2019 IEEE International Conference on Cloud Computing, June 8-13, Milan, Italy (c) IEEE 2019

A highly efficient data locality aware task scheduler for cloud-based systems

Jia Ru*, Yun Yang*, John Grundy†, Jacky Keung‡ and Li Hao§
*School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia 3122
Email:{rjia, yyang}@swin.edu.au
†Faculty of Information Technology, Monash University, Melbourne, Australia 3145
Email:john.grundy@monash.edu
‡Department of Computer Science, City University of Hong Kong, Hong Kong SAR, China
Email:Jacky.Keung@cityu.edu.hk
§Amaris.AI Pte.Ltd, Singapore, 089760
Email:liucoolhao@gmail.com

***Abstract*—Scheduling tasks in the vicinity of stored data can significantly diminish network traffic. Scheduling optimisation can improve data locality by attempting to locate a task and its related data on the same node. Existing schedulers tend to ignore overhead and tradeoff between data transfer and task placement, and bandwidth consumption, by only emphasising data locality without considering other factors. We present a novel data locality aware scheduler for balancing time consumption and network bandwidth traffic – DLAforBT – to improve data locality for tasks and throughput, with the optimal placement policy exhibiting a threshold-based structure. DLAforBT uses bipartite graph modelling to represent data placement, adopts a judgment mechanism and a precise prediction model to determine moving data or moving computation. It integrates an improved Dominant Resource Fairness (DRF) resource allocation to capture tenants' resource allocation and run as many jobs as possible. DLAforBT improves by 16% of data locality rate, and 25% of throughput.**

***Index Terms*—data locality, multi-tenancy, scheduling, bipartite graph modelling, cloud computing**

I. Introduction

With the increase in volumes of data, cloud computing has become more important for large-scale data intensive applications, e.g. search engines and online map services. These applications not only process but also generate very large amount of data, which brings more complicated challenges in the issues of data locality, computation placement and task scheduling. Bandwidth per node within a rack is much higher than bandwidth per node between racks. Avoiding off-switch data exchange is critical for cloud's performance. Putting a task near its required data can reduce network traffic and cost.

To enhance data locality for clouds, there has been much research into scheduling tasks close to their data [1]–[4]. BOLAS in [2] models the scheduling process as a bipartite-graph matching problem to assign data block to the nearest task. Work in [3] studies the data locality problem by utilising data migration and hotspot file prediction for lowering task execution waiting delay. Such schedulers greedily search for a task and allocate it to the node with required data. Limitations are that nodes are not idle quickly enough as assumed. If the nodes are overloaded, forcing tasks to wait for the resource releasing would waste time. Straightforward emphasising data locality regardless of overhead and time cost would sacrifice system's performance. Moreover, data transfer time is dependent on network bandwidth, which helps to decide task allocation. When it takes less time on data transfer, moving computation (data locality) is not cheaper than moving data. However, most of works do not consider the impact of data transfer time on data locality. A key question is how to schedule tasks in the vicinity of their inputs to diminish shuffled data, reduce unnecessary data transfer and network traffic, and improve system's performance. To make scheduling decisions, constructing a performance estimation model that estimates the execution time and waiting time of jobs is essential. Work in [4] calculates task's average completion time based on the node capacity level. However, the accuracy of this task's remaining time evaluation is not high in some situations due to the data locality problem.

We propose a highly efficient ***data locality aware*** scheduler to ***balance time*** consumption and network bandwidth ***traffic***, named as ***DLAforBT***. Our scheduler can improve data locality, throughput, and network bandwidth usage. We transform the data locality scheduling problem into the well-known maximum weighted bipartite matching (MWBM) graph problem; use a judgment mechanism to dynamically adjust task allocation and an embedded precise prediction model to determine moving computation or moving data; rank a list of idle computation capacity nodes in descending order and give different priorities to nodes, aiming to maximise resource usage; and integrate our proposed DRF resource allocation [5] to maximise the number of jobs being allocated.

II. Problem Formalisation and Solution

Our task scheduling uses a bipartite graph matching approach that maps the knowledge of data block distribution and relative performance of nodes. A job is not completed until all the sub tasks are finished. We rank nodes based on idleness utilisation of computation capacity in a descending order. Usually, we choose the top node of the ranking list with a replica. The node with highest available resource utilisation has highest priority and will be first option to process tasks.

