J Grid Computing (2010) 8:133–155
DOI 10.1007/s10723-009-9129-0

Algorithms for Divisible Load Scheduling of Data-intensive Applications

Chen Yu · Dan C. Marinescu

Received: 13 January 2009 / Accepted: 12 August 2009 / Published online: 2 September 2009

Abstract In this paper we introduce the *Divisible Load Scheduling (DLS)* family of algorithms for data-intensive applications. The polynomial time algorithms partition the input data and generate optimal mappings to collection of autonomous and heterogeneous computational systems. We prove the optimality of the solution and report a simulation study of the algorithms.

Keywords Divisible Load Scheduling · Divisible Load Theory (DLT) · Load balancing · Grid computing

1 Introduction and Related Work

In this paper we address the problem of scheduling independent process groups of a data-intensive application in a heterogenous computing environment consisting of a large number of autonomous parallel systems such as clusters, or supercomputers. Our model for *Divisible Load Scheduling (DLS)* is based on several assumptions: (i) we assume a divisible load, one that can be arbitrarily partitioned; (ii) a Coordinator supervises the data-intensive computation carried out by a large number of target systems equipped with Execution Engines; (iii) the process groups running on the target systems process different blocks of data; they are logically equivalent, but their implementations may differ as they are optimized for a particular target system architecture and configuration; (iv) the process groups do not communicate with one another; (v) the performance metric we wish to optimize is the *makespan*, the time from the schedule is generated and the completion time of the last process group; (vi) the target systems are heterogenous and autonomous systems.

Minimizing the makespan of an application with multiple independent tasks on heterogenous environments is frequently discussed in the literature [3, 11, 31, 36]. For example, in [11], the authors compare eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems while in this paper we discuss a single data-intensive computation. In our case data partitioning, mapping process groups to heterogeneous systems, and scheduling are intimately correlated. The scheduling algorithm for the DLS problem not only generates the optimal schedule, but also calculates the data assigned to each process group. We do not have a priori knowledge on the *expected execution time* (computation and communication cost) of

C. Yu (✉) · D. C. Marinescu
School of Electrical Engineering & Computer Science,
University of Central Florida,
Orlando, FL 32816, USA
e-mail: yuchen@cs.ucf.edu

D. C. Marinescu
e-mail: dcm@cs.ucf.edu

each process group; the execution time is determined by the workload allocated to each process group by the scheduling algorithm. The number of process groups can either be static (e.g., required by application) or dynamic, produced by the DLS algorithm.

The scheduling policies for heterogeneous and autonomous systems [17] reported in this paper consider the time when each potential target system is available, the computational power and the workload of the individual system, as well as the network characteristics. Target systems are autonomous and equip with the local schedulers to make local scheduling decisions. The startup time for the given computation on each system is not under the control of a centralized authority. Space sharing of a single cluster ensures that all nodes available start processing at the same time but this is no longer true in the large-scale distributed environment. Thus, neither the traditional data partitioning and scheduling algorithms for the SPMD model [16, 30, 39] in a dedicated parallel environment nor the gang scheduling algorithms [2, 19, 22] for the heterogeneous computing platform can be effective in this case. Moreover, the input data should be made available to an execution site, before the computation could start, a process called *data staging*. The network bandwidth and the communication latency cannot be ignored [26]. Scheduling algorithms that only take into account the computation without regard for communication cost are unlikely to be efficient. Some works such as [3, 28] involves the data transfer cost into the scheduling process. However, the common assumption on these works that the *expected communication costs* are known or can be estimated beforehand does not apply in our case as the amount of data to be transferred is only known as a result of the algorithm.

In 1996, the Divisible Load Theory (DLT) [6] was formalized and a comprehensive summary of the research prior to 2003 can be found in [7]. DLT is a linear mathematical model and provides a practical framework for the mapping on the independent process groups onto heterogeneous computing platforms. Applications of image processing [24], biological computing [21, 25], multimedia applications [1], or database record searching [8] can be approached as divisible load. *One-round* divisible load algorithms allocate exactly one block of data to each system were investigated for several network topologies: linear [13], star [5], multi-level tree [14], bus [4], hypercube [9], and mesh [10]. *Multi-round* algorithms [44] are based on a pipelined execution by overlapping the computation and data staging.

Several papers discuss the application of the DLT to Grid computing [23, 29, 37, 38, 42, 48, 49]. The DLT strategies with one-port communication for the star-shaped and tree-shaped topologies are studied in [5]. In the one-port communication model the Coordinator cannot distribute the workload to target systems in parallel and can only communicate with a single target system at a time. Both the one-round and multi-round algorithms are discussed with the analysis of optimality of the solution in [5]. Despite the similarity of the platform and application model, there are a number of important differences between previously reported work e.g., [5], and the work reported in this paper. First, we assume a multi-port communication model which enables the Coordinator to distribute the load to target systems in parallel. For reasons of performance and data security, data-intensive applications store the data at "data centers" with massive communication bandwidth and low communication latencies. In addition, new network technologies such as BitTorrent [15] substantially improve the performance of parallel data transfer and we expect that multi-port communication will become increasingly more popular. Second, we introduce a new data staging strategy different from one-round and multi-round strategies; we assumes the data staging can begin after the schedule is generated and terminate before the scheduled computation startup time on a target system. It has been proven that all target systems should finish computing at the same time to achieve the optimal solution for one-round and multi-round strategies [5, 43] but we will show that this is not always true for the data staging strategy mentioned above. Last but not least, the DLS algorithms select a subset of resources from the resource pool for parallel execution, in contrast to the linear case where all available target systems participate.

In a recent paper [47], we present a DLS algorithm for the pipelined data staging strategy. In

this paper, we extend our discussion and cover all data staging strategies. The contributions of this paper are: (i) introduce the DLS model with multi-port communication, discuss the parameters, called *performance evaluators*, to characterize the heterogeneity of the Grid resources and illustrate three different data staging strategies; (ii) develop a set of optimal divisible load scheduling algorithms with the polynomial running time and show the optimality of the solution; (iii) evaluate the proposed DLS algorithms by simulation and discuss the general selection criterion based on the feature of the application and the data staging strategies; (iv) present the design and implementation of a DLS meta-scheduler.

2 The System Model

We consider a data-intensive computation $\mathcal{C}$ that consists of m independent duplicates, called m process groups, all of which execute logically equivalent programs on target systems equipped with Execution Engines but each on a different input data segment: $\mathcal{C} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_m\}$. We assume that we have $n \geq m$ heterogeneous target systems that form the *target systems set*: $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2 \ldots, \mathcal{S}_n\}$. These target systems can be owned by different person, organizations or institutions and thus are in different administrative domains but we assume all of them can communicate with the Coordinator.

A subset of target systems, forming the *restricted target set* $\mathcal{Q}^{(\pi)}$ such that $|\mathcal{Q}^{(\pi)}| = m$, will be selected by scheduling algorithm from $\mathcal{S}$ to run the task. The Coordinator along with the m restricted target systems form the single level tree (star) topology as shown in Fig. 1. There is a communication link from the Coordinator (the root of the tree) to each of the restricted target systems (the leafs of the tree), and the Coordinator is able to dispatch the data in data inventory to the restricted target systems in parallel, resulting in a multi-port communication model. Without loss of generality we assume that the Coordinator has no data processing capability thus all workload will be processed by the m process groups mapping one to one onto the m restricted target systems.

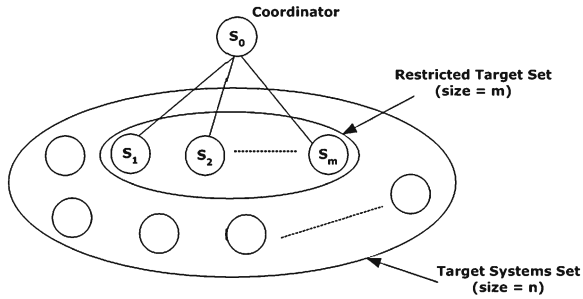

Fig. 1 Single level tree connection between the Coordinator and m restricted target systems. The m restricted target system used for parallel execution are selected from the target systems set

We name this scheduling model as *Divisible Load Scheduling (DLS)*.

