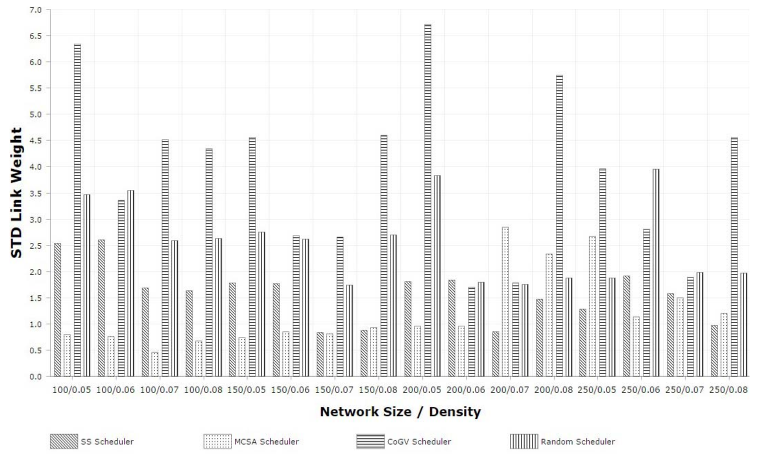
(b) DLR comparison in tree networks



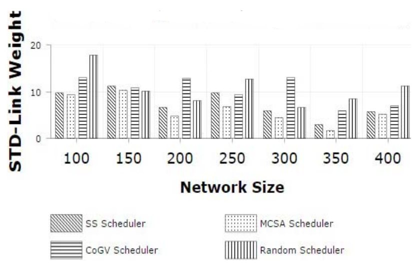
(c) STD-Workload comparison in arbitrary networks



(d) STD-Workload comparison in tree networks



(e) STD-Link Weight comparison in arbitrary networks



(f) STD-Link Weight comparison in tree networks

Fig. 2: Performance of MCSA (when a=b=c=1) scheduler against SS, CoGRS and RS schedulers.

MCSA Scheduler consistently outperforms other schedulers with respect to these criteria.

REFERENCES

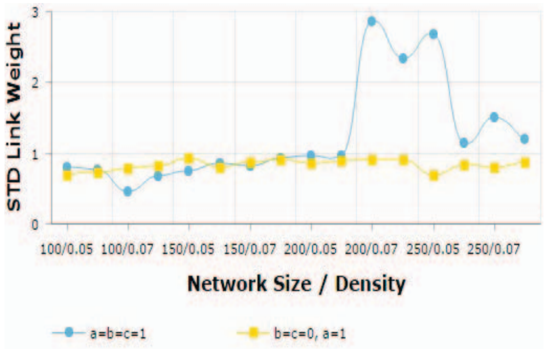
[1] Hadoop CapacityScheduler, http://Hadoop.apache.org/docs/r0.19.1/capacity_scheduler.html [Online; accessed Jan-11-2014].

[2] Hadoop FairScheduler, http://Hadoop.apache.org/docs/r1.2.1/fair_scheduler.html [Online; accessed Jan-11-2014].

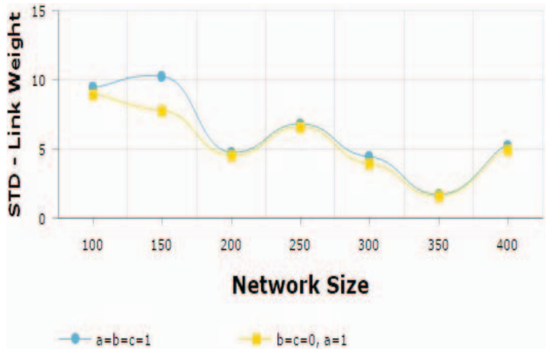
[3] DEAN, J., AND GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. *Commun. ACM 51*, 1 (Jan. 2008), 107–113.

[4] ERD, P., ET AL. {On Random Graphs, I}. *Publicationes mathematicae 6* (1959), 290–297.

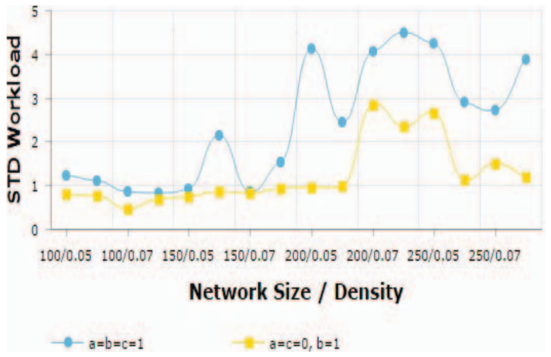
Authorized licensed use limited to: Carleton University. Downloaded on June 18,2020 at 01:19:18 UTC from IEEE Xplore. Restrictions apply.



