Computing (2018) 100:21–46
https://doi.org/10.1007/s00607-017-0564-7

GEODIS: towards the optimization of data locality-aware job scheduling in geo-distributed data centers

Moïse W. Convolbo[1] · Jerry Chou[1] · Ching-Hsien Hsu[2,3] · Yeh Ching Chung[1]

Received: 17 November 2016 / Accepted: 20 June 2017 / Published online: 20 July 2017

Abstract Today, data-intensive applications rely on geographically distributed systems to leverage data collection, storing and processing. Data locality has been seen as a prominent technique to improve application performance and reduce the impact of network latency by scheduling jobs directly in the nodes hosting the data to be processed. MapReduce and Dryad are examples of frameworks which exploit locality by splitting jobs into multiple tasks that are dispatched to process portions of data locally. However, as the ecosystem of big data analysis has shifted from single clusters to span geo-distributed data centers, it is unavoidable that data may still be transferred through the network in order reduce the schedule length. Nevertheless, there is a lack of mechanism to efficiently blend data locality and inter-data center data transfer requirement in the existing scheduling techniques to address data-intensive processing across dispersed data centers. Therefore, the objective of this work is to propose and solve the makespan optimization problem for data-intensive job scheduling on geo-distributed data centers. To this end, we first formulate the task placement and the data access as a linear programming and use the GLPK solver to solve it. We then present a low

✉ Ching-Hsien Hsu
chh@chu.edu.tw

Moïse W. Convolbo
wendkuuni@sslab.cs.nthu.edu.tw

Jerry Chou
jchou@cs.nthu.edu.tw

Yeh Ching Chung
ycchung@cs.nthu.edu.tw

1 National Tsing Hua University, Hsinchu, Taiwan

2 School of Mathematics and Big Data, Foshan University, Foshan, China

3 Chung Hua University, Hsinchu, Taiwan

complexity heuristic scheduling algorithm called GeoDis which allows data locality to cope with the data transfer requirement to achieve a greater performance on the makespan. The experiments with various realistic traces and synthetic generated workload show that GeoDis can reduce makespan of processing jobs by 44% as compared to the state-of-the-art algorithms and remain within 91% closer to the optimal solution by the LP solver.

Keywords Geo-distributed · Data center · Scheduling · Data locality · Batch jobs · Big data analysis

Mathematics Subject Classification 90C05 Linear programming · 90C27 Combinatorial optimization · 90C46 Optimality conditions, duality

1 Introduction

The recent years have witnessed the proliferation of data analytic application jobs due to the growing demand for processing the increasing amount of data generated and stored in data centers. In response, frameworks such as MapReduce [14], Hadoop [19], Spark [49], Dryad [24], etc, have been developed and efficiently used in large clusters for massive data processing. For example, MapReduce schedules jobs based on data locality; which means that processing jobs are split into multiple tasks and dispatched to the node hosting the data to be processed in order to minimize the network overhead [18]. Therefore, data locality aware scheduling improves the individual job response time within a cluster and avoid unnecessary data movement which is considered as the primary reason for delays in completion time [38,54,55].

However, as data keep increasing in size and diversity, organizations start to utilize geographically distributed data centers. In fact, Infrastructure as a Service (IaaS) cloud providers like Amazon [3], Microsoft [41], Google [17] and Rackspace [45] continue to build many data centers across different locations in order to support their organization structures in a modular growth manner. In addition, the rise of new multi-cloud systems as a solution to vendor locking and security has made geo-distributed data centers a prominent platform to deploy applications. Existing data grid systems from the grid scientific communities [53] are also using geo-distributed data centers to support data-intensive applications. Although data locality on clusters have been intensively studied in the past, it has not been applied to geo-distributed data centers until recently [22, 23]. The recent advent of big data has contributed to reaching a data deluge were information generated and collected is overwhelmingly exceeding the capacity of institutions to manage and make use of it within a single data center [29,43,55]. However, it has been shown in a previous study [22] that enforcing locality can degrade the application performance on geo-distributed data centers. In addition, data may be collected and stored in a few number of data centers following different cost and availability constraints, but required to be processed in other locations. Therefore, it is unavoidable that some data must be remotely transferred from one data center to another during the data processing time. A major challenge in data-driven processing is to reduce the impact of data transfer which increases the computation time. Other works

have attempted to use Hadoop MapReduce across geo-distributed data centers. Yet, applying this paradigm directly on heterogeneous interconnected data centers results in huge performance degradation [9]. Consequently, while geo-distributed systems, can be viewed as an opportunity to solve large-scale problems, they require different approaches to resource management for the various applications.

In this paper, we propose a locality-and-network aware scheduler for optimizing the performance of a data analytic frameworks on geo-distributed systems. The input of our scheduling algorithm is a set of jobs submitted into the computing framework. The data requested by the tasks is replicated on multiple data centers, and the location is known to the scheduler. The objective of our scheduling algorithm is to decide where to place these tasks and which replica to access for each task, such that the makespan of job executions is minimized where the makespan of a data analytic job is governed by the completion of the last task on any data center. It is a challenging problem, and the system performance can be significantly affected the scheduling decision for the following reasons.

1. Minimizing makespan on heterogeneous system is known to be a NP-complete problem even when data is not replicated, and task placement is given [16,23].
2. With multiple replicas and different inter-data centers network bandwidth, the transfer time can increase significantly if the data access location is not properly selected.
3. Restricting tasks running on their data location can eliminate data transfer time, but could also suffer from imbalanced workload. Thus, the tradeoff between data locality and data transfer present a dilemma for finding the optimal solution.

In this paper, we made the following contributions.

- To our knowledge, we are the first to consider both data locality (with replication) and data migration together for optimizing data analytic workload executions on geo-distributed data centers.
- We formulated our scheduling problem into a LP optimization problem, and obtain the optimal solution using LP solver as a comparison baseline to our proposed heuristic algorithm.
- We proposed an online heuristic algorithm, GeoDis, which has linear algorithm complexity, and achieves close to optimal results.
- We extensively evaluated our approach using a number of real workload traces and various placement models to understand the performance impact and justify our design decisions.
- We finally demonstrate the robustness of our solution on heterogeneous environment settings with different processing capability in each data center and various network bandwidth settings to show the applicability of GeoDis for a production-level cloud platform.

The remainder of this paper is organized as follows: Sect. 2 reviews the related work in the literature. Section 3 describes our geo-distributed system model. We formulate our job scheduling problem and propose scheduling algorithms in Sects. 4 and 5, respectively. The experimental results are presented in Sect. 6, and the paper is concluded in Sect. 7.

2 Related work

In this section, we present a brief review of scheduling techniques on geo-distributed data centers. The related work can be viewed from three perspectives.

2.1 Scheduling on geo-distributed systems

Early examples of branches of studies similar to geo-distributed datacenters include data grid [6,28,32,48]. Data grids provide a dispersed distributed storage resources for large and complex computational problems. Data grids focus on efficient management including mechanism for controlled sharing and processing large amounts of distributed data. Similar to geo-distributed scheduling, the grid scheduler decides where to run the job based on the requirements and the system availability. An effective schedule in data grids consists of placing the job such as to minimize data movement which is time consuming. Data replication became quickly the main focus in data grid [15]. For example Zarina et al. [57] proposed three replication techniques to minimize the file transfer among sites. However, it is impractical to replicate all data in every location. Other techniques combining tasks scheduling and replica placement were proposed by Anikode and Tang in [6]. 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 sites. However, they assume data can be replicated at the schedule time. In practice, data collection and replication are often for availability and safe guarding purpose and therefore are often fixed before hand [28,36]. Despite the similarities between data grids and geo-distributed data centers, the latter includes more complex tradeoffs making existing solutions in data grids inefficient when they are applied to the new and challenging big data environments.

One of the closest studies to our work was recently conducted by Hung et al. [23]. They proposed a system where tasks are dispatched to the data centers hosting the data. However, they did not consider data replication neither data migration. In order to improve the scheduling performance Abad et al. [1] proposed an adaptive replication model. Similar to [6], they do not consider fixed data location. Tudoran et al. [52] proposed a uniform data management system for scientific workflows. Their system works across geographically dispersed data centers and consists of a set of pluggable services which provides flexibilities to monitor the infrastructure, data compression and geo-replication. The goal is to present the tradeoff between the monetary cost and the time and to optimize the data migration strategy according to the objective function.