Given the restricted target set $\mathcal{Q}^{(\pi)}$ with $|\mathcal{Q}^{(\pi)}| = m$, an *allocation* ν of process groups $\mathcal{G}_i \in \mathcal{C}$ to systems $\mathcal{S}_j \in \mathcal{Q}^{(\pi)}$ is an one-to-one mapping $\nu : \mathcal{G}_i \mapsto \mathcal{S}_j$. Given an allocation ν, the *data partitioning* for ν is to compute a decomposition δ of the input data set with size ω into segments of size $\omega_1, \omega_2, \ldots, \omega_m$ such that $\omega = \sum_{i=1}^{m} \omega_i$ based on the objective function in such a way that process group $\mathcal{G}_i$ works on data segment ω_i. The pair $\pi = (\nu, \delta)$ describing both the allocation of process groups to target systems and the data partitioning is called a *mapping* of $\mathcal{C}$ to $\mathcal{Q}^{(\pi)}$.

For simplicity we consider that the Coordinator broadcasts a task execution request at time $t = 0$; we also ignore the time to compute the schedules, as it is considerably smaller than the task execution time.

The *makespan*, denoted it as $\tau^{(\pi)}$, is the time elapsed from the moment when the schedule is generated by the Coordinator until the last process group finishes. Given a mapping π, the individual *process group completion time* on the restricted target system $\mathcal{S}_i$ in $\mathcal{Q}^{(\pi)}$ is denoted as $T_i^{(\pi)}$. The objective function is defined as the makespan of $\mathcal{C}$ under mapping π, denoted as:

$$\tau^{(\pi)} = \max\left(T_1^{(\pi)}, T_2^{(\pi)}, \ldots, T_m^{(\pi)}\right). \tag{1}$$

The DLS problem is to determine the optimal mapping such that the makespan $\tau^{(\pi)}$ of $\mathcal{C}$ is minimized:

$$\min_{\pi}\left[\tau^{(\pi)}\right]. \tag{2}$$

If there are no restrictions regarding the maximum number of process groups, we can utilize as many target system as we can in order to achieve the optimal makespan. The number of the process groups becomes part of the output of the scheduling algorithm in this case. We call this scenario as *FlexMap*. In practice, we expect to be limited by factors such as the cost to access a resource, or some characteristics of the application that force us to use a definite number of target systems. The *FixMap* scenario addresses this problem and the given size of process groups becomes the input of the scheduling algorithm instead.

Mapping and scheduling decisions are based upon metrics describing the availability of the computation resource and the rate of resource consumption by an application. We use four parameters called *performance evaluators* to characterize the heterogeneity of the target systems. The execution rate, the duty cycle, and the available time characterize the computational capability and the status of a target system and the data transfer rate measures the network sustained stable bandwidth and thus will be used to estimate the data transfer cost.

Execution Rate We assume that the input data set consists of ω *data units (dtu)s*. A *dtu* reflects the logical organization of the data. For example, for the virus origin and orientation refinement application [21], a *dtu* consists of a number of virus projections extracted from a micrograph. We define the *execution rate* μ_i of a task on target system $\mathcal{S}_i$ as the number of *dtu*s processed per unit of time assuming that the data is available locally and the target system $\mathcal{S}_i$ is fully loaded with this task. The execution rate can be estimated based upon actual measurements through *test process* (running code with sample input on target systems), using the past history, or by comparison with similar tasks or with the execution rate on similar systems.

Duty Cycle A parallel system can be time-shared or space-shared. Time-sharing allows multiple tasks to execute "concurrently" on the nodes of a parallel system while space-sharing means to allocate a subset of nodes to one task until completion. We define the *duty cycle* η_i as the fraction of the computational power available for a given task on system $\mathcal{S}_i$. When the execution rate and the duty cycle for a given task are respectively μ_i and η_i, the *actual execution rate* of the task on system $\mathcal{S}_i$ is: $\mu_i\eta_i$. Although the accurate measurement is complicated, a decent approximation to the duty cycle is the *CPU availability*. In time-shared system, CPU availability quantifies the fraction of CPU cycles available to a given task [12, 32]. A model to predict the CPU availability on time-shared Unix systems is discussed in [40]. In the case of space-sharing system, the CPU availability is the fraction of CPUs that will be allocated to the given task.

Available Time The local scheduler on each target system maintains the job queues and sometimes provides the estimated time when each job will be scheduled for execution. For example, each of the nine TeraGrid [18] supercomputer centers runs a service to furnish job wait time estimate. The *Network Weather Service (NWS)* [41] also provides its Batch Queue Prediction Service (QBETS). The estimation is more accurate when the local scheduler has access to the jobs description files, the resource allocation strategy, local scheduling strategy and historical task execution statistics. The estimated startup time on target system $\mathcal{S}_i$ for a given task is called the *available time* of system $\mathcal{S}_i$ for this task, denoted as σ_i. The capability to estimate the available time allows us to reserve the resource predictably and achieve the improved makespan, since the resource with longer available time can be superior to the resource with immediate availability if the former shows much better computational capability.

Data Transfer Rate A data-intensive task may rely on a single data inventory or multiple sites where data is collected. Data staging, the process of transferring data from the single or the multiple inventories to the site when the data is processed consumes a fraction of the networks bandwidth between these sites. The data staging costs are a function of bandwidth, latencies, packet loss, jitter, anomalies of network links and the amount of data being transferred and affect the overall execution of the data-intensive task [26]. We define the *Data Transfer Rate* R_i to describe the sustained data transfer rate between the system

$\mathcal{S}_i$ targeted for execution and the data inventory. There are several theoretical models to predict the bandwidth of a sustained TCP connection. For example, a formula to derive the maximum TCP throughput subjected to light to moderate packet losses for TCP Congestion Avoidance algorithm [20, 34] is introduced in [27]. The end-to-end TCP/IP performance (bandwidth and latency) can also be estimated using tools such as *Network Weather Service (NWS)* [41] and *iperf* (http://dast.nlanr.net/projects/Iperf/).

3 Data Staging Strategies

Data staging can only be initiated after the algorithms described in this paper have determined the size of the data segment allocated to each process group running on a target system. Several data staging strategies are possible:

- *Data Staging Before Available Time (DSBAT)* The data staging begins as soon as the schedule/mapping is available and terminates on or before the time a system is available; the computation starts at the scheduled available time.
- *Data Staging After Available Time (DSAAT)* The data staging could begin only after the system in the restricted target set is available; the computation start as soon as data staging is finished. This strategy corresponds to the one-round model in [5].
- *Pipelined Data Staging (PDS)* This model is an improved version of DSAAT; data staging overlaps with computing. The data staging is carried out by a p stage pipeline such that most of the data staging cost are hidden. This strategy corresponds to the multi-round model in [5].

Figure 2 shows the behavior of three data staging strategies. Intuitively DSBAT strategy can achieve the best makespan as it hides the data staging cost from makespan completely. However, this is not always the case and both DSAAT and PDS strategies have the opportunity to outperform the DSBAT strategy when the input data set size becomes large and the number of process groups is limited to a small number. This is due

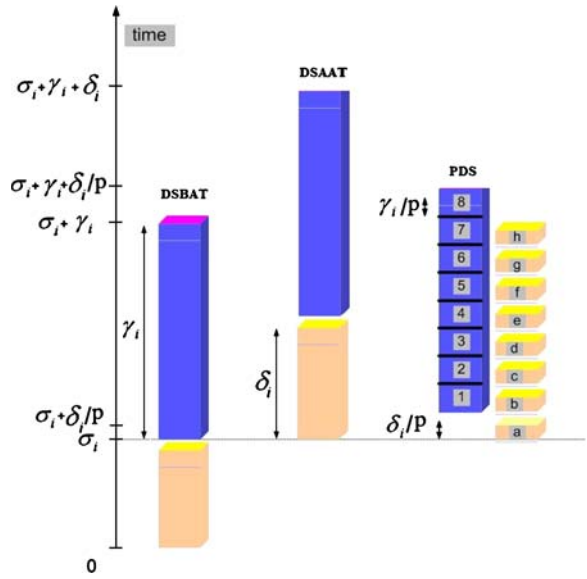

Fig. 2 Compute the individual process group completion time based on different data staging strategies on target system $\mathcal{S}_i$ with the available time σ_i: (*Left*) *DSBAT* Strategy—the data staging happens from the time when mapping is generated to the available time of the target system. The process group completion time is $\sigma_i + \gamma_i$ with γ_i the data processing time. (*Middle*) *DSAAT* Strategy—The data staging starts at time σ_i and lasts δ_i seconds. The process group completion time is: $\sigma_i + \delta_i + \gamma_i$. (*Right*) *PDS* Strategy—The data staging occurs over p pipelined stages. The process group completion time is $\sigma_i + \delta_i/p + \gamma_i$. We let $p = 8$. Computation stages are labeled 1–8 and the corresponding data staging are *a–h*

