DRASH: A Data Replication-Aware Scheduler in Geo-distributed Data Centers

Moïse W. Convolbo*, Jerry Chou*, Shihyu Lu† and Yeh Ching Chung*
wend_kuuni@sslab.cs.nthu.edu.tw, jchou@lsalab.cs.nthu.edu.tw, shiyosylu@iii.org.tw, ychung@cs.nthu.edu.tw
* National Tsing Hua University, Taiwan R.O.C., †Institute for Information Industry, Taiwan R.O.C.

***Abstract*—Driven by the trends of BigData and Cloud computing, there is a growing demand for processing and analyzing data that are generated and stored across geo-distributed data centers. However, due to the limited network bandwidth between data centers and the growing data volume spread across different locations, it has become increasingly inefficient to aggregate data and to perform computations at a single data center. An approach that has been commonly used by data-intensive cluster computation systems, like Hadoop, is to distribute computations based on data locality so that data can be processed locally to reduce the network overhead and improve performance. But limited work has been done to adapt and evaluate such technique for geo-distributed data centers. In this paper, we proposed DRASH (Data-Replication Aware Scheduler), a job scheduling algorithm that enforces data locality to prevent data transfer, and exploits data replications to improve overall system performance. Our evaluation using simulations with realistic workload traces shows that DRASH can outperform other existing approaches by 16% to 60% in average job completion time, and achieve greater improvements under higher data replication factors.**

Keywords—Job scheduling, Geo-distributed system, Data replication, Data center, Data analytic jobs

I. Introduction

Over the past several decades, grid community has been building infrastructures and platforms across geo-distributed data centers to facilitate inter-organizational collaboration. Since the emergence of cloud computing, cloud providers like Amazon [1], Microsoft [2] and Google [3] also start aggressively expanding their service platforms by building and connecting their own data centers worldwide. Today, people even rely on multi-cloud systems to address vendor lock-in and security problems [?]. Thus, computing systems based on geographically distributed data centers has become increasingly popular and accessible. In particular, the recent advent of Big data has contributed to reaching a *data deluge* where information generated and collected is overwhelmingly exceeding the capacity of institutions to manage and make use of it within a single data center. Therefore, geo-distributed data centers have become the essential platform to support large-scale data analytic applications.

In response to the growing demand for data processing, massive parallel processing frameworks like MapReduce [4], Hadoop [5], Spark [6], and Dryad [7], have been developed and efficiently used in large clusters for data processing. Applications based on these models split a job into many independent tasks, each of them independently processing a subset of data, and the data are distributed on the servers in a cluster. Because moving data consumes a large amount of network bandwidth, jobs are scheduled based on data locality, which means that tasks are dispatched to the server hosting the data to be processed in order to minimize the network overhead. However, as shown by the recent studies [8], these frameworks are not suitable to be ran and deployed across geo-distributed data centers directly because the heterogeneous Wide Area Networks across data centers can cause prohibited overhead to the communication and control mechanisms that were designed for fault-tolerance within a cluster.

On the other hand, although many scheduling algorithms have been proposed [9], [10], [11], [12], limited work has been done to adapt and evaluate locality aware scheduler for geo-distributed data centers. Recently, C.C. Hung et. al [13] proposed a state-of-art data locality aware scheduling technique called SWAG. However, this work did not consider data replications which has become a common technique to improve system performance, availability, and reliability, especially for geo-distributed systems. Therefore, in this paper, we introduce a job scheduling technique that can further exploits the existing data replications on each local data center to minimize the average completion time of jobs. To achieve our goal, the algorithm must make two decisions: *(1) where to place the tasks? (2) in which order to execute the jobs in a data center?* The scheduling problem in this context is a multi-variant optimization problem. Due to the number of parameters (the number of data centers in the system, the number of jobs, tasks and number of replica per data), the solution space grows exponentially making the problem more complex. Therefore, we proposed a low complexity heuristic Data-Replication-Aware Scheduler, called DRASH. Our contributions are summarized as below:

- We developed a Data-Replication-aware Scheduler (DRASH) that facilitates the integration of existing data replica in the scheduling decision on geo-distributed environments.
- We evaluated a number of real workload traces and various data placement models to investigate the performance factors in the job scheduling problem on geo-distributed data centers.
- We demonstrated the robustness of our solution in a heterogeneous environment settings with different processing capability in each data center to show the applicability of DRASH.

The rest of the paper is organized as follows: We describe our problem Section II, and propose our scheduling algorithm in Section III. The experimental results are presented in Section IV followed by a discussion on the related works in Section V. Finally, Section VI concludes the paper.