2.2 Data-intensive scheduling

MapReduce [14], Hadoop [19], Spark [49] and Dryad [24] are common platforms for large-scale data-intensive processing on clusters. However, the current implementation of these models still requires addition efforts in adapting them to fit multiple dispersed data centers [56]. For example, Cardosa et al. [9] have demonstrate that Hadoop

MapReduce performs very poorly on geo-distributed data centers interconnected with heterogeneous network. Recently, there has been a humbling amount of work on adapting MapReduce to fit geo-distributed data center environment [10,11,20,30,34,35]. It clearly means that applying a MapReduce architecture directly to a geo-distributed data-intensive model will be counterproductive [37]. MapReduce implementations are originally designed for homogeneous environments. Previous works have demonstrate that heterogeneity even at a single data center level can cause significant performance deterioration in job execution, despite existing optimizations on task scheduling and load balancing. In the context of geo-distributed data centers, it is even worst because in addition to the heterogeneity in each data center (because each data center changes and upgrades its servers and network facilities and creates more heterogeneous environment), the network interconnection between geographically distributed data centers suffers also from the same heterogeneous bandwidth latencies. Some attempts to optimizing MapReduce for heterogeneous environment can be found in the literature. For example, Cheng et al. [13] uses meta-heuristic to optimize the tasks placement by finding the best configuration in the heterogenous cluster. Li et al. [35] propose to improve MapReduce cluster deployment in geo-distributed data-center. However, the simulation of geo-distributed HDFS based on their model still suffers from network heterogeneity latencies. Li et al. [33] studied big data processing on geo-distributed data center using MapReduce by predicting the job completion time. Their model include the input data movement over the inter-data center network however they assume the intermediate data size and shuffle time are known in advance. In practice, when a node fails, Hadoop will find an available worker to reschedule the job; both the size of the intermediate data and their location can change and will require the optimization algorithm to rerun. Although, there is a large amount of research efforts to improve MapReduce for geo-distributed data centers, it does seem to be a trivial endeavor. Delays of data transfer between the nodes in geo-distributed due tu heterogeneity in the data centers architecture cause performance degradation in MapReduce [40,44].

2.3 Makespan minimization scheduling

There are also previous works on improving the makespan (i.e., completion time) that may be thought as alternative solutions for scheduling on a geo-distributed system. A representative class of them have been used in our example scenario [46]. The objective of these scheduling techniques is to minimize the overall completion time or schedule length of the parallel program. Kwok and Ahmad [31] proposed a dynamic scheduling technique for assigning tasks to different processors so as to minimize the makespan. Another class of algorithms [47] dynamically selects tasks during the scheduling tasks. However like most multicore scheduling algorithms, these techniques only consider compute intense problems 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 big data processing environment. Conventionally, the state-of-the-art schedulers on geo-distributed system focus on locality only and dealing with data migration as secondary.

This is because data transfer time has been a bottleneck for distributed systems in general. However, the recent improvement of network infrastructures and topologies, the routing systems and data distribution have improved the data transfer time [42]. In contrast to existing work, we focus on finding the tasks placement that take advantage of the replica and data transfer to minimize the makespan while providing a online scheduler with lower complexity.

3 Background and system model

Public cloud providers such as Amazon, Google and Microsoft have made available a plethora of services supported by distant data centers. Infrastructure as a Service (IaaS) cloud allows users to scale their resources in each of their geo-distributed data centers to support their application demands. In each location, users can acquire virtual machines (VMs) to form a virtual cluster for their needs. Since big data acquisition and storage often exhibit different patterns in terms of volume and frequency, users can decide the capacity in terms of number of processing power and storage in each available data center. Finally, most public data centers rely on the internet service for inter-data center traffic. Such environment is common nowadays and it is significant to address and evaluate the impact of scheduling techniques on data driven application.

3.1 Scheduling in geo-distributed systems

In this paper, we consider data analytic jobs scheduling problem on geo-distributed data centers. We consider a general system comprised of multiple inter connected and dispersed data centers. A data center has available computing resources denoted as the processing capacity. To be more specific, the capacity is the summation of the number of nodes times the number of cores per node.

3.1.1 Data model

Data are produced and stored in different data centers. Moreover, 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. However, the complexity, the volume and the cost of storage made it impractical to replicate the data in all data centers. Therefore, it is common for data to be partially replicated. We assume the initial data placement is given as input of our problem.

3.1.2 Job model

Jobs discussed in this paper are composite units of data analytic applications submitted as batches to process data across data centers. As opposed to interactive jobs, the batch jobs often exhibit a specific characteristic which does not require user intervention. Massive data processing jobs are one class of these applications. A job is associated with an arrival time and a number of divisible sub-jobs/tasks based on the data it processes.

3.1.3 Task model

Applications based on data processing models often split jobs into many tasks. Each task reads a portion of input data and processes it on an available computing resource. Therefore, tasks are assumed to be independent. The execution time of a given task is identical in each data center for the same type of resource. However, for a task to start, the required data must first be transferred in the local storage of the data center running it. Moreover, the job's completion time is determined by its last task to complete in any of the data centers.

3.1.4 Scheduling model

A job scheduler in geo-distributed data centers has two main responsibilities: First, dispatch tasks onto the multiple data centers and second, coordinate the execution of tasks from their respective data center locations. The dispatcher decides on which data center location to run the tasks. The coordinator indicates the order of execution of the tasks in each local data center. In a system where jobs are frequently submitted, a scheduler should periodically perform these two operations. One solution is to reschedule upon the arrival of a job. However, this option results in a high scheduling overhead, especially when small jobs are frequently submitted. Another option is to set a time window, (let's say every 5 min) where the arriving jobs are first put in a queue and scheduled at once. This option also leads to a potential idle time since some data centers might complete their workload before the time window. Finally, the scheduler can be triggered by any data center which completes it workload. In dispersed data centers environment, the last solution can significantly reduce the communication overhead between data centers.

3.2 System model

Figure 1 depicts our system model. The system consists of multiple data centers at different geographical locations. All submitted jobs arrived first, 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 such as processing data from E-commerce websites, social media and IoT sensors often produce and store data locally in the closest data centers. However, these data may be replicated on other data centers following different cost, availability and business requirement. We follow the same model and assume that the initial data location is given as well as the replicated data. We assume users submit jobs to process data distributed across geo-distributed data centers. These jobs consist of divisible tasks that can process a portion of the data. Tasks from the same job can run in any data center. However, before a task can be executed, its required data is remotely copied through the network to the local storage first. We refer to the time required to copy the data as the inter data center *transfer time*. If the task is scheduled in a data center hosting its required data, the inter data center transfer time equals zero. In this model, we assume the tasks process a different amount of data, and the execution time is

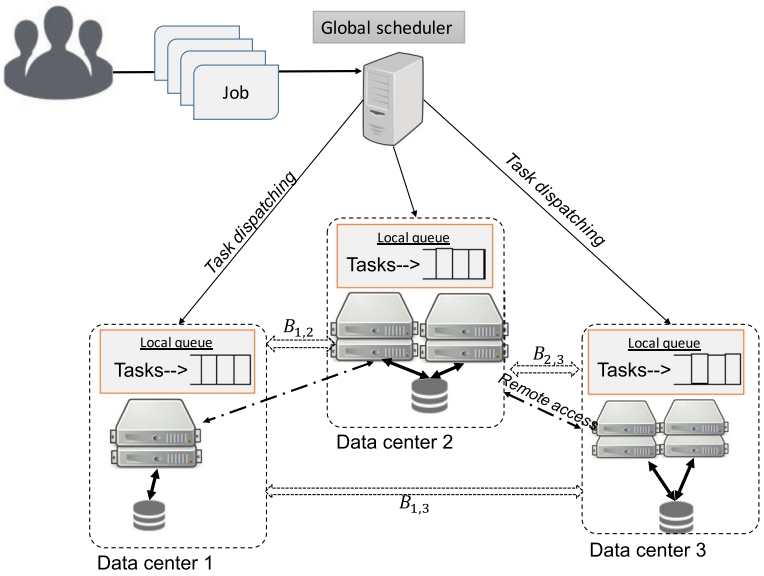

Fig. 1 System architecture of geo-distributed data center scheduling with data replication framework

proportional to the amount of data processed by the tasks [51]. This assumption is considered reasonable and can be estimated using profiling [21,25]; moreover, it has been applied to previous work including [26,27]. A local task queue q_i is maintained by the data center i for the tasks allocated to that particular data center. The completion of a job is governed by the completion of its latest task. A local scheduler in each data center reports the progress of the local queue to the global scheduler for the assigned tasks. We assume that a meta-data containing the data hosted by each data center and that the network bandwidth between the data centers are available and accessible by the global scheduler. Note that this model is relevant to common IaaS cloud providers' systems. For example, Amazon Web Service (AWS) [3] provides data centers in many 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 [23,50].