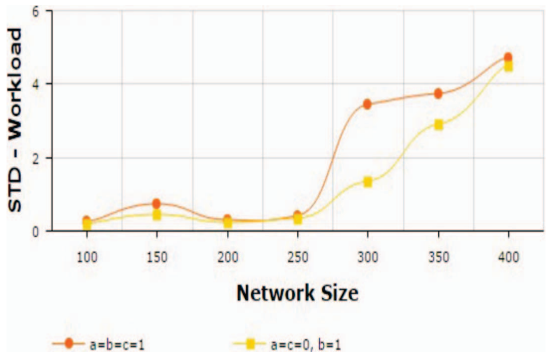
(a) STD-LinkWeight measurement in arbitrary networks: $a = b = c = 1$ vs. $b = c = 0$, $a = 1$



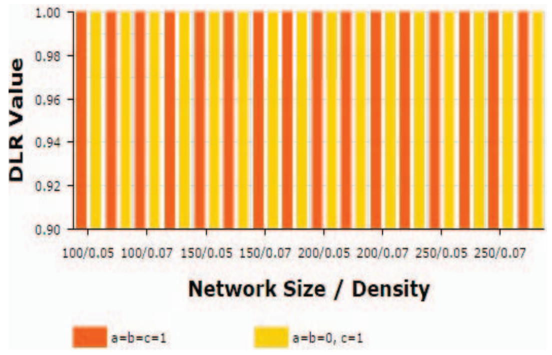
(b) STD-LinkWeight measurement in tree networks: $a = b = c = 1$ vs. $b = c = 0$, $a = 1$



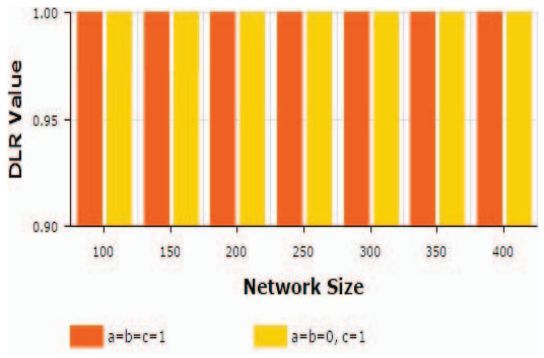
(c) STD-Workload measurement in arbitrary networks: $a = b = c = 1$ vs. $a = c = 0$, $b = 1$



(d) STD-Workload measurement in tree networks: $a = b = c = 1$ vs. $a = c = 0$, $b = 1$



(e) DLR measurement in arbitrary networks: $a = b = c = 1$ vs $a = b = 0$, $c = 1$



(f) DLR measurement in tree networks: $a = b = c = 1$ vs. $a = b = 0$, $c = 1$

Fig. 3: Performance of MCSA Scheduler under different criteria selections

[5] FLOYD, R. W. Nondeterministic algorithms. *Journal of the ACM (JACM) 14*, 4 (1967), 636–644.

[6] GUO, Z., FOX, G., AND ZHOU, M. Investigation of data locality in mapreduce. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)* (2012), IEEE Computer Society, pp. 419–426.

[7] HAMMOUD, M., REHMAN, M. S., AND SAKR, M. F. Center-of-gravity reduce task scheduling to lower mapreduce network traffic. In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing* (2012), CLOUD '12, pp. 49–58.

[8] HAMMOUD, M., AND SAKR, M. F. Locality-aware reduce task scheduling for mapreduce. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on* (2011), IEEE, pp. 570–576.

[9] LIU, Z., ZHANG, Q., AHMED, R., BOUTABA, R., LIU, Y., AND GONG, Z. Dynamic resource allocation for mapreduce with partitioning skew. *IEEE Transactions on Computers 65*, 11 (Nov 2016), 3304–3317.

[10] POLATO, I., R, R., GOLDMAN, A., AND KON, F. A comprehensive view of hadoop research: A systematic literature review. *Journal of Network and Computer Applications 46* (2014), 1 – 25.

[11] SHI, W., WANG, Y., CORRIVEAU, J. P., NIU, B., CROFT, W. L., AND PENG, M. Smart shuffling in mapreduce: A solution to balanced network traffic and workloads. In *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)* (Dec 2015), pp. 35–44.

[12] ZAHARIA, M., BORTHAKUR, D., SEN SARMA, J., ELMELEEGY, K., SHENKER, S., AND STOICA, I. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems* (2010), ACM, pp. 265–278.

Authorized licensed use limited to: Carleton University. Downloaded on June 18,2020 at 01:19:18 UTC from IEEE Xplore. Restrictions apply.