to the internal limitations on DSBAT strategy: (i) each restricted target system has the fixed data staging period which may limit the data a system can process even if it comes with the superior computational capability; (ii) the selection of the restricted target systems is also influenced by the factor that all the input data must be dispatched to them completely before their available times, otherwise, there is no solution for the DSBAT strategy. DSAAT strategy is the easiest scenario for implementation as it does not require the specific features of the local scheduler on the target system. The PDS strategy overlaps data transfer and data processing subject to the condition that data transfer rate R_i is larger or equal to actual execution rate $\mu_i\eta_i$ for every restricted target system $\mathcal{S}_i$: $R_i \geq \mu_i\eta_i$, such that the computation starts at each phase consecutively without waiting for the data. DSAAT strategy indeed is the special case

of PDS strategy with the number of pipeline stage equal to 1 but without the condition on the data transfer rate and actual execution rate. We will further compare three data staging strategies in detail in Section 5.

Given the mapping π for a data-intensive task. Suppose system $\mathcal{S}_i$ is selected as one restricted target system with the available time σ_i, the execution rate μ_i, the duty cycle η_i and the data transfer rate R_i. The data segment size assigned to $\mathcal{S}_i$ is denoted by $\omega_i^{(\pi)}$. Let δ_i and γ_i denote the data staging cost and data processing cost respectively.

The process group completion time $T_i^{(\pi)}$ for the DSBAT strategy is:

$$T_i^{(\pi)} = \sigma_i + \gamma_i = \sigma_i + \frac{\omega_i^{(\pi)}}{\mu_i \eta_i} \tag{3}$$

with the assumption that $\frac{\omega_i^{(\pi)}}{R_i} \leq \sigma_i$. This assumption guarantees $\mathcal{S}_i$ finishes the data staging on or before its available time σ_i. Similarly, the process group completion time for DSAAT strategy is:

$$T_i^{(\pi)} = \sigma_i + \delta_i + \gamma_i = \sigma_i + \frac{\omega_i^{(\pi)}}{R_i} + \frac{\omega_i^{(\pi)}}{\mu_i \eta_i}. \tag{4}$$

The process group completion time for a pipelined strategy with p stages is:

$$T_i^{(\pi)} = \sigma_i + \delta_i + \gamma_i = \sigma_i + \frac{\omega_i^{(\pi)}}{p R_i} + \frac{\omega_i^{(\pi)}}{\mu_i \eta_i}. \tag{5}$$

4 Optimal Divisible Load Scheduling (DLS) Algorithms

The limitation on the number of process groups or the target systems the task can utilize combined with different data staging strategies lead to multiple families of algorithms, Table 1.

The inputs for the DLS algorithms are: (i) the target systems set $\mathcal{S}$ with performance evaluators μ_i, η_i, σ_i, R_i for each target system $\mathcal{S}_i$; (ii) the size of input data set ω; (iii) the size of restricted target set (*FixMap* scenario only) m; and (iv) the number of pipeline stages (PDS model only) p. The outputs of the DLS algorithms are: (i) the makespan $\tau^{(\pi)}$; (ii) the restricted target system set $\mathcal{Q}^{(\pi)}$; and (iii) the data segment size for each restricted target system $\omega_i^{(\pi)}$.

We wish to minimize the makespan $\tau^{(\pi)}$ of the data intensive task. The algorithms return the optimal schedule in polynomial time if the schedule exists.

4.1 Data Staging Before Available Time (DSBAT) Strategy

Although the data staging happens after the schedule is generated by the Coordinator and before the available time on each target system thus the data staging cost is hidden from the makespan completely, the DSBAT strategy has the substantial difference with the case without data staging process for the following limitations: (i) the workload allocated to each restricted target system must be no more than the amount of input data it can receive during its data staging period; the maximal amount of data the restricted target system $\mathcal{S}_i$ can receive is calculated as $R_i\sigma_i$; (ii) the sum of the workload received by all restricted target systems needs to be equal to the input data set size, otherwise, there is no solution for the DSBAT strategy.

4.1.1 The FLX-DSBAT and FLX-NODS Algorithms

In this case there is not limit on the number of the process groups and the number of target systems we can harness for task parallel execution. Before we introduce the *FLX-DSBAT* algorithm, we first present the necessary and sufficient conditions for the optimal solution.

Table 1 Divisible load scheduling (DLS) algorithms

	No DS	DS prior σ_i	DS after σ_i	Pipelined DS
FlexMap	*FLX-NODS*	*FLX-DSBAT*	*FLX-DSAAT*	*FLX-PDS*
FixMap	*FIX-NODS*	*FIX-DSBAT*	*FIX-DSAAT*	*FIX-PDS*

σ_i is the available time of the restricted target system. DS denotes data staging

Proposition 1 *Let $\mathcal{Q}^{(\pi)}$ be the restricted target set and $\tau^{(\pi)}$ be the makespan under mapping π. The mapping $\pi : \mathcal{C} \mapsto \mathcal{Q}^{(\pi)}$ is the optimal solution of FLX-DSBAT scenario if and only if three conditions are satisfied:*

1. *For each process group on the restricted target system in $\mathcal{Q}^{(\pi)}$, it finishes data staging on or before its available time:*

$$\sigma_i \geq \frac{\omega_i^{(\pi)}}{R_i}, \quad \forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}. \tag{6}$$

2. *The process group completion time on each restricted target system in $\mathcal{Q}^{(\pi)}$ whose data staging process can finish earlier than its available time is equal to $\tau^{(\pi)}$:*

$$T_i^{(\pi)} = \tau^{(\pi)}, \quad \forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)} \ \textit{where} \ \sigma_i > \frac{\omega_i^{(\pi)}}{R_i}. \tag{7}$$

3. *Any system $\mathcal{S}_q$ not in $\mathcal{Q}^{(\pi)}$ only becomes available after or at the makespan under mapping π:*

$$\tau^{(\pi)} \leq \sigma_q, \quad \forall \mathcal{S}_q \notin \mathcal{Q}^{(\pi)}. \tag{8}$$

Proof We first show that the three conditions are necessary. Let $\pi = (\nu, \delta)$ be the optimal mapping for *FLX-DSBAT* scenario.

(i) Note that $\frac{\omega_i^{(\pi)}}{R_i}$ gives the time spent on the data staging. It is easy to see that Condition 1 holds based on the definition of *FLX-DSBAT* scenario.

(ii) Assume there exist two target systems $\mathcal{S}_x, \mathcal{S}_y \in \mathcal{Q}^{(\pi)}$ such that:

$$\sigma_x > \frac{\omega_x^{(\pi)}}{R_x}, \ \ \sigma_y > \frac{\omega_y^{(\pi)}}{R_y}, \textit{and} \ \ T_x^{(\pi)} < T_y^{(\pi)}.$$

Assume the process group completion time on target system $\mathcal{S}_z \in \mathcal{Q}^{(\pi)}$ is $\tau^{(\pi)}$. Note that $\mathcal{S}_z$ could be $\mathcal{S}_y$ or another target system, but in either case we have $T_x^{(\pi)} < T_z^{(\pi)} = \tau^{(\pi)}$. If we reassign a fraction of the workload from system $\mathcal{S}_z$ to $\mathcal{S}_x$, we end up achieving a new data partition scheme δ' corresponding to a new mapping $\pi' = (\nu, \delta')$ such that $T_x^{(\pi')} = T_z^{(\pi')} < \tau^{(\pi)}$. This procedure can be repeated and eventually leads to a new makespan $\tau^{(\pi')}$ with $\tau^{(\pi')} < \tau^{(\pi)}$ which contradicts the fact that π is an optimal mapping.