2330-2186/16 $31.00 © 2016 IEEE
DOI 10.1109/CloudCom.2016.53

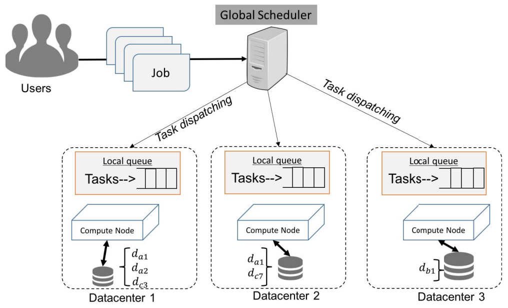


Fig. 1: System architecture

II. Problem Description

A. System Model

Fig. 1 depicts the system model of a typical geo-distributed system consisted of multiple data centers. All jobs are submitted in a centralized global scheduler to be dispatched among the data centers for processing. The global scheduler maintains a FIFO queue Q for all submitted jobs. Data intensive applications for instance, often produce and store data locally (e.g., E-commerce websites and social media). However, these data may be replicated on other data centers. We follow the same model and assume that the initial data location as well as the replication policy are given. In this paper, we consider data analytic jobs that can be divided into multiple independent sub-jobs/tasks. A local task queue q is maintained by each data center for the tasks scheduled to that particular data center. Due to the expensive data transfer delay in geo-distributed systems, a task can only be scheduled on a data center which already has its required data. Meanwhile, the completion of a job is governed by the latest completion among all its tasks. Once tasks are dispatched to data centers, they remain in the local task queues until they are executed. A local scheduler in each data center reports the progress of the local queue to the global scheduler for the assigned tasks. The model described above is relevant to common cloud platforms. For example, Amazon Web Service (AWS) [1] provides data centers in multiple geographic regions where data centers are grouped by zone. Each zone provides users with a local storage system and compute nodes accessible from any other location. Moreover, the very similar system model is used by previous studies [14], [13].

B. Problem Definition

We consider a system consisted of N geo-distributed data centers denoted by $S = \{DC_1, DC_2, \cdots, DC_i, \cdots, DC_N\}$. The arriving jobs in the system is represented by a set of jobs, $J = \{J_1, J_2, \cdots, J_K\}$, and a set of tasks $T = \{t_i^j | 1 \leq j \leq K, 1 \leq i \leq M\}$, where t_i^j means the i^{th} task of job J_j, and M is the maximum number of tasks per job. For the simplicity of discussion, we assume each task takes one unit of processing time for the rest of paper. But in our experiments, we generalize our algorithm, and consider workloads with varied execution time per task. Each task is associated with a required data, so we denote d_i^j as the data required by task t_i^j. The data is replicated in the system, and we use $\mathcal{L}_{j,k}^i = \{0, 1\}$

TABLE I: An example of a geo-distributed system which consists of 3 data centers, and stores the data for 3 jobs. Data is replicated in two places, so each task can be scheduled to either of those two locations.

	Job	DC1	DC2	DC3
Initial data location	J_1	d_1^1	$d_2^1, d_3^1, \cdots, d_5^1$	$d_6^1, d_7^1, \cdots, d_{10}^1$
	J_2	$d_1^2, d_2^2, \cdots, d_6^2$	d_7^2, d_8^2, d_9^2	d_{10}^2
	J_3	d_1^3	$d_2^3, d_3^3, \cdots, d_7^3$	d_8^3
Replica data location	J_1	d_7^1, d_8^1	d_9^1, d_{10}^1	$d_1^1, d_2^1, \cdots, d_6^1$
	J_2	d_{10}^2	$d_1^2, d_2^2,$	$d_3^2, d_4^2, \cdots, d_9^2$
	J_3	d_4^3, d_5^3, d_6^3	d_7^3, d_8^3	d_1^3, d_2^3, d_3^3

to denote whether the data for t_i^j is replicated on data center k. To prevent the expensive data transfer cost, a task can only be scheduled to a data center with its requested data. Therefore, our goal is to find a scheduling decision of tasks, so that the average completion time of jobs is minimized.

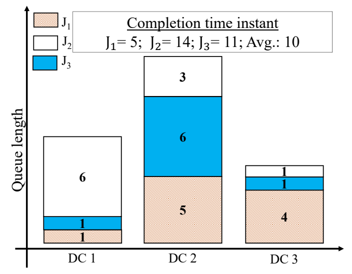
As an example shown in Table I, it represents a geo-distributed system consists of 3 data centers, and stores the data for 3 jobs. Because the data is replicated in two places, each task can be scheduled to either of those two locations. Say we consider a scheduling algorithm that simply places tasks to their initial data location, and execute jobs in the FCFS order: $\{J_1, J_2, J_3\}$. The completion time of J_1 on $DC_1, \cdots, DC_3$ is 1, 5, 4, respectively. So the completion time of J_1 is 5. J_2 is executed after J_1, so its completion time is 8 (=max(7, 8, 5)). Finally, we can get the completion of j_3 is 14. Hence the average completion time is 9. However, if we consider the same job scheduling order, but use the replica data locations, then the completion time of jobs becomes 6, 13, and 16 instead, and the average completion time increases significantly to 11.6.

As shown from the example, minimizing the average completion time needs to make two scheduling decisions: (1) Where to place the tasks of each submitted job? (2) In which order to execute the submitted jobs? Minimizing average completion time is known to be a NP-complete problem even when data is not replicated, and task placement has been fixed [15], [13]. Therefore, we propose a heuristic algorithm to make the two decisions in Section III.