The data placement is modeled by a weighted bipartite graph $G = (T \cup S, E)$, where, T is the set of tasks, S is the set of nodes, $E \subseteq T \times S$ is the set of edges between T and S. wei_i is the set of edges' weights and indicates remaining available resource utilisation. Resource utilisation of node n_i is u^i_{res}. Available resource utilisation of node n_i is $u^i_{ava\_res} = 1 - u^i_{res}$, which is denoted as weight $wei(*, i)$. Edge $e(t, s)$ denotes the input data blocks of task $t \in T$ is placed on node $n_s \in S$. $S^G_{pre}(t)$ is the set of t's preferred nodes in G. When $S^G_{pre}(t)$ is larger than 1, task t selects node n_s which has maximum weight wei_s, $n_s \in S^G_{pre}(t)$, under the condition that several nodes have the same required data. Resource allocation is described as $f : T \to S$ that allocates task t to node $n_{f(t)}$. Defining α is the allocation for t. There exist multiple edges $e(t, s)$, between t and nodes. To avoid overload and balance loads, choose the node with the maximum weight $wei(t, s)$ and mark n_s as $n_{\alpha(t)}$. Task t is allocated to $n_{\alpha(t)}$. Node $n_{\alpha(t)}$'s resource utilisation is $u^{\alpha(t)}_{res}$, and its available resource utilisation is $u^{\alpha(t)}_{ava\_res} = 1 - u^{\alpha(t)}_{res}$. Task allocation problem is transformed to maximum weighted bipartite matching problem. This problem finds a complete allocation that reduces the number of remote tasks, and improves system's throughput and locality rate. Edge $e(t, \alpha(t))$ indicates that $n_{\alpha(t)}$ has t's data. Putting t on $n_{\alpha(t)}$ or other node depends on $u^{\alpha(t)}_{ava\_res}$. Judgment mechanism evaluates data transfer time $T^{tran}_t(d_t)$ and resource releasing time $W^{rel}_{\alpha(t)}$. $T^{tran}_t(d_t)$ of task t can be estimated as [1]:

$$T^{tran}_t(d_t) = \frac{|d_t|}{cap_t} = |d_t| \times (\frac{rtt_t(n_i, n_j)}{tw_t}) \quad (1)$$

where, $|d_t|$ is the size of data block d_t, $cap_t = \frac{tw_t}{rtt_t(n_i,n_j)}$ is task t's transfer capacity, tw_t is the TCP window size of an initiated TCP connection in t which specifies the maximum number of data bytes to be received, $rtt_t(n_i, n_j)$ is the round-trip delay time for TCP connection between node n_i and n_j.

A multilayer Perceptron (MLP) model as a back-propagation method based on neural networks is precise enough to estimate resource releasing time i.e. the remaining execution time of tasks. This work uses Keras [6] MLP API to realise time prediction.

III. Our data locality aware scheduler

Fig. 1 presents our DLAforBT's architecture. DataNodes are spread across multiple racks and store data (1). Submitted jobs are divided into *n* tasks. These *n* tasks need data blocks that are mostly stored in DataNodes 1 and 2, so most of tasks should be run on Rack 1 and the relevant containers should be created on Rack 1. Task 1 is deployed on DataNode 1 which has 3 data blocks (2). DatasNodes are ranked based on idle resource utilisation (3). Task 1 occupies some resources of DataNode 1. DataNode 2's idle resource utilisation is thus higher than DataNode 1, and has higher priority. Even if DataNodes 1 and 2 both have the same data, task 2 is still put on DataNode 2 (4). DataNodes 1 and 2 are occupied, so the new coming task 3 cannot gain sufficient resource from these 2 DataNodes. Thus, task 3 needs to consider waiting for DataNode 1 or 2 releasing resources, or move to another node. DLAforBT estimates enough resource releasing time of DataNode 1 or 2 and calculates data transfer time to DataNode 3 which has the second most required data. If our judgment mechanism finds resource releasing time is larger than data transfer time, task 3 is allocated to DataNode 3 (5). If all nodes on Rack 1 are busy and only DataNode 5 is idle but has 2 required data blocks for task *n* (6), DLAforBT will need to transfer orange data block from Rack 1 and yellow data block from DataNode 4 to 5 (7). Our judgment mechanism finds data transfer time to DataNode 5 is smaller than waiting time for releasing resource from DataNodes 1 and 2 (8). Thus task *n* is put on DataNode 5 (9). Our resource allocation uses our previous work (10) – 3-dimensional demand vector <CPU, memory, vdisk> DRF algorithm [5] to optimise the number of jobs being serviced.