3.3 Motivational example

To illustrate the motivation of our work and demonstrate how complex the problem is, we consider a simplified example of a system consisting of 3 data centers hosting 8 data ($f1 \sim f8$) dispersed among the data centers. We assume that each data center only processes one task at a time and each resource can process 15 MB of data per second. The bandwidths among data centers are presented on Table 1a. The initial data location and where the replicas are placed are given in Table 1b. All the files containing the data to be processed are assumed equal size of 15 MB. We want to split a job into 8 tasks ($T1 \sim T8$) to process the data. Note that a task Tx will process one copy of the

Table 1 Example settings

	DC1	DC2	DC3
(a) Network bandwidth (Mbs)			
DC1	–	50	150
DC2	50	–	100
DC3	150	100	–

DC1	DC2	DC3
(b) Initial data location and replicas (**in bold**)		
$f1, f2$	$f3$	
$f5, f7, f8$	$f6$	$f4$
	f1	**f1**
f6		**f6**

data fx. We present three common schedules in geo-distributed data centers: Fig. 2a is a representative schedule for a class of schedulers which favor the tasks placement on the initial data location. They are oblivious to the replicas [23]. The schedule length in this case is 5 s. The second class of schedulers exploits the replica to minimize the makespan as shown in Fig. 2b, the task $T1$ can be placed in $DC1$, $DC2$ or $DC3$ since both contain a copy of $f1$. By scheduling the $T1$ in $DC3$, the processed file is the replica in $DC3$. The resulting schedule length is 4 s which represents $(5-4)/5 = 20\%$ improvement. The last class of schedulers presented in Fig. 2c allows tasks to be placed anywhere in the system. However, before the tasks can be executed, the required data might first be transferred to the local storage. For instance, if the task $T8$ uses the initial location of $f8$, the job will complete at time instant 4. However, if we allow migration, $T8$ can be scheduled either in $DC2$ or $DC3$. The bandwidth between $DC1$ and $DC2$ is 100 and 150 MB between $DC1$ and $DC3$. Therefore, $T8$ is placed on the fastest link. As a result, it will require $\frac{15}{150} = 0.1$ s to transfer $f8$. The completion time of the job is 3.1 s. Even with a very simplified example, allowing data to be transferred can reduce the makespan up to $(5 - 3.1)/5 = 38\%$. However, as we discussed later, the decision on whether to use the replica or the initial data and where to place the task and to transfer the data from should be carefully made in order to minimize the makespan. In the following section, we first formulate the scheduling problem then propose a solution for this type of problem which is common in big data analysis.

4 Problem definition

This section presents the problem definition. We start by showing that the targeted problem can be formulated into a linear programming problem, and an optimal scheduling solution can be found using LP solvers such as GLPK [39].

4.1 Problem input

Our scheduling problem has the following input arguments:

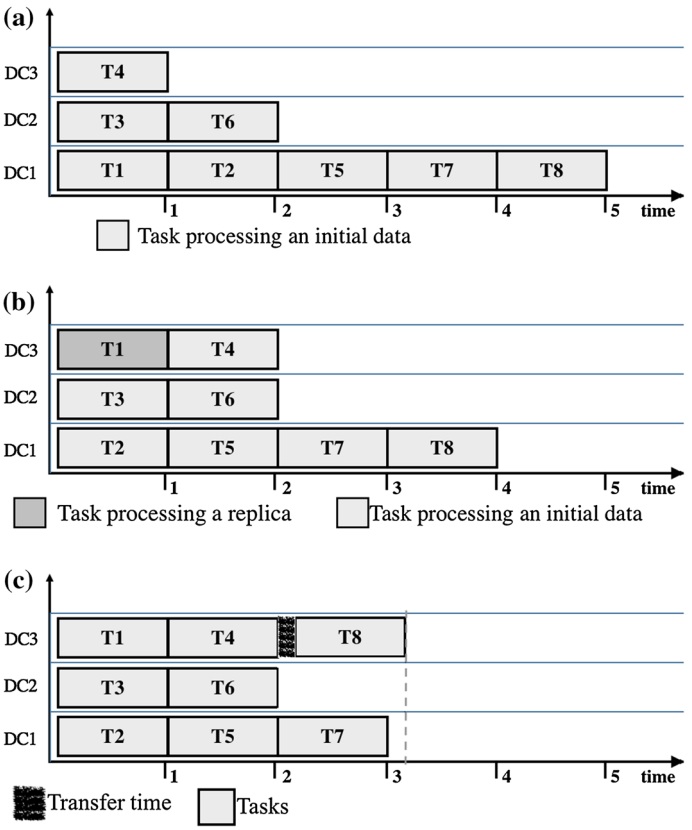

Fig. 2 Example of different schedules on geo-distributed data centers. **a** Example of a schedule on first copy of data. **b** Example of a schedule where tasks are forced to data locality. **c** Example of a schedule with data transfer, $T1$ uses the replica of $f1$ and $T8$ is transferred from $DC1$ to $DC3$

System environment: A geo-distributed system consisted of N data centers, denoted by $\mathbf{DC} = \{1, \ldots, N\}$. The computation capacity of a data center i is denoted as C_i. The network bandwidth between any two data centers i and i' is also given and denoted by $B_{i,i'}$.

Job description: $\mathbf{J} = \{1, \ldots, M\}$ denotes the set of jobs currently waiting to be scheduled by the global scheduler. A job is further divided into a set of K independent data processing tasks, so that the tasks of job j is denoted by $T_j = \{t_1^j, \ldots, t_K^j\}$, where t_k^j is the kth task from job j. The computation workload of each tasks is assumed to be known prior to scheduling, and denoted as w_k^j for task t_k^j.

Data description: A set of data stored in the system denoted by $D = \{d_k^j, \forall t_k^j\}$, where d_k^j is the data required to be processed by the task t_k^j. For a data, d_k^j, its size is s_k^j, and its replicated location is denoted by $L_{j,k}^i$, which is set to 1 if data d_k^j is replicated at data center i, otherwise it is set to 0. Hence for instance, in a system with 3 replica per

Table 2 Summary of the symbols used in this paper

Type	Variable	Description
Input	$\mathbf{DC} = \{1, \ldots, N\}$	A geo-distributed system which consists of N data centers
	$\mathbf{J} = \{1, \ldots, M\}$	The system has M jobs waiting to be scheduled
	$\mathbf{T} = \{t_k^j \mid j \in J, k = 1 \sim K\}$	Each job is consisted of K tasks, where t_k^j refers the kth task of job j
	$\mathbf{D} = \{d_k^j \mid j \in J, k = 1 \sim K\}$	A set of data stored in the system, where d_k^j is the data required by task t_k^j
	w_k^j	Computation time of task t_k^j
	s_k^j	Size of data d_k^j
	C_i	Computation capacity of data center i
	$B_{i,i'}$	Network bandwidth between data center i and data center i'. $B_{i,i'} = \infty$ if $i = i'$.
	$L = \{L_{j,k}^i \mid \forall i = 1 \sim N, j = 1 \sim M, k = 1 \sim K\}$	A data placement where the value is set to 1 if data d_k^j is replicated at data center i; otherwise it is set to 0
Output	$S = \{S_{j,k,x}^i \mid \forall i, x \in \mathrm{DC}, t_k^j \in T\}$	A scheduling decision, where the value of $S_{j,k,x}^i$ is set to 1 if task t_k^j is scheduled to data center i, and access its data from data center x
Derived variable	E_i	Total execution time on data center i
	T_S	The makespan of a schedule decision S

data, the assignment of data replication will be restricted by $\sum_i L_{j,k}^i = 3, \forall d_k^j \in D$ (Table 2).

4.2 LP formulation

Given the above problem inputs, a schedule is described as $S = \{S_{j,k,x}^i \mid \forall i, x \in \mathrm{DC}, t_k^j \in T\}$, where the value of $S_{j,k,x}^i$ is set to 1 if task t_k^j is scheduled to data center i, and its data is accessed from data center x; otherwise, $S_{j,k,x}^i$ is set to 0. In other words, for each task $t_k^j \in T$, a schedule must decide where to execute the task (i.e. DC_i), and where to access the data required by the task (i.e. DC_x). For the ease of discussion, in the rest of this paper we call the first decision as the *task placement decision*, and the second decision as the *data access decision*.