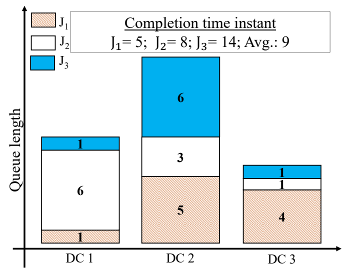
C. Examples

Here, we use the previous example in Table I to illustrate our scheduling problem by comparing the results from different algorithms. The first two algorithms, SRPT and SWAG, are not aware of data replication. So we assume they only place tasks at their initial data location, and minimize completion time by swapping job execution order. Then we show the completion time can be significantly improved if the task placement decision is also considered, and the scheduling algorithm needs to be re-designed to achieve better results.

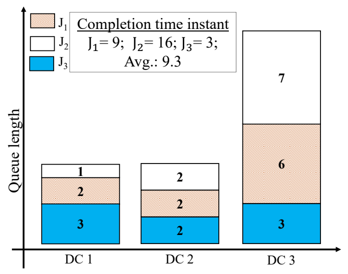
1) SRPT: The Shortest Remaining Processing Time algorithm decides the execution order of jobs based on their remaining processing time, which is determined by the maximum number of tasks among all the data centers. Thus, based on the initial data locations, the processing time are 5 for J_1, 6 for J_2, and 6 for J_3 in the example. The execution order is to sort the jobs by their remaining processing in ascending order, and use the total execution time as the tiebreaker. Hence, in



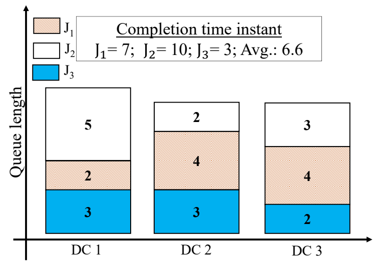
(a) SRPT



(b) SWAG using initial data location



(c) SWAG using replica data location



(d) Data Replication-aware (DRASH)

Job	DC1	DC2	DC3
J_1	$\mathbf{t^1_7}, \mathbf{t^1_8}$	$t^1_2, t^1_4, \mathbf{t^1_9}, \mathbf{t^1_{10}}$	$\mathbf{t^1_1}, \mathbf{t^1_3}, \mathbf{t^1_5}, t^1_6$
J_2	$t^2_1, t^2_2, t^2_4, t^2_6, \mathbf{t^2_{10}}$	t^2_7, t^2_8	$\mathbf{t^2_3}, \mathbf{t^2_5}, \mathbf{t^2_9}$
J_3	$t^3_1, \mathbf{t^3_4}, \mathbf{t^3_6}$	$t^3_2, t^3_5, \mathbf{t^3_7}$	$\mathbf{t^3_3}, t^3_8$

(e) Tasks placement by DRASH. Tasks that use replica data are marked in bold font.

Fig. 2: Scheduling results comparison between different scheduling algorithms and task placement decisions based on the system setting shown in Table I.

our example, the execution order of SRPT is: $\{J_1, J_3, J_2\}$, and its average completion time is 10 as shown in Fig. 2a.

2) SWAG [13]: It is a workload-aware greedy scheduling technique that prioritizes jobs execution among data centers. SWAG schedules jobs iteratively. To capture the workload, SWAG keeps track of the queue length of each data center. The queue length is initialized to 0 at the beginning, but it will be updated after the tasks of a job is scheduled to its data center. In each iteration, SWAG picks the job that will result in the lowest maximum queue length among all data centers. Again, SWAG is not aware of data replication, so we show its scheduling result under the initial data location in Fig. 2b. As shown in the figure, J_1 is selected first, because it has earliest completion time at SWAG5, while the others will finish at 6 instead. Then the queue lengths are increased to $\{1, 5, 4\}$ to reflect the workload caused by J_1. Next, J_2 is selected, because the maximum queue length will be 8. In contrast, if J_3 is selected before J_2, its maximum queue length will be 11 on DC_2. Therefore, J_3 should be executed at the end, and the average completion time from this scheduling algorithm is 9.

To understand how data replication can affect scheduling results, and why we need to design a replication-aware scheduling algorithm, Fig. 2c shows the scheduling results from SWAG if the tasks are scheduled on the replica locations instead. As shown, the average completion time will increases from 9 to 9.3. Since data replication is normally controlled by its data collection sources, and for fault tolerant purpose, we cannot assume the initial or primary data location will favor one particular scheduling algorithm and produce reasonable scheduling performance. Especially, in practice, the initial data placement distribution is more likely to be skewed and caused unbalanced workload and longer average job completion time.

3) DRASH: To addressed unbalanced workload and take advantage of the data replication, we introduce a data replication-aware schedule, DRASH. The algorithm is detailed in Section III, and here we first show its scheduling results in Fig. 2d. As shown from the figure, DRASH can significantly reduce the average completion time to 6.3, which is more than 30% less than any of previous results we showed. Comparing to the others, it is easy to see that DRASH can achieve better results because not only the load is balanced among data centers, but also the load among tasks is balanced for a job. More importantly, the reason DRASH can achieve more evenly balanced load because we are able to migrate the load among data centers by taking advantage of the replica data without transferring the data. As shown by the task placement decision of DRASH in Fig. 2e, some of tasks are scheduled to the data center with initial data locations, but others are scheduled to the replica locations. Therefore, clearly a much better scheduling decision can be found if the replica data is also considered. But the scheduling algorithm must intelligently decide the task placement and job ordering at the same time, and it is a NP-complete problem.