(iii) Assume there exists $\mathcal{S}_x \notin \mathcal{Q}^{(\pi)}$ such that $\sigma_x < \tau^{(\pi)}$. Now we can construct a new allocation ν' with $\mathcal{Q}^{(\pi')} = \mathcal{Q}^{(\pi)} \cup \{\mathcal{S}_x\}$. A new data partitioning δ' over $\mathcal{Q}^{(\pi')}$ will ensure a shorter makespan because some workload will be distributed to the new system. Thus the new mapping $\pi' = (\nu', \delta')$ leads to a shorter makespan $\tau^{(\pi')} < \tau^{(\pi)}$ which contradicts the fact that π is an optimal mapping.

Next, we show that the three conditions are sufficient. Consider mapping π and assume that we have $\tau^{(\pi)}$ and $\mathcal{Q}^{(\pi)}$.

(i) As the condition $\sigma_i \geq \frac{\omega_i^{(\pi)}}{R_i}, \ \forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}$ holds, all target systems in $\mathcal{Q}^{(\pi)}$ will finish data staging before or on their available times.

(ii) Assume all systems in $\mathcal{Q}^{(\pi)}$ satisfying the condition $\sigma_i > \frac{\omega_i^{(\pi)}}{R_i}$ form another target systems set $\mathcal{S}^{(t)}$ and finish computing at the same time, say $\tau^{(\pi)}$. Consider a new data partitioning δ' among systems in $\mathcal{S}^{(t)}$ and the corresponding mapping $\pi' = (\nu, \delta')$ with the same restricted target set. There will be at least one system, say $\mathcal{S}_x$ in $\mathcal{S}^{(t)}$, who holds more data than the amount originally assigned to it, such that $\tau^{(\pi)} < T_x^{(\pi')} \leq \tau^{(\pi')}$. This indicates that redistributing data among systems in $\mathcal{S}^{(t)}$ cannot lead to a shorter makespan. As the systems in $\mathcal{Q}^{(\pi)} - \mathcal{S}^{(t)}$ cannot accept more data, the data redistribution between systems in $\mathcal{S}^{(t)}$ and systems in $\mathcal{Q}^{(\pi)} - \mathcal{S}^{(t)}$ will also lead to a longer makespan.

(iii) Consider the condition $\tau^{(\pi)} \leq \sigma_i, \ \forall \mathcal{S}_i \notin \mathcal{Q}^{(\pi)}$. It implies no new target system can be added into the the restricted target set $\mathcal{Q}^{(\pi)}$ to improve the makespan $\tau^{(\pi)}$. We conclude that π must be the optimal solution and finish the proof. □

Algorithms 1 and 2 present the pseudo-code for the *FLX-NODS* and the *FLX-DSBAT* algorithms, respectively.

Algorithm 1 Pseudo-code for the *FLX-NODS* algorithm

Input: Target systems set $\mathcal{S}$, task input data set size ω
Output: makespan $\tau^{(\pi)}$, restricted target set $\mathcal{Q}^{(\pi)}$, data segment size for each restricted target system $\omega_i^{(\pi)}$

Sort target systems in $\mathcal{S}$ based on their available times, the earliest first;
$\mathcal{Q}^{(\pi)} \leftarrow \emptyset$;
for $i = 1$ *to SizeOf(*$\mathcal{S}$*)* **do**
  Add system $\mathcal{S}_i$ into $\mathcal{Q}^{(\pi)}$: $\mathcal{Q}^{(\pi)} \leftarrow \mathcal{Q}^{(\pi)} \cup \{\mathcal{S}_i\}$;
  Generate new makespan $\tau^{(i)}$;
  if i *!= SizeOf(*$\mathcal{S}$*) and Target system* $\mathcal{S}_{i+1}$*'s available time* $\geq \tau^{(i)}$ **then**
    Exit for loop;
  end
end
Set makespan: $\tau^{(\pi)} = \tau^{(i)}$;
foreach *Target system* $\mathcal{S}_i$ *in* $\mathcal{Q}^{(\pi)}$ **do**
  Compute data segment size: $\omega_i^{(\pi)} = \mu_i \eta_i (\tau^{(\pi)} - \sigma_i)$;
end

The necessary and sufficient conditions for the optimal solution of *FLX-NODS* scenario is similar to Proposition 1 except that there is no limitation on the size of the data segment each target system can get. The *FLX-NODS* algorithm iteratively adds new target system which is ordered by their available times into restricted target set and updates the makespan accordingly until no more target system can be involved into restricted target set such that the makespan is further improved. One key step of the *FLX-NODS* algorithm is to compute the makespan after adding a new target system into the restricted target set. As the sum of the size of data segments allocated to restricted target systems should be equal to the task input data set size, we have the relation to compute the makespan $\tau^{(j)}$ after the *jth* iteration of *FLX-NODS* algorithm:

$$\omega = \sum_{\forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}} \mu_i \eta_i (\tau^{(j)} - \sigma_i), \tag{9}$$

with the makespan $\tau^{(j)}$ the only unknown.

The *FLX-DSBAT* algorithm first verifies if a solution exists by accumulating the maximum data segment size that each target system in $\mathcal{S}$ can receive during its data staging process and comparing the result, expressed as $\sum_{\forall \mathcal{S}_i \in \mathcal{S}} R_i \sigma_i$, with the input data set size ω. When a solution exists the *FLX-DSBAT* algorithm will generate the temporary restricted target set $\mathcal{S}^{(p)}$ and the temporary makespan $\tau^{(p)}$ first by running the *FLX-NODS* algorithm without considering the data staging. Then the corresponding data segment size allocated to the restricted target system $\mathcal{S}_i$, which is expressed by $\mu_i \eta_i (\tau^{(p)} - \sigma_i)$, is compared with the maximum data segment size it can receive during its data staging period, which is computed as $R_i \sigma_i$. If the former is larger, a *data set reduction* operation occurs by reducing $\mathcal{S}_i$'s workload to $R_i \sigma_i$ and thus ensures the target system $\mathcal{S}_i$ can begin computing at its available time σ_i.

After reducing the data segment size assigned to system $\mathcal{S}_i$, the longer makespan is reached due to workload redistribution and the process group completion time on such system is shorter than the makespan. The restricted target systems experiencing the data set reduction are removed from the target systems set $\mathcal{S}$ but put into restricted target set $\mathcal{Q}^{(\pi)}$. The workload they bear are deducted from the input data set ω and the algorithm continues to the next iteration and generates the new temporary restricted target set and temporary makespan by *FLX-NODS* algorithm on the updated $\mathcal{S}$ and ω. The algorithm continues until no data set reduction operations happen on the target systems in the temporary target set $\mathcal{S}^{(p)}$ and by combining the $\mathcal{S}^{(p)}$ with current $\mathcal{Q}^{(\pi)}$ we achieve the final $\mathcal{Q}^{(\pi)}$. The last step of the algorithm is to compute the data segment size allocated to each restricted target system. The workload to each

Algorithm 2 Pseudo-code for the *FLX-DSBAT* algorithm

Input: Target systems set $\mathcal{S}$, task input data set size ω
Output: makespan $\tau^{(\pi)}$, restricted target set $\mathcal{Q}^{(\pi)}$, data segment size for each restricted target system $\omega_i^{(\pi)}$