Our objective is to find the optimal schedule that achieves the minimum makespan. In this work, we define the makespan of a schedule as the latest job completion time of all the data centers resulting from the scheduling decision. Therefore, if E_i denoted the total execution time on data center i resulting from a schedule S, the makespan of S

is denoted as T_S, and $T_S = \max_{\forall i \in \mathrm{DC}} |E_i|$. Therefore, our problem can be formulated into a linear programming form as below:

$$\min T_S$$

$$\text{s.t} \quad E_i = \sum_{i,x \in \mathrm{DC}, t_k^j \in T} S_{j,k,x}^i \times \left[\frac{w_k^j}{C_i} + \frac{s_k^j}{B_{i,x}} \right] \leq T_S, \forall i \in \mathrm{DC} \tag{1}$$

$$\sum_{i,x \in \mathrm{DC}} S_{j,k,x}^i = 1, \forall t_k^j \in T \tag{2}$$

$$\sum_{i,x \in \mathrm{DC}, t_k^j \in T} \left(S_{j,k,x}^i \times \left(1 - L_{j,k}^x \right) \right) = 0 \tag{3}$$

$$S_{j,k,x}^i \in \{0, 1\} \tag{4}$$

In our problem setting, the total execution time of a data center is the aggregated execution time of all the tasks scheduled on the data center. The execution time of a task can be further divided into the task computation time and data transfer time (i.e., if data is accessed from a remote data center). Therefore, in the first equation, $\frac{w_k^j}{C_i}$ represents the computation time of executing task t_k^j on data center i, and $\frac{s_k^j}{B_{i,x}}$ represents the data transfer time from the data location x to task execution location i. The execution time of a task is multiplied by $S_{j,k,x}^i$, so that the value remains if and only if the scheduling assignment does occur (i.e., $S_{j,k,x}^i = 1$). The inequality constraint Eq. (1) ensure the completion time of any data center is bounded by the makespan, T_S.

Constraints in the second and third equations Eqs. (2) and (3) are added to further ensure the feasibility of our scheduling decision. Equation (2) ensures exactly one task placement decision and data access decision were made for each task t_k^j. In other words, a task cannot be placed in multiple data centers, and its required data cannot be access from multiple locations as well. Equation (3) ensures the data access decision must refer to a data center that actually has the required data by the task. For example, in $\sum(S_{j,k,x}^i \times (1 - L_{j,k}^x)) = 0$, let's assume a task t_k^j is schedule to data center i. If the required data d_k^j is located in data center x, then $Lx_{j,k} = 1$ and $(1 - L_{j,k}^x))$ will be 0. In contrast, if the data is not hosted by x, $(1 - L_{j,k}^x))$ will be 1; in such case we should force $S_{j,k,x}^i$ to be equal to 0. That is why the sum of the decision variable $S_{j,k,x}^i$ times $(1 - L_{j,k}^x)$ "*1 minus the value of the data access decision*" should be forced to be 0 for all data centers and all tasks in the system.

If we want to enforce data locality in our scheduling problem by avoiding any remote data access, we can simply add the following constraint into our formulation to restrict the scheduling decision.

$$S_{j,k,x}^i = 0, \quad \forall i \neq x \tag{5}$$

5 Data locality-aware scheduling

In this section, we derive the optimal solution for our scheduling problem on geo-distributed data centers and show the computational complexity of the optimal solution before we present a heuristic that guarantee near-optimal performance with lower complexity. In addition, we discussed the SWAG [23] algorithm which is used for comparison.

5.1 Schedule optimization with the LP solver

In this paper, it is assumed that the data are generated or collected and often replicated across data centers. The total execution time of a task depends on two decisions: the task placement and the data access.

As presented in Eqs. (1)–(5), we can formulate and solve the tasks placement and data access as a linear programming. The optimal solution got from the linear formulation is for a schedule instant (e.i: For a given set of jobs, the data location and the bandwidth between the data centers). This solution also implies that all data centers are initially available and no job is waiting in their respective local queues. In practice, the queues might not be initially empty. In addition, we target a system which admits users jobs submitted regularly at a certain rates. Therefore, we need an online scheduling mechanism. At arrival, jobs are first placed in a waiting queue. Then when a data center in a system is in idle state, it requests jobs from the arriving job queue. This is a trigger for the global scheduler to solve the problem Eqs. (1)–(4)/(5). All the jobs in the waiting queue are then given as input to the LP solver. We use the GLPK solver [39]. Then, the local queues q_i of each data center is updated with the new makespan resulting form the current schedule instant. Note that the constrains Eqs. (1)–(4) are used when we allow migration while Eqs. (1)–(5) are used when we want to enforce data locality.

Suppose that we have already scheduled jobs from a fist schedule instant and the global queue of data centers contains $Q = \{q_1, q_2, \ldots, q_N\}$. In the second schedule instant, we can reformulate the constrain Eq. (1) of the linear programming as follow:

$$E_i = \sum_{i,x \in \mathrm{DC}, t_k^j \in T} S_{j,k,x}^i \times \left[\frac{w_k^j}{C_i} + \frac{s_k^j}{B_{i,x}} \right] \leq (\mathbf{T_S} - \mathbf{q_i}) \tag{6}$$

Here, $(\mathbf{T_S} - \mathbf{q_i})$ is the schedule length on the data center i. If we minimize T_S under constraint Eq. (6), then we can substitute T_S by $T_T = \max(T_S + q_i)$ from the first schedule instant. Furthermore, the initial constrain can be written $T_S = \max(T_S + 0)$ if we assume all queues q_i are initially empty.

Although, there is an optimal solution, it is impractical to use an LP solver in the online scheduler due to the high computation time required to solve the problem. Even with a single machine, the makespan minimization problem of a splitting job is proven to be strongly-NP-hard [42]. The motivation behind this work is to introduce a low-complexity algorithm which can schedule the jobs on geo-distributed data centers such that the total schedule length is minimized.

5.2 GeoDis scheduler

As explained earlier, the problem is more complex on geo-distributed system and requires efficient tasks placement to benefit from the replica and to avoid unnecessary data transfer among data centers. Although significant improvement can be observed from existing algorithms including locality aware schedulers, there is still a room for improvement when we include the possibility to migrate the data as we presented in the example section. 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 that data locality reduces the data transfer time. Therefore, data locality should be favored whenever it is possible. Should data still be transferred to gain better performance, we start by the smallest chunks to avoid migrating a large portion of data. Finally, whenever a data is requested from another data center, we select the data from the data center with the fastest network bandwidth.

Rationale 2 The tasks from a job needs to be balanced among data centers. This is because the makespan of a job is bounded by the completion time of the latest tasks on any data center. Therefore, balancing the tasks from a job can shorten its maximum schedule length if the tasks distribution is skewed because of the initial data location. Also, balancing the load of a job can help us balance the load between data center more easily for our third rationale.

Rationale 3 The total load among data centers needs to be balanced as well. This is because in the worst case the total makespan of a workload is bounded by the maximum load among data centers, and the highly loaded data center can easily become the bottleneck for minimizing the makespan. 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.

Rationale 1 is addressed by sorting the tasks by their required data size in the descending order such as larger data get schedules first. *Rationale* 2 suggests to balance the load. At this point, there is a conflict with the Rationale 1. If we minimize the transfer time by only placing the tasks on the data location, due to the skewness of the data, the makespan will increase considerably. According to the *Rationale* 3 the workload should be balanced among data centers. Next, we rely on these rationales to design our algorithm shown in Algorithm 1. It consists of two steps explained as follows.

The first step in our approach is to avoid migrating large amount of data. Therefore, data locality will be favored when the required data to be processed reaches a certain threshold. In addition, when a file must be transferred, it must be done through the faster bandwidth between a data center hosting a copy and the placement location to minimize the transfer time. The result of the first step is to decide which job to schedule first. The intuition is that tasks from the same job can run together without

affecting the quality of the schedule. We get as a result of the first step an ordered list based on the shortest job first policy. In the second step, the tasks are effectively placed. The distinction between the two phases lies in the optimization mechanism to reduce the complexity. Further more, to know which job can be balanced among the data centers, in step 1 *line7–line17*, we first initialize e_j to zero. Then for all the jobs in the queue, we iteratively compute the resulting execution time: $[\frac{w_k^j}{C_d} + \frac{s_k^j}{B_{d,x}}]$ for all $d, x \in S$. We then choose as targeted data center ($target$) to place the task, the data center which result in the minimum makespan (*line13*). Ultimately, the makespan of the current job is measured by e_j will be q_{max} (*line15*). Following, we sort the jobs by their makespan in ascending order. In the second step, we rely on the shortest job first and perform the balance in the task placement similarly to the previous step.