III. DRASH Scheduling Algorithm

As explained earlier in the example scenario, the problem is more complex on geo-distributed system and requires efficient tasks placement to benefit from the replica. Although significant improvement can be observed from existing algorithms, there is still a room for improvement as we presented in Fig. 2d. However, in order to avoid the high computation complexity discussed in problem definition section, our method combines the tasks placement and the job ordering in series of decisions which can be computed in linear time. Our approach to solve the problem is based of the following rationales:

Rationale 1: It is known the shortest job first scheduling algorithm can achieve the minimum average completion time when the problem has only a single server. Thus in our problem with multiple data centers, we also prefer to schedule the job with the minimum execution time first.

Rationale 2: The tasks from a job needs to be balanced among data centers. This is because the minimum completion time of a job is bounded by its maximum number of tasks on data centers. Therefore, balancing the tasks from a job can shorten its minimum execution time especially if the task distribution is skewed because of the initial data placement distribution. Also, balancing the load of job can help us balance the load between data center more easily for our third rationale.

Rational 3: The total load among data centers needs to be balanced as well. This is because the worst case job completion time is bounded by the maximum load among data centers, and the highly loaded data center can easily become the bottleneck for minimizing the completion time for every jobs. However, the load of a data center depends on the scheduling decision between jobs. Therefore, the scheduling algorithm must be aware of the current loading after each scheduling decision, and place a task to the data center that has the request data and with the minimum load.

Our algorithm is shown in Algo. 1, and it consists of two steps explained as follows.

Step1. Sort the jobs according their minimum execution time(rationale 1). The minimum execution time of a job is defined as the minimum completion time of a job when it runs alone without any other workload in the system. To minimize job completion time, we iteratively placing the tasks of a job to a data center by choosing the one with the lowest current load (rationale 2). Noted the we use q_i to record the set of tasks that have been scheduled on data center DC_i. The queues $Q = \{q_1, \cdots, q_N\}$ are reset after each job assignment in line9 because we only consider one job at a time in Step1.

Step2. According to the above job order, we then start iteratively placing the tasks of a job by minimizing the maximum load among data centers (rationale 3). Again we iteratively place the tasks of a job to a data center by choosing the one with the lowest current load. But different from Step1, here we keep the load q_i after each job assignment to reflect the accumulated load on data centers.

To illustrate our scheduling algorithm, we now revisit the results presented in Fig. 2d. For Step1, we take job J_1 as an example and explain its task placement results step-by-step as follows. First, task t_1^1 can be scheduled either on DC_1 or DC_3 according to the given data placements in Table I. We pick DC_1 for t_1^1 because currently both data centers have no load, and we use data center ID as a tiebreaker in this example. Then task t_2^1 is scheduled on DC_2 for the same reason. Next, task t_3^1 is scheduled on DC_3, because it can only be scheduled on DC_2 or DC_3, and the current load on DC_3 is 0 while the load on DC_2 is 1. After we schedule the rest of tasks for J_1, we will find t_1^1, t_4^1, t_6^1 on DC_1, t_2^1, t_7^1, t_8^1 on DC_2, and $t_3^1, t_5^1, t_9^1, t_{10}^1$ on DC_3. Therefore, the minimum execution e_1 for J_1 is 4. Similarly, we can get the minimum execution time J_2 and J_3 are 4 and 3, respectively. Again, we use job ID as

Algorithm 1 DRASH

```
1:  procedure DRASH($S, J, T, \mathcal{L}$)
2:  Input:
3:  $S$: data centers; $J$: jobs; $T$: tasks; $\mathcal{L}$: data placements
4:  Output:
5:  $Q = \{q_1, \cdots, q_N\}$: schedule queue of data centers
6:  //Step1: Sort jobs by their minimum execution time
7:      $e_i \leftarrow 0, \forall J_j \in J$
8:      for $J_j \in J$ do
9:          $q_d \leftarrow 0, \forall q_d \in Q$
10:         for $t_i^j \in J_j$ do
11:             $target \leftarrow min_d|q_d|, \forall DC_d \in S$ and $\mathcal{L}_{j,d}^i = 1$
12:             $q_{target} = q_{target} \cup t_i^j$
13:         $q_{max} \leftarrow max_d|q_d|, \forall q_d \in Q)$
14:         $e_j \leftarrow |q_{max}|$
15:     Sort $J_i \in J$ by $e_i$ in ascending order
16: //Step2: Place tasks on the data center with minimum load
17:     $q_d \leftarrow 0, \forall q_d \in Q$
18:     for $J_j \in J$ do
19:         for $t_i^j \in J_j$ do
20:             $target \leftarrow min_d|q_d|, \forall DC_d \in S$ and $\mathcal{L}_{j,d}^i = 1$
21:             $q_{target} = q_{target} \cup t_i^j$
22:     return Q
```

a tiebreaker in this example, so the final job order resulting from Step1 will be $\{J_3, J_1, J_2\}$.