Algorithm 1 DLAforBT Scheduler

1: **while** (jobs run their sub tasks) **do**
2: $Res^n_{rem} \rightarrow$ remaining available resources of node n_n
3: $Dem_t \rightarrow$ the resource demand of task t
4: **for** (each job j of J in the queue) **do**
5: Partition job j into sub tasks T
6: **end for**
7: **// Maximum weighted bipartite graph formation to model the connections between T and S**
8: **for** (each task t of T) **do**
9: **for** (data locality nodes for task t) **do**
10: **if** (node n_s meets task t's data requirements) **then**
11: $S \leftarrow S \cup n_s$
12: $wei(t, s) \leftarrow$ The edge weight is associated with idleness utilisation for computation capacity of node n_s
13: $E \leftarrow E \cup (t, s)$
14: **end if**
15: **end for**
16: find feasible nodes for t, $n_{f(t)}$, and collect $n_{f(t)}$ in S
17: $max\_wei(t, s) \leftarrow$ Find the maximum edge weight in E
18: **end for**
19: **// Judgment mechanism**
20: **while** ($Res^{f(t)}_{rem} < Dem_t$) **do**
21: Calculate the required data transfer time for t, $T^{tran}_t(d_t)$
22: Calculate $n_{f(t)}$'s resource releasing time $W^{rel}_{f(t)}$
23: **if** ($W^{rel}_{f(t)} < T^{tran}_t(d_t)$) **then**
24: Put t on node $n_{f(t)}$ to wait for resources
25: **else**
26: Find a neighbour node $n_{nei} \leftarrow (Res^{n_{nei}}_{rem} > Dem_t)$
27: Move t to n_{nei}
28: **end if**
29: **end while**
30: **// Resource allocation strategy**
31: Call our previous work-improved DRF allocation strategy [5]
32: **end while**

Algorithm 1 shows the pseudo code of DLAforBT. Submitted jobs are divided into tasks. When jobs run their tasks currently, these running tasks are collected in task set T (*Lines:1-6*). Next, DLAforBT finds the feasible nodes of each running task. If node n_s provides proper execution resources to meet the task t's requirements and data retrieval of the original job, n_s is a feasible node of t, and there is a corresponding edge $e(t, s)$ in a weighted bipartite graph $G = (T \cup S, E)$ (*Lines:10-11*). We label weight $wei(t, s)$ as available resource utilisation of n_s (*Lines:12-13*). The feasible nodes are kept in set S (*Line:16*).

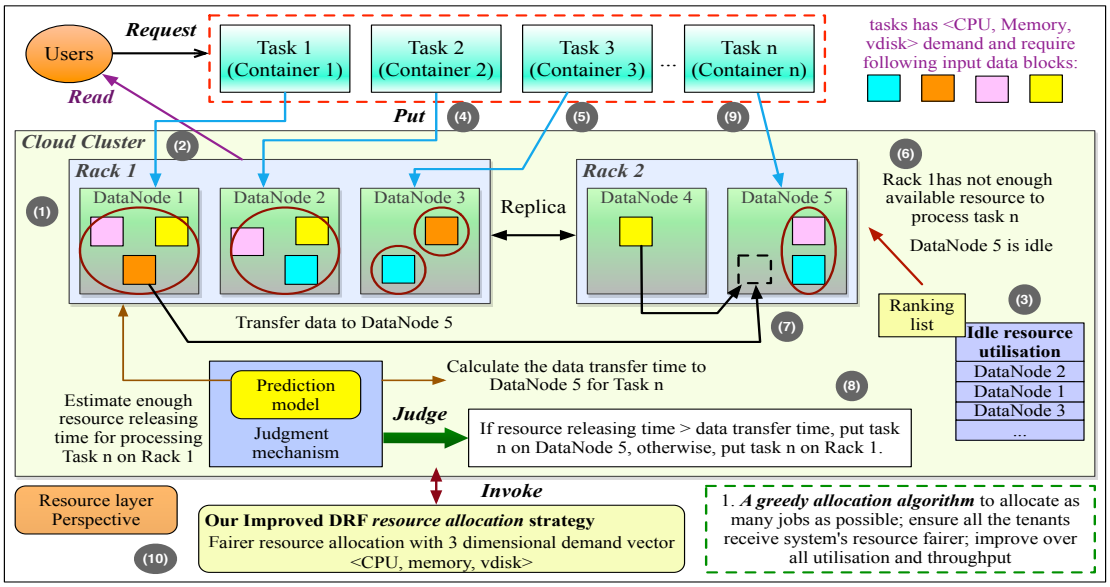

Fig. 1. DLAforBT scheduler architecture: moving computation or moving data

We choose the node with maximum weighted edge in $n_{f(t)}$ as first option to run t (*Line:17*). Yet, even if finding proper node $n_{f(t)}$ for t, $n_{f(t)}$ may not provide enough resources for t. We use a judgment mechanism to decide moving t to other node, or waiting on $n_{f(t)}$ until $n_{f(t)}$ has enough resources. If $W^{rel}_{f(t)}$ is less than $T^{tran}_t(d_t)$, t costs less tradeoff on $n_{f(t)}$, so we put t on $n_{f(t)}$ (*Lines:23-24*). Otherwise, t costs more to wait for resources on $n_{f(t)}$, so move t to other node (*Lines:26-27*).