Algorithm 1 GEODIS

1: **procedure** GEODIS(DC, B, J, T, $\mathcal{L}$, s, Q)
2: **Input:**
3: DC: data centers;B: Network bandwidth; J: jobs; T: tasks; L: data placements; s: data size ; $Q = \{q_1, \cdots, q_N\}$: schedule queue of data centers
4: **Output:**
5: $Q = \{q_1, \cdots, q_N\}$: updated schedule queue of data centers
6: //Step1: Balance the load and sort jobs by their minimum execution time
7: $e_j \leftarrow 0, \forall J_j \in J$
8: **for** $J_j \in J$ **do**
9: $curr\_q_d \leftarrow q_d, \forall q_d \in Q$
10: **for** $t_k^j \in J_j$ **do**
11: **for** $d \in DC$ **do**
12: **for** $x \in DC$ **do**
13: $target \leftarrow min_d|curr\_q_d + [\frac{w_k^j}{C_d} + \frac{s_k^j}{B_{d,x}}]|, \forall DC_d, DC_x \in S$ and $L_{j,k}^x = 1$
14: $curr\_q_{target} = curr\_q_{target} \cup t_k^j$
15: $curr\_q_{max} \leftarrow max|curr\_q_d|)$
16: $e_j \leftarrow |curr\_q_{max}|$
17: Sort $J_j \in J$ by e_j in ascending order
18: //Step2: Place tasks on the data center with minimum load
19: **for** $J_j \in J$ **do**
20: **for** $t_k^j \in J_j$ **do**
21: **for** $d \in DC$ **do**
22: **for** $x \in S$ **do**
23: $target \leftarrow min_d|q_d + [\frac{w_k^j}{C_d} + \frac{s_k^j}{B_{d,x}}]|, \forall DC_d, DC_x \in S$ and $L_{j,k}^x = 1$
24: $q_{target} = q_{target} \cup t_k^j$
25: **return** Q

5.3 Workload-aware greedy scheduling (SWAG)

SWAG [23] is a workload-aware greedy scheduling technique that prioritizes jobs execution among data centers. SWAG and GeoDis use the same job and task model. SWAG schedules jobs iteratively and keeps track of the queue length in each data

center. Initially, the queues in each data center is set to zero. Iteratively, SWAG picks a job and base on the data location, place the tasks such as the length of the queues are minimized, then recodes the highest queue length. The jobs with the minimum highest queue length is then scheduled and the queue lengths are updated in each data center respectively to the completion time of the latest task. Clearly, in each iteration, SWAG picks the job that will result in the lowest maximum queue length among all data centers. However, SWAG is not aware of data replication and schedule job based on the initial data location. 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 is more likely to be skewed and caused unbalanced workload and longer job makespan. In addition, SWAG does not propose any mechanism to transfer data and the processing task to a remote data center. In contrast, we intend in GeoDis to take advantage of the replicas and task migration.

6 Performance evaluation and analysis

We evaluated and compared the following approaches: Using the GLPK solver [39] and allowing migration as specified in the problem setting. The second approach also uses the LP solver with extra condition to enforce the locality. They are labeled respectively as **LP_Migration** and **LP_Locality**. **GeoDis** is our solution which also allows data transfer from remote data center. The fourth algorithm is **SWAG** [23] which uses the initial data location as tasks placement polity.

The goal of these evaluations is to show the potential of locality and data migration for a scheduler in geo-distributed data centers. Specifically, in Fig. 3 we show the overall evaluation. Following, we discuss the impact of the system utilization in Fig. 4 and brought up some analysis on the data distribution in Figs. 5 and 6. Finally, we reported the time to solution analysis of the algorithms in Fig. 7. In the following, we present the setup and the analysis of the experimentation conducted on simulations with realistic settings.

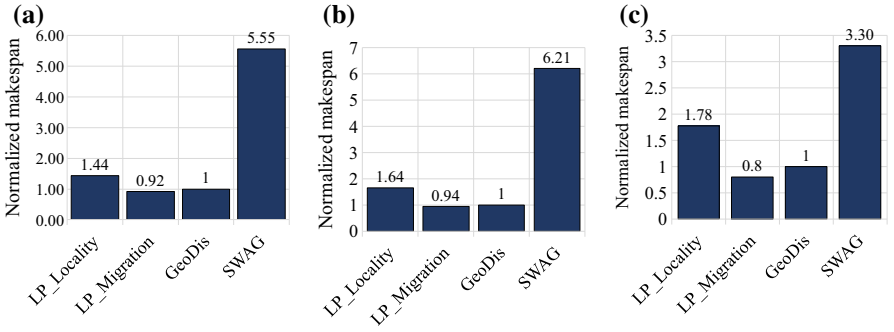

Fig. 3 Performances on makespan minimization with different workloads normalized to GeoDis. **a** Performance on Facebook 2009 trace. **b** Performance on Facebook 2010 trace. **c** Performance on exponential synthetic trace

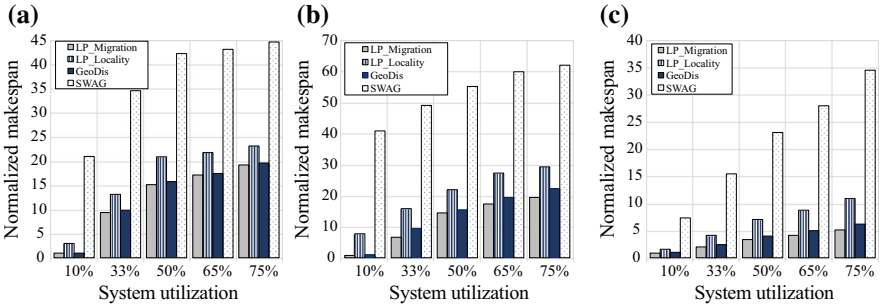

Fig. 4 Performances on makespan minimization at different system utilization normalized to GeoDis. **a** Performance on Facebook 2009 trace. **b** Performance on Facebook 2010 trace. **c** Performance on exponential synthetic trace

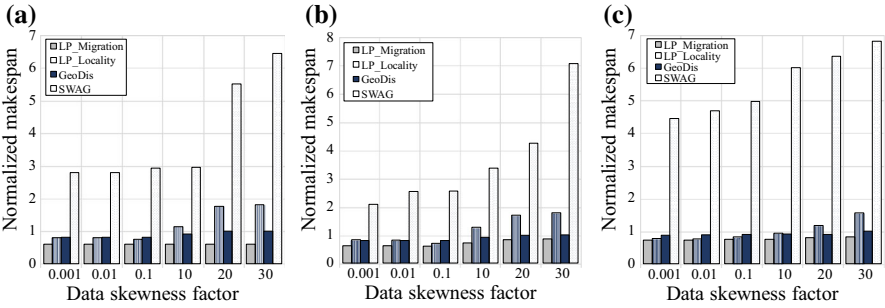

Fig. 5 Performances on makespan minimization with different initial data skewness normalized to GeoDis. **a** Performance on Facebook 2009 trace. **b** Performance on Facebook 2010 trace. **c** Performance on exponential synthetic trace

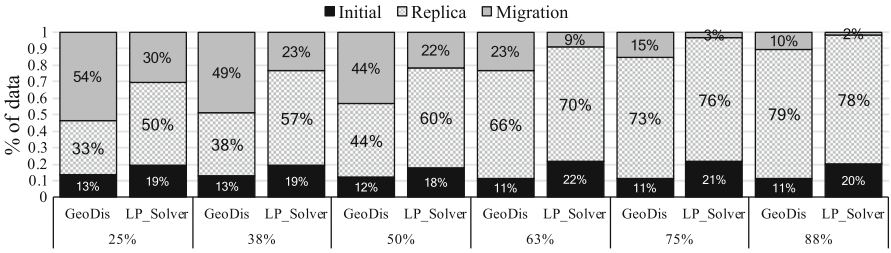

Fig. 6 Impact of the data replication

6.1 Setup

6.1.1 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 3. The capacity of a data center is equal to the $\#nodes \times \#cores\_per\_node$. Unless specified, the classical 3-replica