if $\sum_{\forall \mathcal{S}_i \in \mathcal{S}} R_i\sigma_i \geq \omega$ **then**
  $\mathcal{Q}^{(\pi)} \leftarrow \emptyset$;
  while *1* **do**
    Generate the temporary restricted target set $\mathcal{S}^{(p)}$ and temporary makespan $\tau^{(p)}$ by running *FLX-NODS* algorithm on $\mathcal{S}$ and ω;
    Set flag = false;
    foreach *Target system* $\mathcal{S}_i$ *in temporary restricted target set* $\mathcal{S}^{(p)}$ **do**
      Calculate the data segment size allocated to $\mathcal{S}_i$ by *FLX-NODS* algorithm: $\omega_i{=}\mu_i\eta_i(\tau^{(p)} - \sigma_i)$;
      Calculate the maximum data segment size $\mathcal{S}_i$ can get before its available time: $d_i{=}R_i\sigma_i$;
      if $\omega_i > d_i$ **then**
        Remove $\mathcal{S}_i$ from target systems set: $\mathcal{S} \leftarrow \mathcal{S} - \{\mathcal{S}_i\}$;
        Put $\mathcal{S}_i$ into final restricted target set: $\mathcal{Q}^{(\pi)} \leftarrow \mathcal{Q}^{(\pi)} \cup \{\mathcal{S}_i\}$;
        Reduce input data set size: $\omega = \omega - d_i$;
        Set flag = true;
      end
    end
    if *flag == false* **then**
      Set final restricted target set: $\mathcal{Q}^{(\pi)} \leftarrow \mathcal{Q}^{(\pi)} \cup \mathcal{S}^{(p)}$;
      Set makespan: $\tau^{(\pi)} = \tau^{(p)}$;
      Exit while loop;
    end
  end
  foreach *target system* $\mathcal{S}_i$ *in* $\mathcal{Q}^{(\pi)}$ **do**
    Calculate data segment size allocated to $\mathcal{S}_i$:
    $\omega_i^{(\pi)} = \min\left(R_i\sigma_i,\ \mu_i\eta_i(\tau^{(\pi)} - \sigma_i)\right)$;
  end
else
  No solution;
end

restricted target system is computed by taking the smaller between the workload it can receive through data staging before its available time and the workload calculated by the simple derivation on the formula 3:

$$\omega_i^{(\pi)} = \min\left(R_i\sigma_i,\ \ \mu_i\eta_i(\tau^{(\pi)} - \sigma_i)\right). \tag{10}$$

4.1.2 The FIX-DSBAT and FIX-NODS Algorithms

The scheduling is more intricate when we limit the number of process groups or the size of restricted target set. Occasionally, the algorithm has to replace a system that is already included in the restricted target set by a better performing one to obtain a shorter makespan while keeping the size of restricted target set fixed. The available time of the target system becomes less important as in the *FlexMap* scenario since the system with longer available time can be superior to the one with much earlier availability if the former shows much better computational capability. To compensate for this restriction the algorithms for the *FixMap* scenario select the most "powerful" target systems, instead of using the systems with the earliest

availability. We now present the necessary and sufficient conditions for the optimal solution of the *FIX-DSBAT* scenario.

Proposition 2 *Let $\mathcal{Q}^{(\pi)}$ be the restricted target set with fixed size m and $\tau^{(\pi)}$ be the makespan under mapping π. The mapping $\pi : \mathcal{C} \mapsto \mathcal{Q}^{(\pi)}$ is the optimal solution of FIX-DSBAT scenario if and only if three conditions are satisfied:*

1. *For each process group on the restricted target system in $\mathcal{Q}^{(\pi)}$, it finishes data staging on or before its available time:*

$$\sigma_i \geq \frac{\omega_i^{(\pi)}}{R_i}, \quad \forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}. \tag{11}$$

2. *The process group completion time on each restricted target system in $\mathcal{Q}^{(\pi)}$ whose data staging process can finish earlier than its available time is equal to $\tau^{(\pi)}$:*

$$T_i^{(\pi)} = \tau^{(\pi)}, \quad \forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)} \ \textit{where} \ \sigma_i > \frac{\omega_i^{(\pi)}}{R_i}. \tag{12}$$

3. *For each system $\mathcal{S}_q$ outside the restricted target set under mapping π, either it becomes available after or at the makespan $\tau^{(\pi)}$:*

$$\tau^{(\pi)} \leq \sigma_q \tag{13}$$

or, if $\tau^{(\pi)} > \sigma_q$, the following relation holds:

$$\min_{\forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}} \omega_i^{(\pi)} \geq \min\left((\tau^{(\pi)} - \sigma_q)\mu_q\eta_q, \ R_q\sigma_q\right). \tag{14}$$

Proof We first show that the three conditions are necessary. Let $\pi = (\nu, \delta)$ be an optimal mapping. Note that the proofs for necessity of Conditions 1 and 2 are exactly same as their counterparts in the proof of Proposition 1. We discuss only the proof for the third condition.

Assume there is one $\mathcal{S}_q \notin \mathcal{Q}^{(\pi)}$ such that:

$$\tau^{(\pi)} > \sigma_q \ \textit{and} \ \min_{\forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}} \omega_i^{(\pi)} < \min\left(\left(\tau^{(\pi)} - \sigma_q\right)\mu_q\eta_q, \ R_q\sigma_q\right).$$

The expression $(\tau^{(\pi)} - \sigma_q)\mu_q\eta_q$ gives the projected data segment size that can be processed by $\mathcal{S}_q$ in the time interval $[\sigma_q, \tau^{(\pi)}]$ and the expression $R_q\sigma_q$ expresses the maximum size of data that can be transferred to $\mathcal{S}_q$ before its available time. The smaller one of these two values represents the actual size of data segment that $\mathcal{S}_q$ can process in the time interval $[\sigma_q, \tau^{(\pi)}]$ if DSBAT strategy is considered.

Given the mapping π with the makespan $\tau^{(\pi)}$, assume that $\mathcal{S}_p \in \mathcal{Q}^{(\pi)}$ processes the least amount of data among systems in $\mathcal{Q}^{(\pi)}$. Thus, we have:

$$\omega_p^{(\pi)} = \min_{\forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}} \omega_i^{(\pi)} < \min\left(\left(\tau^{(\pi)} - \sigma_q\right)\mu_q\eta_q, \ R_q\sigma_q\right), \tag{15}$$

which implies that when we use a new mapping π', generated by replacing $\mathcal{S}_p$ by $\mathcal{S}_q$, to process the same amount of input data set (equal to ω), the new makespan $\tau^{(\pi')}$ will be less than $\tau^{(\pi)}$ by redistributing the input data among systems in $\mathcal{Q}^{(\pi')}$. This contradicts with the statement that $\tau^{(\pi)}$ is the optimal mapping.

Next, we show the sufficiency of these three conditions. Condition 1 indicates that all systems in the restricted target set $\mathcal{Q}^{(\pi)}$ are able to finish their data staging on time and Condition 2 tells us that redistributing data among systems in $\mathcal{Q}^{(\pi)}$ cannot lead to a shorter makespan (see the proof of Proposition 1). Consider a system not in $\mathcal{Q}^{(\pi)}$. According to Condition 3, either the system becomes available after or at $\tau^{(\pi)}$ which means that adding it to $\mathcal{Q}^{(\pi)}$ cannot improve the makespan, or it is available before $\tau^{(\pi)}$, but the amount of data it can process is less than the amount any system in $\mathcal{Q}^{(\pi)}$ can process; thus it makes no sense to swap it with any system already in $\mathcal{Q}^{(\pi)}$. We conclude that π must be an optimal mapping. We finishes the proof. □

The pseudo-code in Algorithm 3 shows the three steps of the *FIX-DSBAT* algorithm. The systems in $\mathcal{S}$ is first sorted based on the maximum amount of data they can receive during their data staging phases which is computed by expression $R_i\sigma_i$ for each system $\mathcal{S}_i$. The solution exists if the total amount of data that can be transferred during the data staging phase to the top m systems