IV. Experiments

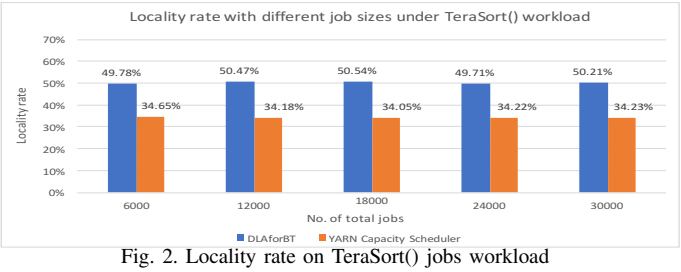

Fig. 2. Locality rate on TeraSort() jobs workload

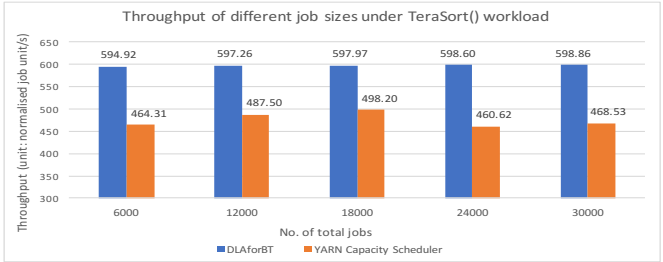

Fig. 3. Throughput on TeraSort() jobs workload

We compared DLAforBT's performance to YARN Capacity Scheduler [7]. Our experiments select **TeraSort** application that is a standard MapReduce sort to sample input data and use map/reduce to sort the data into a total order. In Fig. 2, DLAforBT's mean locality rate achieves 50.14%, and Capacity scheduler's is only 34.27%. DLAforBT gets 16% improvement. DLAforBT both considers data locality and focuses on reducing the tradeoff. In Fig. 3, DLAforBT's mean throughput achieves 597.52 job units/s. Whatever the number of jobs changes, DLAforBT's throughput is very steady. However, Capacity Scheduler's throughput is not stable. When the number of jobs is 6,000, its throughput is 464.31 job units/s, but when the number of jobs is 18,000, its throughput can reach 498.20 job units/s. Its average throughput is 475.83 job units/s. In comparison, DLAforBT improves this by 25.57%.

V. Conclusions

Improving data locality for task scheduling is crucial, but only using the principle of moving computation close to data is inappropriate. To reduce bandwidth cost and improve performance, we propose a highly efficient data locality aware scheduler for balancing time consumption and network traffic, named ***DLAforBT***. DLAforBT uses a bipartite graph to model task placement to improve data locality driven task allocation. A neural network prediction model is built to estimate resource release time. A judgment mechanism is used to decide moving data or computation. DLAforBT integrates an improved DRF allocation method to run as many jobs as possible. DLAforBT improves 16% of data locality rate and 25% of throughput.

References

[1] H. Chen, W. Lin, and Y. Kuo, "MapReduce scheduling for deadline-constrained jobs in heterogeneous cloud computing systems," *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 127–140, 2018.

[2] R. Xue, S. Gao, L. Ao, and Z. Guan, "Bolas: bipartite-graph oriented locality-aware scheduling for mapreduce tasks," in *14th Int. Sym. Parallel and Distributed Computing*. IEEE, 2015, pp. 37–45.

[3] C. Li, J. Zhang, T. Ma, H. Tang, L. Zhang, and Y. Luo, "Data locality optimization based on data migration and hotspots prediction in geo-distributed cloud environment," *Knowledge-Based Systems*, vol. 165, pp. 321–334, 2019.

[4] Z. Tang, J. Zhou, K. Li, and R. Li, "A mapreduce task scheduling algorithm for deadline constraints," *Cluster Computing*, vol. 16, no. 4, pp. 651–662, 2013.

[5] R. Jia, J. Grundy, Y. Yang, J. Keung, and H. Li, "Providing fairer resource allocation for multi-tenant cloud-based systems," in *7th Int. Conf. on Cloud Computing Technology and Science*. IEEE, 2015, pp. 306–313.

[6] "Keras," in *https://keras.io/*.

[7] "Hadoop," in *http://hadoop.apache.org*.