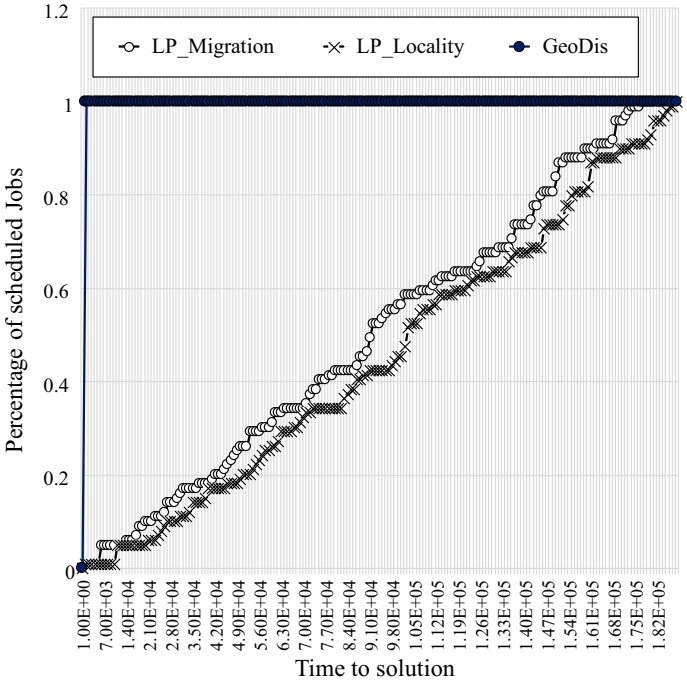

Fig. 7 Time complexity analysis: comparison of time to solution

Table 3 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

policy is adopted by default and replicated data are randomly placed in the data centers. The bandwidth among data centers are based on Amazon EC2 small instance using S3 as storage in each data center. The bandwidth values in Table 4 are the Min between the network among the source data center from which data is migrated and destination data center; and the storage i/o bandwidth at the destination data center [2]. A similar setting has been used by previous work [22].

Table 4 Bandwidth among data centers (Mbps)

	DC1	DC2	DC3	DC4	DC5	DC6	DC7	DC8
DC1	–	931	376	822	99	677	389	935
DC2	931	–	97	672	381	82	408	93
DC3	376	97	–	628	95	136	946	175
DC4	822	672	628	–	945	52	77	50
DC5	99	381	95	945	–	822	685	535
DC6	677	82	136	52	822	–	69	639
DC7	389	408	946	77	685	69	–	243
DC8	935	93	175	50	535	639	243	–

Table 5 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

6.1.2 Workloads traces

We used two traces from Facebook workloads [4,5]: 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 traces. The arrival rates are respectively 3.54 and 3.4 s for `FB-2009` and `FB-2010`. The summary can be found in the Table 5. For example, jobs with less than 150 tasks are considered small, 151 to 600 tasks are considered medium and any job having more than 601 tasks is considered a large job. Since each task processes a portion of data, the size of a job often measured by the number of tasks. Moreover, the model followed here is extended from that presented by Hung et al. [23]. Along with these traces, we randomly generate traces from a random generator to evaluate the performance under various tests settings. The synthetic workload generator randomly generates traces based on the data model described by Chen et al. [12]. By default, we randomly generated 1000 jobs following a uniform distribution. The initial data location follows 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. In the extreme case, data are skewed in one data center when α is equal to 30. Note that using Zipf-like distributions in data analytic applications simulation provides a wide range of data skewness to assess the scheduling performance. This scenario has been evaluated for various Web cashing, Data Grid and geo-distributed data analytics related publications [7,8,23,27].

6.2 Performance evaluation and analysis

6.2.1 The overall performance analysis

This section compares the performance of the GeoDis with that of SWAG and the LP solver. For this purpose, we consider the three traces described above with the default settings. The values in Fig. 3 are normalized to GeoDis. In the `FB-2009` trace which is dominated by small jobs—87.30% of them have fewer tasks, GeoDis reduced the makespan by $(5.54 - 1)/5.54 = 82\%$ as compared to SWAG. For the performance on `FB-2010` trace presented in Fig. 3b, GeoDis outperforms SWAG by $(6.21-1)/6.21 = 84\%$ on the makespan. We explain this by the fact that the tasks placement policy used in SAWG is based on the initial data location. As a consequence, when the data are initially skewed on the same fewer data centers, the resulting makespan is high. Enforcing the data locality with LP_Locality reduces the makespan by 74% in comparison to SWAG. Furthermore, the optimization decision based on the data transfer made by LP_Migration and GeoDis adds even more substantial contribution to the makespan minimization. Specifically, as shown in Fig. 3b, the makespan of GeoDis still remains within only $10 \sim 14\%$ higher than the result by LP_Migration, with an observation that small tasks favors the LP solver more than the larger tasks. Enforcing data locality in the placement decision has not been beneficial to LP_Locality either. This observation is consistent throughout Fig. 3a–c. For example, in the results shown in Fig. 3c, GeoDis outperforms LP_Locality by 44%. LP_Migration on the other hand, can take advantage on the idle data centers to schedule tasks and transfer their required data as far as it minimizes the overall makespan. We can draw two conclusions in this overall performance analysis: First, enforcing locality or relying on the initial data location has a significant impact on the makespan. Secondly, when data are skewed in fewer data centers, there is a huge potential to rely on the data transfer to minimize the overall makespan. As reported in this analysis, GeoDis can achieve as close as 86–91% of the solution by the LP solver.

6.2.2 System tilization

In this section we want to measure the performance of the algorithms with regard to the system utilization. The system utilization is measured by the ratio of the workload divided by the system capacity C_i. The jobs inter-arrival rate determines how many jobs are queued between each schedule instant. In fact a larger rate means less jobs will arrive to the global scheduler while a small rate (e.g every second) will result in a larger number of jobs arriving for the same schedule instant. The default setting assumes a fixed arrival rate in the system. In reality, jobs arrive at different rate in the course of the day. One way to investigate the robustness of our scheduling algorithm is to assume that arrival time follows an exponential distribution, leading to various levels of system utilization. Following this, we increase the job inter-arrival rate such that at any given schedule instant, we can reach a certain system utilization. We measured and reported the performance of the algorithms on Fig. 4. The results show that as the system utilization increases, the makespan increase for both algorithms. However, for the different traces, the variance between the makespan produced by GeoDis remains

small. In Fig. 4a and b, we present the log values of the makespan in order to show how close are the results. Again it shows how close is GeoDis as compared to the LP_Migration. Unlike the other algorithms, SWAG results in a longer makespan. A higher system utilization results in a larger number of jobs per schedule instant and therefore requires tasks to be placed carefully. As we report in Fig. 4c, the resulting makespan of the LP_Migration and that of GeoDis remains within 5% of the makespan while SWAG performs up 89% longer makespan over GeoDis.

These results are in line with those we analyzed in the overall analysis. Moreover, unlike the LP_Locality and SWAG, the makespan of both the LP_Migration and GeoDis algorithm do not exhibit a larger variation as the system utilization increase. For example, in Fig. 4c, increasing the system utilization by 15% (50% utilization to 65% system utilization), it only occurs on average in 2 and 6% makespan increase for LP_Migration and GeoDis respectively. This is a testimony from which we can conclude that the two algorithms still perform very well in different system utilization level.

6.2.3 Data characteristic analysis

In Fig. 5 we report the makespan obtained for each data characteristics. We show the impact of the initial data location within the geo-distributed data centers on the makespan produced by the algorithms. Data are generally produced or collected by different techniques with different objective functions. Therefore, it is not always possible to accurately predict or set the initial data location in a simulation. To capture various initial data locations and to assess the performance of the algorithms under different initial conditions, this section evaluates the initial data characteristics. It encompasses different initial data location and the replicas as well. To this end, we used different data skewness and present the results in Fig. 5. The skewness factor $\alpha = [\frac{1}{1000}, 0.1]$ results in a uniform distribution, while any value higher than 1 results in a skewed distribution (at $\alpha = 30$, almost all the data are initially stored in one data center). The first observation is that both of LP solutions and GeoDis perform closely at $\alpha = [\frac{1}{1000}, 0.1]$. Recall that the core technique of GeoDis is to balance the workload per job level and at the overall data centers level. When data are uniformly distributed, both advantages are considerably reduced as shown with the `FB-2009` trace in Fig. 5a and the `FB-2009` trace in Fig. 5b. The difference in the makespan produced by the algorithms is even smaller for small tasks (e.g the case of Fig. 5a). Furthermore, on the other hand, when the data are severely skewed (at $\alpha = 30$), more transfers can be initiated to minimize the makespan. That is the reason why LP_Migration has the least makespan increase throughout the different skewness factors. It is also essential to note the transitions between the LP_Locality and GeoDis over the two categories: GeoDis is within a ratio of $2 \sim 9\%$ of the makespan produced by LP_Locality for uniform distributed data. However, for skewed data, the former outperform the results of the LP_Locality by $25 \sim 82\%$ as shown in Fig. 5a. The results show that our algorithm has a best performance over the LP_Locality. The reason is that as data are more and more skewed, there is an opportunity to balance the processing tasks through data migration; at the same time the locality of tasks placement is reduced to fewer number of data centers resulting in imbalance load among data centers. Since

the makespan is governed by the latest task to complete in any data center, the over loaded data center is bottleneck in this particular situation.