Then in Step2, we start scheduling the tasks from jobs according to the job order above. The final task placement results are shown in Table 2e. Noted that the task placement decision of each job from Step2 is different from Step1, because now the initial queue length will depend on the previous job scheduling results. For instance, task J_1^1 is now scheduled on DC_3 instead of on DC_1, because the load from J_3 makes DC_3 to be less loaded than DC_1 when scheduling J_1^1. At the end, we did find our algorithm can evenly balanced the load among data centers, and therefore achieved a much lower average completion than other approaches.

IV. Performance Evaluation and Analysis

We built a prototype for DRASH to leverage the batch job scheduling service on a geo-distributed environment. We evaluate our scheduler with simulations. In this section, we describe its performance conducted under extensive evaluations on realistic production traces from Facebook workloads [16], [17] and a synthetic workload derived from a random generator. With these traces, we examined a wide range of environment parameter settings including the replication factor, the distribution of various size of data, the data distribution and the system utilization. We compared the results with the state-of-art SWAG strategy and a classical Shortest Remaining Processing Time based algorithm. Each of these algorithms is applied on a tasks placement based on the initial data location labelled as *Init.SWAG* and *Init.RSPT*; and a random placement labelled as *Rand.SWAG* and *Rand.RSPT*. Our key metric is the average completion time. Through the series of experiments, we demonstrate that DRASH can take advantage of the replicated data to significantly reduce the completion time.

TABLE II: General system setting

Heterogeneous distributed system		
data center	# of Nodes	# of cores per node
DC1	13	24
DC2	7	12
DC3	7	8
DC4	12	8
DC5	31	32
DC6	31	32
DC7	10	16
DC8	8	12

TABLE III: Settings of Job Traces

Trace Type	Avg. Job Size	Small jobs $(1-150)$	Medium jobs $(151-600)$	Large jobs (>601)
FB-2009	76.86	87.30%	8.55%	4.15%
FB-2010	157.12	48.12%	4.71%	47.18%
Synthetic	662.06	6.93 %	23.15 %	69.92%

A. Setup

System: The default system setting is composed of 8 data centers, each with a different number of nodes and cores per node as described in Table II. The capacity of a data center equals to the $\#nodes \times \#cores\_per\_node$. Unless specified, the classical 3-replica policy is adopted by default and replicated data are randomly placed in the data centers.

Workloads Traces: We used two traces from Facebook workloads: one (`FB-2010`) with the data input path and the other (`FB-2009`) without that information. In total, there are about 24000 jobs in each trace. These realistic traces have been used in previous works [13], [18]. The arrival rates are respectively $3.54s$ and $3.4s$ for `FB-2009` and `FB-2010`. The summary can be found in the Table III. Along with these traces, we randomly generate traces from a random generator to assess the performance under various tests settings and to show that DRASH can fit different workloads. By default, we randomly generated 1000 jobs following a uniform distribution. The initial data location follows also a uniform distribution by default and a Zipf distribution $\alpha = [0.001, 30]$ which indicated the skewness of data location on the data centers. Note that the larger α value means a higher skewness.

B. The Overall Performance Analysis

Here we compared the overall performance between SWAG, SRPT and DRASH and present the result in Fig 3. The results are normalized to our algorithm. Clearly, DRASH outperforms the two other methods by a large margin. On `FB-2009` traces presented on Fig. 3a, where the traces contain more mall jobs than the other traces, DRASH is $2.3\times$ faster than SWAG init., and $2.4\times$ faster as compared to RSPT. On the results shown in Fig 3b, DRASH performs even better on `FB-2010` traces with $(3.34 - 1)/3.34 = 70\%$ in general. In fact, `FB-2010` traces consist of large jobs and therefore greater opportunity for DRASH. In contrast, for not data-aware schedulers, a larger job size results in longer individual job completion time. This observation is confirmed by the results of the synthetic traces which consists of 69.9% of larger jobs. We also verified this behavior with uniform distributed tasks. Although DRASH still performs best, the margin is considerably reduced to almost 16%, while SWAG and SRPT showed no progress. These results indicate that DRASH takes more advantage of the replica.

C. System Utilization

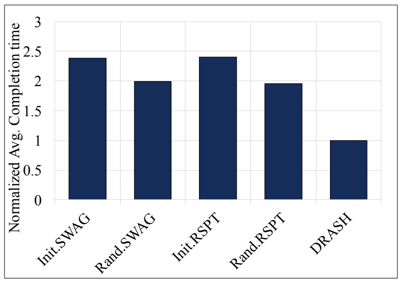
We measured and reported the performance of the algorithms on Fig. 4 at different system utilization. To this end, we increase the job inter-arrival rate such that at any given time instant, we can reach a certain system utilization. The x-axis represents the system utilization computed as a ratio of workload divided by the system capacity. The results in the Fig. 4a show that as the system utilization increases, the variance of the average completion time produced by DRASH remains small. However, for SWAG and SRPT as the utilization increase the resulting completion time increases almost linearly. That is because, when we increase the load it becomes more difficult to achieve the scheduling decisions. A higher system utilization favors our algorithm because as the load increase the entire system requires a better tasks placement. Unfortunately, in higher utilization SWAG and SRPT generates more imbalanced load between data centers during the tasks placement. More importantly, at a small system utilization our algorithm already performs good; at a heavy load it performs even better due to the placement decision using the replica.