Algorithm 3 Pseudo-code for the *FIX-DSBAT* algorithm

Input: Target systems set $\mathcal{S}$, task input data set size ω, the number of process groups m
Output: makespan $\tau^{(\pi)}$, restricted target set $\mathcal{Q}^{(\pi)}$, data segment size for each restricted target system $\omega_i^{(\pi)}$

```
Sort target systems in S based on the maximum data segment size each system
can receive during its data staging process, the largest first;
Q^(π) ← ∅;
if Σ_{i=1}^{m} R_i σ_i ≥ ω then
    Put the first m target systems in the ordered S into restricted target set
    Q^(π);
    Compute makespan τ^(π) by running FLX-DSBAT algorithm on Q^(π) and ω;
    while 1 do
        Generate candidates set S^(c);
        if S^(c) ≠ ∅ then
            Find system S_p in Q^(π) whose available time is before τ^(π) and
            processed the least workload among all systems in Q^(π);
            Find system S_q in S^(c) who will process the largest workload within
            time interval [σ_q, τ^(π)];
            if S_q will process more data than S_p then
                Remove S_p from Q^(π): Q^(π) ← Q^(π) − {S_p};
                Put S_q into Q^(π): Q^(π) ← Q^(π) ∪ {S_q};
                Generate new makespan τ^(π) by running FLX-DSBAT algorithm
                on the updated Q^(π) and original ω;
            else
                Exit while loop;
            end
        else
            Exit while loop;
        end
    end
    foreach Target system in restricted target set Q^(π) do
        if its available time is before makespan τ^(π) then
            Compute data segment size using formula 10;
        else
            Assign zero workload to it;
        end
    end
else
    No solution;
end
```

in the sorted set exceeds the total amount of input data ω. If the solution exists, the initial restricted target set $\mathcal{Q}^{(\pi)}$ is constructed by including the top m systems in the sorted set $\mathcal{S}$. Then the algorithm invokes the *FLX-DSBAT* algorithm on restricted target set $\mathcal{Q}^{(\pi)}$ and input data set size ω which will return the initial makespan $\tau^{(\pi)}$. With the initial task makespan, the *Candidates Set* $\mathcal{S}^{(c)}$ is generated by including such systems that they are not in $\mathcal{Q}^{(\pi)}$ but their available times are earlier than $\tau^{(\pi)}$. Such a target system in $\mathcal{S}^{(c)}$ is called a *candidate system* because it is possible to obtain a shorter makespan if we swap it with one system already in $\mathcal{Q}^{(\pi)}$. With the formula 10, we search the system among systems in $\mathcal{S}^{(c)}$ which will process the most workload if we put it into the restricted

target set based on current $\tau^{(\pi)}$ and the system in current $\mathcal{Q}^{(\pi)}$ with the least workload assigned and swap them if the former is larger. After swapping operation, the *FLX-DSBAT* algorithm is called to compute the new makespan using the updated $\mathcal{Q}^{(\pi)}$ and the original ω as inputs. With the new makespan, the candidates set is regenerated and the possibility of swapping operation is evaluated. The procedure repeats until (i) there is no possibility of swapping operation which indicates that the m most "powerful" systems have been selected already; or (ii) the candidates set becomes empty. The last step of the algorithm is to compute the workload for each restricted target system. The restricted target systems that becomes available on or after the makespan is identified and assigned zero workload while the others are computed by formula 10.

We note that when data staging cost is not considered, we have the *FIX-NODS* scenario. We can achieve the *FIX-NODS* algorithm by making several simple modifications on the *FIX-DSBAT* algorithm: (i) order the target systems based on their available times, the earliest first; (ii) there is no need to verify the existence of the solution as the solution will always be reached; (iii) both the projected workload for systems in candidates set and the actual workload for systems in restricted target set are computed as $\mu_i\eta_i(\tau^{(\pi)} - \sigma_i)$ instead of applying the formula 10.

4.2 Data Staging After Available Time (DSAAT) and Pipeline Data Staging (PDS) Strategies

As mentioned earlier, DSAAT strategy is a special case of PDS strategy. Therefore, in this section we focus on the algorithms for PDS strategy. We assume there are p pipeline stages and we will get the DSAAT strategy when $p = 1$.

4.2.1 The FLX-PDS Algorithm

We first present the sufficient and necessary conditions for the optimal solution of *FLX-PDS* scenario below.

Proposition 3 *Let $\mathcal{Q}^{(\pi)}$ be the restricted target set and $\tau^{(\pi)}$ be the makespan under mapping π. The mapping $\pi : \mathcal{C} \mapsto \mathcal{Q}^{(\pi)}$ is the optimal solution of FLX-PDS scenario if and only if two conditions are satisfied:*

1. *The process group completion time, $\forall \mathcal{G}_i \in \mathcal{C}$, under mapping π, is the same and equal to the makespan:*

$$T_1^{(\pi)} = T_2^{(\pi)} = ... = T_m^{(\pi)} = \tau^{(\pi)}. \tag{16}$$

2. *Any system $S_q \notin \mathcal{Q}^{(\pi)}$ only becomes available after or at the makespan $\tau^{(\pi)}$:*

$$\tau^{(\pi)} \leq \sigma_q, \quad \forall \mathcal{S}_q \notin \mathcal{Q}^{(\pi)}. \tag{17}$$

We omit the proof here as it is quite similar to the proof of Proposition 1.

Algorithm 4 shows the pseudo-code of *FLX-PDS* algorithm.

The *FLX-PDS* algorithm shares the same idea with the *FLX-NODS* and *FLX-DSBAT* algorithms. The target systems set is ordered by the target systems' available times. A new target system with the earliest available time is added into restricted target set iteratively. The makespan of the task is updated during each iteration until no improvement is achieved. To update the makespan, assume $\tau^{(i)}$ denotes the makespan after the *ith* iteration. For the first iteration, only one system, $\mathcal{S}_1$, is in $\mathcal{Q}^{(\pi)}$. The makespan $\tau^{(1)}$ is computed by formula 5 as

$$\tau^{(1)} = \sigma_1 + \delta_1 + \gamma_1 = \sigma_1 + \frac{\omega}{pR_1} + \frac{\omega}{\mu_1\eta_1}.$$

After putting $\mathcal{S}_2$ into $\mathcal{Q}^{(\pi)}$, the Coordinator needs to relocate some workload from system $\mathcal{S}_1$ to $\mathcal{S}_2$ and we denote the amount of workload reallocating to $\mathcal{S}_2$ from $\mathcal{S}_1$ as Δ_1. Note that the new makespan $\tau^{(2)}$ can be expressed by formula 5 as

$$\tau^{(2)} = \sigma_2 + \delta_2 + \gamma_2 = \sigma_2 + \frac{\Delta_1}{pR_2} + \frac{\Delta_1}{\mu_2\eta_2}.$$

Consider the improvement of the makespan, we have another way to express $\tau^{(2)}$ as

$$\tau^{(1)} - \tau^{(2)} = \frac{\Delta_1}{pR_1} + \frac{\Delta_1}{\mu_1\eta_1}.$$

Algorithm 4 Pseudo-code for the *FLX-PDS* algorithm

Input: Target systems set $\mathcal{S}$, input data set size ω, pipeline stage p
Output: makespan $\tau^{(\pi)}$, restricted target set $\mathcal{Q}^{(\pi)}$, data segment size for each restricted target system $\omega_i^{(\pi)}$

Sort target systems in $\mathcal{S}$ based on their available times, the earliest first;
$\mathcal{Q}^{(\pi)} \leftarrow \emptyset$;
for $i = 1$ *to SizeOf(*$\mathcal{S}$*)* **do**
  Add system $\mathcal{S}_i$ into $\mathcal{Q}^{(\pi)}$: $\mathcal{Q}^{(\pi)} \leftarrow \mathcal{Q}^{(\pi)} \cup \{\mathcal{S}_i\}$;
  Generate new makespan $\tau^{(i)}$;
  if i *!= SizeOf(*$\mathcal{S}$*) and Target system* $\mathcal{S}_{i+1}$*'s available time* $\geq \tau^{(i)}$ **then**
    Exit for loop;
  end
end
$\tau^{(\pi)} = \tau^{(i)}$;
foreach *Target system in* $\mathcal{Q}^{(\pi)}$ **do**
  Compute data segment size using the derivation on the formula 5:
  $\omega_i^{(\pi)} = \frac{\tau^{(\pi)} - \sigma_i}{\frac{1}{pR_i} + \frac{1}{\mu_i \eta_i}}$;
end

From these two different expressions of $\tau^{(2)}$, we derive the following formulas

$$\Delta_1 = \frac{\tau^{(1)} - \sigma_2}{\left(\frac{1}{pR_1} + \frac{1}{\mu_1\eta_1}\right) + \left(\frac{1}{pR_2} + \frac{1}{\mu_2\eta_2}\right)} \tag{18}$$

and

$$\tau^{(2)} = \tau^{(1)} - \left(\frac{\Delta_1}{pR_1} + \frac{\Delta_1}{\mu_1\eta_1}\right). \tag{19}$$

Similar to the derivation of $\tau^{(2)}$, assume we have j systems in the restricted target set, $\mathcal{S}_1, \mathcal{S}_2, ...\mathcal{S}_j$, where $j > 2$, and the makespan is $\tau^{(j)}$. To update the makespan after adding the system $\mathcal{S}_{j+1}$, we have the following observations:

(i) The process group completion time on each of the first j systems in the restricted target set is improved by the same amount:

$$\begin{aligned}
&\Delta_1\left(\frac{1}{pR_1} + \frac{1}{\mu_1\eta_1}\right) \\
&\quad = \Delta_2\left(\frac{1}{pR_2} + \frac{1}{\mu_2\eta_2}\right) \\
&\quad = ... \\
&\quad = \Delta_j\left(\frac{1}{pR_j} + \frac{1}{\mu_j\eta_j}\right).
\end{aligned} \tag{20}$$

(ii) The total workload reallocated from the first j systems in the restricted target set to the newly added system $\mathcal{S}_{j+1}$ is:

$$\omega_{j+1} = \sum_{i=1}^{j} \Delta_i. \tag{21}$$

Note that from formula 5 we have another way to express the ω_{j+1} as

$$\omega_{j+1} = \frac{\tau^{(j+1)} - \sigma_{j+1}}{\frac{1}{pR_{j+1}} + \frac{1}{\mu_{j+1}\eta_{j+1}}}. \tag{22}$$

(iii) The makespan is improved by the amount equal to the improvement of the process group completion time on each of the first j systems:

$$\tau^{(j)} - \tau^{(j+1)} = \Delta_i\left(\frac{1}{pR_i} + \frac{1}{\mu_i\eta_i}\right), \quad 1 \leq i \leq j. \tag{23}$$

The formulas 20, 21, 22, and 23 allow us to derive the expressions for Δ_1 and $\tau^{(j+1)}$ as:

$$\Delta_1 = \frac{\tau^{(j)} - \sigma_{j+1}}{\left(\frac{1}{pR_1} + \frac{1}{\mu_1\eta_1}\right) + \left(\frac{1}{pR_{j+1}} + \frac{1}{\mu_{j+1}\eta_{j+1}}\right)\left(1 + \sum_{i=2}^{j}\left(\frac{\frac{1}{pR_1}+\frac{1}{\mu_1\eta_1}}{\frac{1}{pR_i}+\frac{1}{\mu_i\eta_i}}\right)\right)} \tag{24}$$

and

$$\tau^{(j+1)} = \tau^{(j)} - \left(\frac{\Delta_1}{pR_1} + \frac{\Delta_1}{\mu_1\eta_1}\right). \tag{25}$$

At last, the size of the data segment allocated to the restricted target system $\mathcal{S}_i$ can be computed by the following formula derived from formula 5:

$$\omega_i^{(\pi)} = \frac{\tau^{(\pi)} - \sigma_i}{\frac{1}{pR_i} + \frac{1}{\mu_i\eta_i}}. \tag{26}$$

4.2.2 The FIX-PDS Algorithm

The *FIX-PDS* scenario constrains the number of process groups m, or in other words, the number of target systems that Coordinator can utilize for task parallel execution.

The necessary and sufficient conditions for the optimality of the solution from the *FIX-PDS* scenario are summarized by the following proposition.

Proposition 4 *Let $\mathcal{Q}^{(\pi)}$ be the restricted target set with fixed size m and $\tau^{(\pi)}$ be the makespan under mapping π. The mapping $\pi : \mathcal{C} \mapsto \mathcal{Q}^{(\pi)}$ is the optimal solution of FIX-PDS scenario if and only if two conditions are satisfied:*

1. *The process group completion time $T_i^{(\pi)}$, $\mathcal{G}_i \in \mathcal{C}$, under mapping π, is the same and equal to the makespan:*

$$T_1^{(\pi)} = T_2^{(\pi)} = ... = T_m^{(\pi)} = \tau^{(\pi)}. \tag{27}$$

2. *For each system outside the restricted target set under mapping π, $\forall \mathcal{S}_q \notin \mathcal{Q}^{(\pi)}$, either the system becomes available after or at the makespan $\tau^{(\pi)}$: $\tau^{(\pi)} \leq \sigma_q$, or, if $\tau^{(\pi)} > \sigma_q$, the following relation holds:*

$$\min_{\forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}} \left(\tau^{(\pi)} - \sigma_i\right) \frac{1}{\left(\frac{1}{pR_i} + \frac{1}{\mu_i\eta_i}\right)} \geq \left(\tau^{(\pi)} - \sigma_q\right) \frac{1}{\left(\frac{1}{pR_q} + \frac{1}{\mu_q\eta_q}\right)}. \tag{28}$$

Proof We first show that the two conditions are necessary. Let $\pi = (\nu, \delta)$ be the optimal mapping for the *FIX-PDS* scenario.

(i) Assume two process groups, $\mathcal{G}_x, \mathcal{G}_y \in \mathcal{C}$ exist such that under the allocation ν corresponding to the mapping π with makespan $\tau^{(\pi)}$, one process group, $\mathcal{G}_x$, is able to finish earlier than the other: $T_x^{(\pi)} < T_y^{(\pi)}$ and $T_y^{(\pi)} = \tau^{(\pi)}$. If we reassign a fraction of the workload from system $\mathcal{S}_y$ to $\mathcal{S}_x$, we may achieve the new data partition scheme δ' corresponding to a new mapping $\pi' = (\nu, \delta')$ such that: $T_x^{(\pi')} = T_y^{(\pi')} < \tau^{(\pi)}$. This process can be repeated until the completion time of each process group becomes the same and thus will lead to a new makespan: $\tau^{(\pi')} < \tau^{(\pi)}$, which contradicts with the assumption that $\tau^{(\pi)}$ is optimal.

(ii) Assume $\mathcal{S}_q \notin \mathcal{Q}^{(\pi)}$ and satisfies the relation:

$$\tau^{(\pi)} > \sigma_q \text{ and } \min_{\forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}} \left(\tau^{(\pi)} - \sigma_i\right) \frac{1}{\left(\frac{1}{pR_i} + \frac{1}{\mu_i\eta_i}\right)} < \left(\tau^{(\pi)} - \sigma_q\right) \frac{1}{\left(\frac{1}{pR_q} + \frac{1}{\mu_q\eta_q}\right)}.$$

The expression $(\tau^{(\pi)} - \sigma_q)\frac{1}{(\frac{1}{pR_q} + \frac{1}{\mu_q\eta_q})}$ gives workload that can be processed by $\mathcal{S}_q$ within

the time interval $[\sigma_q, \tau^{(\pi)}]$, based on the formula 26.

Given the mapping π with the makespan $\tau^{(\pi)}$. Assume that $\mathcal{S}_p \in \mathcal{Q}^{(\pi)}$ processes the lowest workload among systems in $\mathcal{Q}^{(\pi)}$. Thus, we have:

$$\begin{aligned}
&\left(\tau^{(\pi)} - \sigma_p\right) \frac{1}{\left(\frac{1}{pR_p} + \frac{1}{\mu_p \eta_p}\right)} \\
&= \min_{\forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}} \left(\tau^{(\pi)} - \sigma_i\right) \frac{1}{\left(\frac{1}{pR_i} + \frac{1}{\mu_i \eta_i}\right)} \\
&< \left(\tau^{(\pi)} - \sigma_q\right) \frac{1}{\left(\frac{1}{pR_q} + \frac{1}{\mu_q \eta_q}\right)}.
\end{aligned}$$

This implies that if we create a new mapping π', generated by replacing $\mathcal{S}_p$ by $\mathcal{S}_q$, for the same input workload ω, then $\tau^{(\pi')} < \tau^{(\pi)}$. This contradicts with the statement that $\tau^{(\pi)}$ is optimal.

Next we show that the two conditions are sufficient.

(i) Assume all systems in $\mathcal{Q}^{(\pi)}$ finish computation at the same time $\tau^{(\pi)}$. Consider a new data partitioning δ' and the corresponding mapping $\pi' = (\nu, \delta')$ with the same restricted target set. There will be at least one system, say $\mathcal{S}_x$, such that: $\tau^{(\pi)} < T_x^{(\pi')} \leq \tau^{(\pi')}$, which will lead to a longer makespan.