In Fig. 6 we present the data access results. The goal is to evaluate the impact of the replication factor over the data access patterns. Here, a replication factor of 50% means each data has a replica in half of the data centers that compose the system. In this experiment we only consider the LP_Migration and GeoDis. The other algorithms do not consider migration and therefore are not interesting for this analysis. Here we want to report the percentage of data processed by the tasks either the initial data or fist copy, the replica or data which is migrated then processed in different data center. A general observation for both algorithms reads like this: as the number of replicas increases, more tasks are scheduled based on locality. The reason for this observation is that as more data are replicated, more tasks can be placed on the replicas reducing the total execution time. In addition, the percentage of initial data processed by the tasks remain consistent between 11–22%. Migration and locality combined make 80% of the data process. When we break locality and migration apart, at a lower replication factor, GeoDis migrates more data than it relies on locality; 54% of data migrated over 33% using locality. For 50–63%, both algorithms rely more on locality with more than 70% of the data process for the LP solver and 66% for GeoDis. In the extreme, although unrealistic, when we duplicate data over 88% of the data centers, very few data transfers are needed to minimize the makespan. This results is in conformity with our assumptions. Data is only migrated when workload schedule can benefit the from it to minimize the overall makespan.

6.2.4 Evaluation of the running time

We evaluate the running time of the algorithms in this experiment. We removed SWAG from this experiments. We set the arrival rate to 0.1 ms to ensure that there is no idle data center during the course of the scheduling process. In total we scheduled 1300 jobs and record the time taken to compute the task placement and the data access. Figure 7 shows the time to solution between GeoDis and the LP solver. In this figure, our algorithm considerably outperforms the LP solver. As compared to GeoDis, it takes on average 3 additional minutes to run the LP solver for a workload consisting of 5 jobs. We further observe that there is a slight difference between the LP_Migration and LP_Locality. Intuitively, more constrains to satisfy make the placement decision harder to compute. Therefore, when we force data locality with an additional constrain, we increase the size of the solution matrix and therefore it will require more time to compute the solution. This validates the efficiency of GeoDis and our goal of generating the schedule within a reasonable time.

6.2.5 Discussion

In this study, we evaluated two main optimization conditions on geo-distributed systems. One considering data migration and the second one forcing locality. Our model and simulations lead to the following insights:

– The approach which allows migration in the optimization process is a subset of the one that forces data locality. We came to this conclusion because we observed that

when the network connection is very low (e.g higher transfer time) both approaches perform equally.
– From the previous observation, it worths using an approach which allows data migration in case we have no knowledge of the workload size or the initial data location. The reality is that, forcing locality requires more time to solution while in any case an optimal solution forcing locality can perform any better than an optimal solution with migration.

7 Conclusion

Geo-distributed data centers have increasingly enabled the deployment of large-scale data-intensive applications. The advent of big data, have also changed the way we collect, reproduce and store data. In addition, to guarantee service availability institutions often replicate the data in different locations. Data locality aware scheduling have been efficiently used in cluster systems to avoid the growing network bandwidth requirement. However, scheduling the data processing jobs for the data hosted in different datacenters suffer sometimes of imbalanced load since data are expected to be skewed. As a result, data may still be transferred over the network to other data centers. To efficiently schedule jobs composed of multi-tasks in such environment such as to combine the benefit of data locality and avoid unnecessary data movement, we firstly demonstrate that the task placement and the data access can be formulated as a linear programing problem. Then, we proposed a low complexity algorithm for data processing jobs which are split and placed on different data centers. Through simulations with realistic traces we have demonstrated that the makespan our solution remains within 91% close to the optimal solution by the LP solver and can outperform the locality aware schedule by 44%, when initial data are skewed.

References

1. Abad CL, Lu Y, Campbell RH (2011) Dare: adaptive data replication for efficient cluster scheduling. In: 2011 IEEE international conference on cluster computing, pp 159–168. doi:10.1109/CLUSTER.2011.26
2. Abawajy JH, Deris MM (2014) Data replication approach with consistency guarantee for data grid. IEEE Trans Comput 63(12):2975–2987. doi:10.1109/TC.2013.183
3. AWS: Amazon Web Service (2006). http://aws.amazon.com
4. Ananthanarayanan G, Ghodsi A, Shenker S, Stoica I (2013) Effective straggler mitigation: attack of the clones. In: Presented as part of the 10th USENIX symposium on networked systems design and implementation (NSDI 13). USENIX, Lombard, IL, pp 185–198. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/ananthanarayanan
5. Ananthanarayanan G, Kandula S, Greenberg A, Stoica I, Lu Y, Saha B, Harris E (2010) Reining in the outliers in map-reduce clusters using mantri. In: Proceedings of the 9th USENIX conference on operating systems design and implementation, OSDI'10. USENIX Association, Berkeley, CA, USA, pp 265–278. http://dl.acm.org/citation.cfm?id=1924943.1924962
6. Anikode LR, Tang B (2011) Integrating scheduling and replication in data grids with performance guarantee. In: Global telecommunications conference (GLOBECOM 2011), 2011 IEEE, pp 1–6. doi:10.1109/GLOCOM.2011.6134492
7. Breslau L, Cao P, Fan L, Phillips G, Shenker S (1999) Web caching and Zipf-like distributions: evidence and implications. In: INFOCOM '99. Eighteenth annual joint conference of the IEEE computer and communications societies. Proceedings. IEEE, vol 1, pp 126–134. doi:10.1109/INFCOM.1999.749260

8. Cameron DG, Carvajal-Schiaffino R, Millar AP, Nicholson C, Stockinger K, Zini F (2003) Evaluating scheduling and replica optimisation strategies in optorsim. In: Proceedings. First Latin American Web Congress, pp 52–59 (2003). doi:10.1109/GRID.2003.1261698
9. Cardosa M, Wang C, Nangia A, Chandra A, Weissman J (2011) Exploring MapReduce efficiency with highly-distributed data. In: Proceedings of the second international workshop on MapReduce and its applications, MapReduce '11, ACM, New York, NY, USA, pp 27–34. doi:10.1145/1996092.1996100
10. Cavallo M, Modica GD, Polito C, Tomarchio O (2016) Application profiling in hierarchical Hadoop for geo-distributed computing environments. In: 2016 IEEE symposium on computers and communication (ISCC), pp 555–560. doi:10.1109/ISCC.2016.7543796
11. Chen W, Paik I, Li Z (2016) Cost-aware streaming workflow allocation on geo-distributed data centers. IEEE Trans Comput 66(2):256–271. doi:10.1109/TC.2016.2595579
12. Chen Y, Ganapathi A, Griffith R, Katz R (2011) 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, pp 390–399. doi:10.1109/MASCOTS.2011.12
13. Cheng D, Rao J, Guo Y, Jiang C, Zhou X (2017) Improving performance of heterogeneous mapreduce clusters with adaptive task tuning. IEEE Trans Parallel Distrib Syst 28(3):774–786. doi:10.1109/TPDS.2016.2594765
14. Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1):107–113. doi:10.1145/1327452.1327492
15. Elghirani A, Subrata R, Zomaya AY (2007) Intelligent scheduling and replication in datagrids: a synergistic approach. In: Seventh IEEE international symposium on cluster computing and the grid (CCGrid '07), pp 179–182. doi:10.1109/CCGRID.2007.65
16. Garg N, Kumar A, Pandit V (2007) Order scheduling models: hardness and algorithms. In: Proceedings of the 27th international conference on foundations of software technology and theoretical computer science, FSTTCS'07, Springer, Berlin, pp 96–107
17. Google Compute Engine (2011). https://cloud.google.com/compute/
18. Greenberg A, Hamilton J, Maltz DA, Patel P (2008) The cost of a cloud: research problems in data center networks. SIGCOMM Comput Commun Rev 39(1):68–73. doi:10.1145/1496091.1496103
19. Apache Hadoop Project (2013). http://hadoop.apache.org
20. Heintz B, Chandra A, Sitaraman RK, Weissman J (2016) End-to-end optimization for geo-distributed mapreduce. IEEE Trans Cloud Comput 4(3):293–306. doi:10.1109/TCC.2014.2355225
21. Herodotou H, Dong F, Babu S (2011) No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In: Proceedings of the 2nd ACM symposium on cloud computing, SOCC '11, ACM, New York, NY, USA, pp 18:1–18:14. doi:10.1145/2038916.2038934
22. Hu Z, Li B, Luo J (2016) Flutter: scheduling tasks closer to data across geo-distributed datacenters. In: IEEE INFOCOM 2016 - the 35th annual IEEE international conference on computer communications, pp 1–9. doi:10.1109/INFOCOM.2016.7524469
23. Hung CC, Golubchik L, Yu M (2015) Scheduling jobs across geo-distributed datacenters. In: Proceedings of the sixth acm symposium on cloud computing, SoCC '15, ACM, New York, NY, USA , pp 111–124. doi:10.1145/2806777.2806780
24. Isard M, Budiu M, Yu Y, Birrell A, Fetterly D (2007) Dryad: distributed data-parallel programs from sequential building blocks. In: Proceedings of the 2007 Eurosys conference. Association for Computing Machinery, Inc., Lisbon, Portugal. http://research.microsoft.com/apps/pubs/default.aspx?id=63785
25. Jalaparti V, Ballani H, Costa P, Karagiannis T, Rowstron A (2012) Bridging the tenant-provider gap in cloud services. In: Proceedings of the third ACM symposium on cloud computing, SoCC '12, ACM, New York, NY, USA, pp 10:1–10:14. doi:10.1145/2391229.2391239
26. Jalaparti V, Bodik P, Menache I, Rao S, Makarychev K, Caesar M (2015) Network-aware scheduling for data-parallel jobs: plan when you can. SIGCOMM Comput Commun Rev 45(4):407–420. doi:10.1145/2829988.2787488
27. Jin Y, Gao Y, Qian Z, Zhai M, Peng H, Lu S (2016) Workload-aware scheduling across geo-distributed data centers. In: 2016 IEEE Trustcom/BigDataSE/ISPA, pp 1455–1462. doi:10.1109/TrustCom.2016.0228
28. Jolfaei F, Haghighat AT (2012) 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, pp 283–288