D. Data Characteristic Analysis

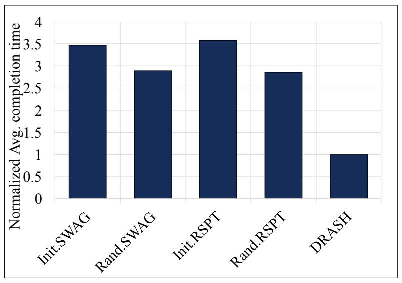
In practice, data are produced or collected by different techniques with their own objective functions. Therefore, it is not always possible to accurately predict or set the initial data location. To capture various initial data location and to assess the performance of the algorithms under different initial conditions, this section evaluates the initial data characteristic. We used different initial data skewness and present the result in Fig. 5. The skewness factor $\theta = [\frac{1}{1000}, 0.1]$ results in a uniform distribution, while any value higher than 1 results in a skewed distribution (at $\theta = 30$, almost all the data are initially produced in one data center). Both *Rand.SWAG* and *Rand.SRPT* algorithms perform well on skewed distribution when exploiting the data replication. However, as seen in Fig. 5a and Fig. 5b, there is a longer completion time ($3 \sim 4\times$ longer) when initial data become more and more skewed. In contrast, DRASH performs better for highly skewed initial data. This stability demonstrates that DRASH can adapt to different workload setting. Ultimately, the skewness benefits more our algorithm because there is a higher chance to balance the load on the other data centers hosting the replicas. The experiments on the synthetic workload presented in Fig. 5c further show that DRASH converges to a stable value as the skewness factor increases. Thus, regardless the initial data location or the job characteristics, our algorithm still performs with a larger margin.

E. Data Replication Analysis

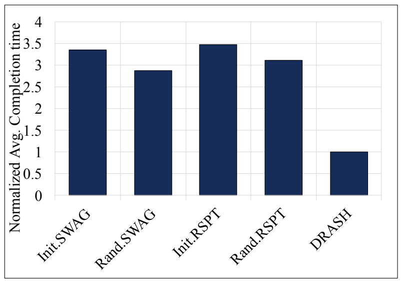
In these experiments, we are interested in the impact of the number of replica on our algorithms. In Fig. 6, we conducted the experiments on different replication factors and present the results of two versions: in Fig. 6a the jobs are uniformly distributed and in Fig. 6b a skewed version is presented. The x-axis represents the percentage of replication on the total number of data centers in the system. As expected, when the number of replica is increased, the average completion time



(a) Performance of Facebook 2009 Trace

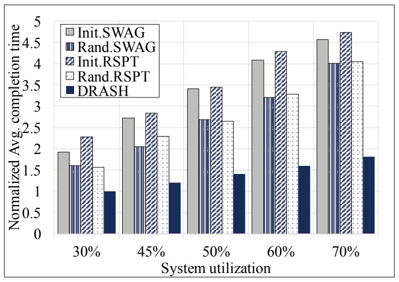


(b) Performance of Facebook 2010 Trace

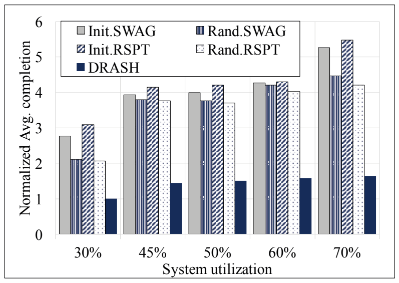


(c) Performance of exponential synthetic Trace

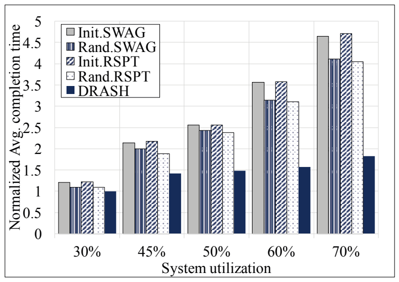
Fig. 3: Performances of Average execution time with different workloads normalized to DRASH



(a) Performance of Facebook 2009 Trace

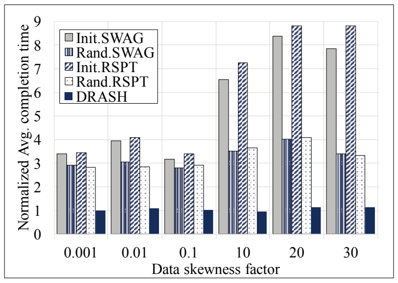


(b) Performance of Facebook 2010 Trace

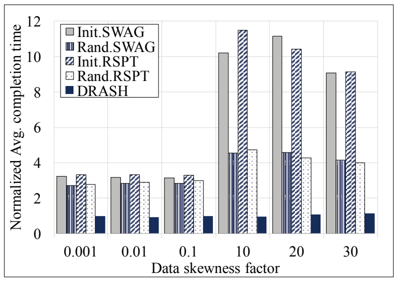


(c) Performance of exponential synthetic Trace

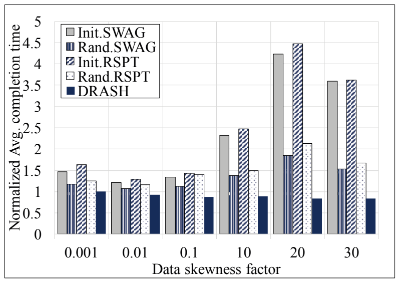
Fig. 4: Performances of Average execution time (normalized to DRASH 30%) with different resource utilization



(a) Performance of Facebook 2009 Trace



(b) Performance of Facebook 2010 Trace



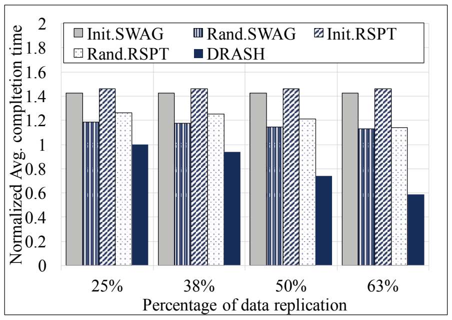
(c) Performance on exponential synthetic Trace

Fig. 5: Analysis on data skewness (normalized to DRASH 0.001)

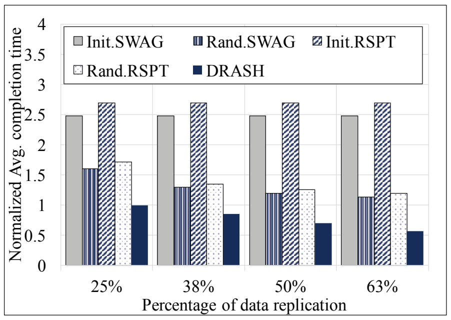
is considerably reduced for both cases. On average before 50% of data replication, DRASH is $1.3\times$ faster; above 50% it goes up to $2.7\times$ faster than SWAG and RSPT, for each data replicated on one additional data center. With skewed jobs, both of SWAG and RSPT performance is significantly improved as well. Data is often expected to be skewed, random placements result in a better schedule than the initial placement for SWAG and RSTP, however DRASH still outperforms both algorithms significantly. All these results demonstrate that if the scheduling technique includes data locality and harness the replicated data, average completion time can be significantly be reduced, and that is main purpose of DRASH.

V. Related Works

The related work can be viewed from three perspectives. **Scheduling on Geo-Distributed Systems**: Branches of studies on geo-distributed datacenters include data grid[9], [10], [19], [20]. Some of the scheduling techniques on geo-distributed systems have considered data locality in their approach. One of the closest studies to our work was recently conducted by Hung et al [13]. They proposed a system where tasks are dispatched to the data centers hosting the data. However, they did not consider data replication. L. R. Anikode et. al in [9] designed an integrated algorithm for both the placement of the replica and the tasks. It considers both remote and local access to data. They proved theoretically that the performance of applications can significantly improve when considering the different data hosted by different data centers. However, they use data migration for locality and therefore differ from our work. In our work, data and replicas are already given and we avoid data migration during scheduling since it incurs a high network penalty [21], [10]. In order to improve the scheduling performance C. L. Abad et. al [11] proposed an adaptive replication model. Similar to [9], they do not consider fixed data location.
Data-intensive Scheduling: MapReduce [4], Hadoop [5], Spark [6] and Dryad [7] are common platforms for large-scale data-intensive processing. However, the current implementation of these models still requires addition efforts in adapting them to fit multiple dispersed data centers [22]. Therefore,



(a) Uniform dist. data



(b) Skewed data

Fig. 6: Performance gain over increasing replication replication factor (normalized to DRASH 25%)

they are not applicable to geo-distributed systems because their application jobs require frequent communication and transfer of intermediate data among clusters. This will result inevitably in a huge network bottleneck [23]. Therefore, applying a MapReduce architecture directly to a geo-distributed data-intensive model will be counterproductive.

Makespan minimization Scheduling: There are also previous works on improving the makespan (i.e., completion time) that may be thought as alternatives solution for scheduling on a geo-distributed system. A representative class of them have been used in our example scenario [24]. The objective of these scheduling techniques is to minimize the overall completion time or schedule length of the parallel program. Y.-K. Kwok and I. Ahmad [25] proposed a dynamic scheduling technique for assigning tasks to different processors so as to minimize the makespan. Another class of algorithms [26] dynamically selects tasks during the scheduling tasks. However like most multicore scheduling algorithms, these techniques only consider compute intense problem and therefore are oblivious to data locality.

To some extent, there are few efforts on adapting the use of existing and fixed data replicas to the geo-distributed environment which is common in today's cloud environment. In contrast to existing work, we focus on finding the tasks placement that take advantage of the replica to minimize the completion time.