29. Kloudas K, Mamede M, Preguiça N, Rodrigues R (2015) Pixida: optimizing data parallel jobs in wide-area data analytics. Proc VLDB Endow 9(2):72–83. doi:10.14778/2850578.2850582
30. Koshiba Y, Chen W, Yamada Y, Tanaka T, Paik I (2015) Investigation of network traffic in geo-distributed data centers. In: 2015 IEEE 7th international conference on awareness science and technology (iCAST), pp 174–179 (2015). doi:10.1109/ICAwST.2015.7314042
31. Kwok YK, Ahmad I (1999) Fastest: a practical low-complexity algorithm for compile-time assignment of parallel programs to multiprocessors. IEEE Trans Parallel Distrib Syst 10(2):147–159. doi:10.1109/71.752781
32. Lee YC, Zomaya AY (2007) Practical scheduling of bag-of-tasks applications on grids with dynamic resilience. IEEE Trans Comput 56(6):815–825. doi:10.1109/TC.2007.1042
33. Li P, Guo S, Miyazaki T, Liao X, Jin H, Zomaya A, Wang K (2016) Traffic-aware geo-distributed big data analytics with predictable job completion time. IEEE Trans Parallel Distrib Syst 28(6):1785–1796. doi:10.1109/TPDS.2016.2626285
34. Li P, Guo S, Yu S, Zhuang W (2015) Cross-cloud mapreduce for big data. IEEE Trans Cloud Comput 26(3):1–14. doi:10.1109/TCC.2015.2474385
35. Li S, Lu Q, Zhang W, Zhu L (2015) A mapreduce cluster deployment optimization framework with geo-distributed data. In: 2015 IEEE 12th Intl Conf on ubiquitous intelligence and computing and 2015 IEEE 12th intl conf on autonomic and trusted computing and 2015 IEEE 15th intl conf on scalable computing and communications and its associated workshops (UIC-ATC-ScalCom), pp 943–949. doi:10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2015.179
36. Li, W., Yang, Y., Yuan, D.: 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, pp 496–502. doi:10.1109/DASC.2011.95
37. Liao X, Gao Z, Ji W, Wang Y (2015) An enforcement of real time scheduling in spark streaming. In: Green computing conference and sustainable computing conference (IGSC), 2015 sixth international, pp 1–6. doi:10.1109/IGCC.2015.7393730
38. Lin W, Qian Z, Xu J, Yang S, Zhou J, Zhou L (2016) Streamscope: continuous reliable distributed processing of big data streams. In: 13th USENIX symposium on networked systems design and implementation (NSDI 16), USENIX Association, Santa Clara, CA, pp 439–453. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/lin
39. Makhorin A (2012) Gnu linear programming kit, version 4.52. http://www.gnu.org/software/glpk/glpk.html
40. Mandal A, Xin Y, Baldine I, Ruth P, Heerman C, Chase J, Orlikowski V, Yumerefendi A (2011) Provisioning and evaluating multi-domain networked clouds for Hadoop-based applications. In: 2011 IEEE third international conference on cloud computing technology and science, pp 690–697. doi:10.1109/CloudCom.2011.107
41. Microsoft Azure (2010). https://azure.microsoft.com/
42. Nguyen VH, Tuong NH, Tran VH, Thoai N (2013) An MILP-based makespan minimization model for single-machine scheduling problem with splitable jobs and availability constraints. In: Computing, management and telecommunications (ComManTel), 2013 international conference on, pp 397–400. doi:10.1109/ComManTel.2013.6482427
43. Pu Q, Ananthanarayanan G, Bodik P, Kandula S, Akella A, Bahl P, Stoica I (2015) Low latency geo-distributed data analytics. SIGCOMM Comput Commun Rev 45(4):421–434. doi:10.1145/2829988.2787505
44. Pu Q, Ananthanarayanan G, Bodik P, Kandula S, Akella A, Bahl P, Stoica I (2015) Low latency geo-distributed data analytics. In: Proceedings of the 2015 ACM conference on special interest group on data communication, SIGCOMM '15, ACM, New York, NY, USA, pp 421–434. doi:10.1145/2785956.2787505
45. Rackspace (1998). https://www.rackspace.com/
46. Schrage L (1968) A proof of the optimality of the shortest remaining processing time discipline. Oper Res 16(3):687–690. https://www.rackspace.com/
47. Sih GC, Lee EA (1993) A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Trans Parallel Distrib Syst 4(2):175–187. doi:10.1109/71.207593
48. Sooezi N, Abrishami S, Lotfian M (2015) Scheduling data-driven workflows in multi-cloud environment. In: 2015 IEEE 7th international conference on cloud computing technology and science (CloudCom), pp 163–167. doi:10.1109/CloudCom.2015.95

49. Apache Spark? (2013). http://spark.apache.org/
50. Toosi AN, Buyya R (2015) 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), pp 186–194. doi:10.1109/UCC.2015.35
51. Tripathi R, Vignesh S, Tamarapalli V, Medhi D (2017) Cost efficient design of fault tolerant geo-distributed data centers. IEEE Trans Network Service Manag 14(2):289–301. doi:10.1109/TNSM.2017.2691007
52. Tudoran R, Costan A, Antoniu G (2016) Overflow: multi-site aware big data management for scientific workflows on clouds. IEEE Trans Cloud Comput 4(1):76–89. doi:10.1109/TCC.2015.2440254
53. Venugopal S, Buyya R (2008) An SCP-based heuristic approach for scheduling distributed data-intensive applications on global grids. J Parallel Distrib Comput 68(4):471–487. doi:10.1016/j.jpdc.2007.07.004
54. Vulimiri A, Curino C, Godfrey PB, Jungblut T, Karanasos K, Padhye J, Varghese G (2015) Wanalytics: geo-distributed analytics for a data intensive world. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data, SIGMOD '15, ACM, New York, NY, USA, pp 1087–1092. doi:10.1145/2723372.2735365
55. Vulimiri A, Curino C, Godfrey PB, Jungblut T, Padhye J, Varghese G (2015) Global analytics in the face of bandwidth and regulatory constraints. In: 12th usenix symposium on networked systems design and implementation (NSDI 15), USENIX Association, Oakland, CA, pp 323–336. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/vulimiri
56. Wang L, Tao J, Ranjan R, Marten H, Streit A, Chen J, Chen D (2013) G-Hadoop: mapreduce across distributed data centers for data-intensive computing. Future Gener Comput Syst 29(3):739–750. doi:10.1016/j.future.2012.09.001. Special section: recent developments in high performance computing and security
57. Zarina M, Ahmad F, bin Mohd Rose AN, Nordin M, Deris MM (2013) Job scheduling for dynamic data replication strategy in heterogeneous federation data grid systems. In: Informatics and applications (ICIA), 2013 second international conference on, pp 203–206. doi:10.1109/ICoIA.2013.6650256