VI. Conclusion

In this work, we have proposed a novel data replication-aware scheduling algorithm which can leverage the existing replicas to minimize the average completion time of job submitted to a geo-distributed system. As it has become a de-facto approach for cloud applications to replicate their data across data centers to prevent data loss and guarantee service availability, there is a need of adapting the scheduling techniques to these geo-distributed data centers. Our method consists of prioritizing the data centers based on the data hosted. Then we attempt to balance the load of each individual job so as to minimize the makespan. Finally, we used an approach similar to a water-filling algorithm to place the tasks. Our evaluations using realistic workload traces show that DRASH can outperform other existing geo-distributed job scheduling approaches by 16% to 60% in average job completion time, and achieve greater performance when more replications are available.

Acknowledgement

This study is conducted under the "Big Data Technologies and Applications Project (2/4)" of the Institute for Information Industry which is subsidized by the Ministry of Economic Affairs of the Republic of China.

References

[1] "AWS: Amazon Web Service," 2006. [Online]. Available: http://aws.amazon.com

[2] "Microsoft Azure," 2010. [Online]. Available: https://azure.microsoft.com/

[3] "Google Compute Engine," 2011. [Online]. Available: https://cloud.google.com/compute/

[4] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[5] "Apache Hadoop Project," 2013. [Online]. Available: http://hadoop.apache.org

[6] "Apache Spark," 2013. [Online]. Available: http://spark.apache.org/

[7] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2007 Eurosys Conference*. Lisbon, Portugal: Association for Computing Machinery, Inc., March 2007. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=63785

[8] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 29–42. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855741.1855744

[9] L. R. Anikode and B. Tang, "Integrating scheduling and replication in data grids with performance guarantee," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, Dec 2011, pp. 1–6.

[10] F. Jolfaei and A. T. Haghighat, "The impact of bandwidth and storage space on job scheduling and data replication strategies in data grids," in *Computing Technology and Information Management (ICCM), 2012 8th International Conference on*, vol. 1, April 2012, pp. 283–288.

[11] C. L. Abad, Y. Lu, and R. H. Campbell, "Dare: Adaptive data replication for efficient cluster scheduling," in *2011 IEEE International Conference on Cluster Computing*, Sept 2011, pp. 159–168.

[12] M. Zarina, F. Ahmad, A. N. bin Mohd Rose, M. Nordin, and M. M. Deris, "Job scheduling for dynamic data replication strategy in heterogeneous federation data grid systems," in *Informatics and Applications (ICIA),2013 Second International Conference on*, Sept 2013, pp. 203–206.

[13] C.-C. Hung, L. Golubchik, and M. Yu, "Scheduling jobs across geo-distributed datacenters," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, ser. SoCC '15. New York, NY, USA: ACM, 2015, pp. 111–124.

[14] A. N. Toosi and R. Buyya, “A fuzzy logic-based controller for cost and energy efficient load balancing in geo-distributed data centers,” in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, Dec 2015, pp. 186–194.

[15] N. Garg, A. Kumar, and V. Pandit, “Order scheduling models: Hardness and algorithms,” in *Proceedings of the 27th International Conference on Foundations of Software Technology and Theoretical Computer Science*, ser. FSTTCS’07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 96–107. [Online]. Available: http://dl.acm.org/citation.cfm?id=1781794.1781804

[16] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, “Reining in the outliers in map-reduce clusters using mantri,” in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI’10. Berkeley, CA, USA: USENIX Association, 2010, pp. 265–278. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924962

[17] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, “Effective straggler mitigation: Attack of the clones,” in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL: USENIX, 2013, pp. 185–198. [Online]. Available: https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/ananthanarayanan

[18] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, “The case for evaluating mapreduce performance using workload suites,” in *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, July 2011, pp. 390–399.

[19] N. Sooezi, S. Abrishami, and M. Lotfian, “Scheduling data-driven workflows in multi-cloud environment,” in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov 2015, pp. 163–167.

[20] Y. C. Lee and A. Y. Zomaya, “Practical scheduling of bag-of-tasks applications on grids with dynamic resilience,” *IEEE Transactions on Computers*, vol. 56, no. 6, pp. 815–825, June 2007.

[21] W. Li, Y. Yang, and D. Yuan, “A novel cost-effective dynamic data replication strategy for reliability in cloud data centres,” in *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, Dec 2011, pp. 496–502.

[22] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen, and D. Chen, “G-hadoop: Mapreduce across distributed data centers for data-intensive computing,” *Future Generation Computer Systems*, vol. 29, no. 3, pp. 739 – 750, 2013, special Section: Recent Developments in High Performance Computing and Security. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X12001744

[23] X. Liao, Z. Gao, W. Ji, and Y. Wang, “An enforcement of real time scheduling in spark streaming,” in *Green Computing Conference and Sustainable Computing Conference (IGSC), 2015 Sixth International*, Dec 2015, pp. 1–6.

[24] L. Schrage, “A proof of the optimality of the shortest remaining processing time discipline,” *Operations Research*, vol. 16, no. 3, pp. 687–690, 1968. [Online]. Available: http://www.jstor.org/stable/168596

[25] Y.-K. Kwok and I. Ahmad, “Fastest: a practical low-complexity algorithm for compile-time assignment of parallel programs to multiprocessors,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 2, pp. 147–159, Feb 1999.

[26] G. C. Sih and E. A. Lee, “A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 2, pp. 175–187, Feb 1993.