(ii) Consider a mapping π and a system $\mathcal{S}_q \notin \mathcal{Q}^{(\pi)}$ such that $\sigma_q \geq \tau^{(\pi)}$. Thus, it would not be helpful to add $\mathcal{S}_q$ to $\mathcal{Q}^{(\pi)}$. On the other hand, assume $\mathcal{S}_q \notin \mathcal{Q}^{(\pi)}$ has the available time $\sigma_q < \tau^{(\pi)}$. The second condition of the proposition tells us that the workload that can be processed by $\mathcal{S}_q$ from its available time to the current $\tau^{(\pi)}$ is less than or equal to the minimum workload allocated to any system in $\mathcal{Q}^{(\pi)}$. This implies that swapping $\mathcal{S}_q$ with any system in $\mathcal{Q}^{(\pi)}$ cannot shorten the makespan. We finishes the proof. □

Algorithm 5 shows the pseudo-code of *FIX-PDS* algorithm.

First, the target systems in the target systems set S is sorted based on their available times. Then an initial restricted target set $\mathcal{Q}^{(\pi)}$ of size m is generated by iteratively adding to $\mathcal{Q}^{(\pi)}$ new systems. During each iteration, the makespan is updated. The algorithm for the makespan update is same as the one introduced in *FLX-PDS* algorithm (formula 5, formulas 18 and 19, and formulas 24 and 25). It is possible that after k ($k < m$) iterations, the makespan is already shorter than or equal to the available time of any target system not in the restricted target set, which indicates that no improvement will be achieved by involving more target systems into $\mathcal{Q}^{(\pi)}$. There are two options in dealing with this situation: (i) we simple add the next $m - k$ target systems into $\mathcal{Q}^{(\pi)}$ from the ordered set $\mathcal{S}$ such that the size m is reached. We call these systems as "dummy restricted target systems" as they become available after the makespan thus no formal resource reservations will be made on them and no workload will be actually distributed to them; or (ii) we update m to the smaller value k and thus reduce the size of restricted target set and the number of process groups. The *FIX-PDS* algorithm we present implements the first strategy but it can be easily modified to support the second strategy by simply replacing the clauses of adding $m - k$ systems into $\mathcal{Q}^{(\pi)}$ by the clause of updating value of m to k. With the initial restricted target set and makespan, the candidates set $\mathcal{S}^{(c)}$ is constructed. All target systems with available times earlier than the current $\tau^{(\pi)}$ will be added into $\mathcal{S}^{(c)}$. With the formula 26, we search the system in $\mathcal{S}^{(c)}$ that has the largest projected workload and the system in $\mathcal{Q}^{(\pi)}$ with the least workload assigned based on the current $\tau^{(\pi)}$ and swap them if the former can process more. After the swapping operation, the makespan is updated by simulating the *FLX-PDS* scenario with $\mathcal{Q}^{(\pi)}$ as the target system set and the original ω as the input data set size. With the updated makespan, the algorithm reconstructs the candidates set and continues until (i) the candidates set becomes an empty set; or (ii) no swapping operation is possible. Computing the data partitions is the last step of the algorithm. The systems in $\mathcal{Q}^{(\pi)}$ whose available time is after or at the time $\tau^{(\pi)}$ are identified and assigned zero workload while the others are computed by formula 26.

Algorithm 5 Pseudo-code for the *FIX-PDS* algorithm

Input: Target systems set $\mathcal{S}$, input data set size ω, the number of process groups m, pipeline stage p
Output: makespan $\tau^{(\pi)}$, restricted target set $\mathcal{Q}^{(\pi)}$, data segment size for each restricted target system $\omega_i^{(\pi)}$

```
Sort target systems in S based on their available times, the earliest first;
Q^(π) ← ∅;
for i = 1 to m do
    Add system S_i into Q^(π): Q^(π) ← Q^(π) ∪ {S_i};
    Generate new makespan τ^(π);
    if i < m and Target system S_{i+1}'s available time ≥ τ^(π) then
        Add systems S_{i+1}, S_{i+2}, ..., S_m into Q^(π):
        Q^(π) ← Q^(π) ∪ {S_{i+1}, S_{i+2}, ..., S_m};
        Exit for loop;
    end
end
while 1 do
    Generate candidates set S^(c);
    if S^(c) ≠ ∅ then
        Find system S_p in Q^(π) whose available time is before τ^(π) and
        processed the least workload among all systems in Q^(π);
        Find system S_q in S^(c) who will process the largest workload within
        time interval [σ_q, τ^(π)];
        if S_q will process more data than S_p then
            Remove S_p from Q^(π): Q^(π) ← Q^(π) − {S_p};
            Put S_q into Q^(π): Q^(π) ← Q^(π) ∪ {S_q};
            Generate new makespan τ^(π);
        else
            Exit while loop;
        end
    else
        Exit while loop;
    end
end
foreach Target system in restricted target set Q^(π) do
    if Its available time is before the makespan τ^(π) then
        Compute data segment size using formula 26;
    else
        Assign zero workload to it;
    end
end
```

5 Simulation Study

We simulate an ensemble of 200 target systems and investigate the makespan function of the input data set size and the number of process groups m. We also observe the size of the restricted target set for the *FlexMap* scenario. Each target system is characterized by a vector consisting of: the available time σ, the execution rate μ, the duty cycle η, and the data transfer rate R. These four random variables are normally distributed; the mean and the standard deviations of the four random variables are, respectively: 25 and 5 h for the available time σ, 450 and 80 Mdtu/hour for the execution

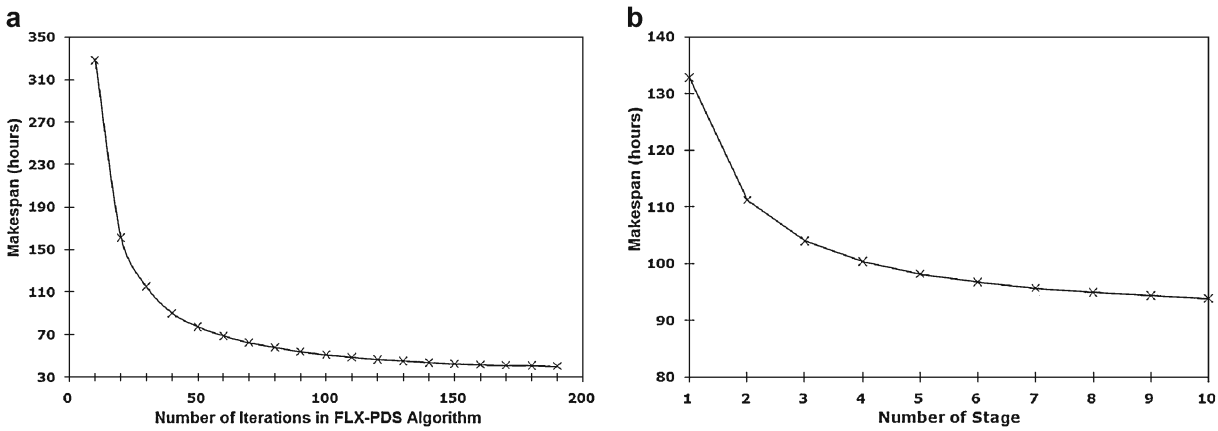

Fig. 3 **a** The makespan (in hours) function of the number of iterations for a fixed input data set size, $\omega = 10^{12}$ *dtu*, for the *FLX-PDS* algorithm. **b** The makespan function of p, the number of pipeline stages for the *FIX-PDS* algorithm; the input data set size is 10^{12} dtu and the number of process groups is 85

rate μ, 0.75% and 0.05% for the duty cycle η, and, finally, 550 and 30 Mdtu/hour for the data transfer rate R. The average execution rate and the data transfer rate are related to the biological application discussed in [21, 46] and measured by running the test process on the multiple clusters in University of Central Florida. The available time and the duty cycle are selected based upon the analysis on the trace of workload on each cluster.

In our simulation we first construct 200 random vectors $(\mu_i, \sigma_i, \eta_i, R_i)$, $\forall \mathcal{S}_i \in \mathcal{S}$ which characterize the target systems set $\mathcal{S}$. Then the DLS algorithms select the restricted target set $\mathcal{Q}^{(\pi)}$, compute makespan and the amount of data allocated to each system. The DLS algorithms are deterministic, thus for a given target systems set configuration and input data set size, the makespan and the generated restricted target set are the same for each run.

Figure 3a shows the makespan function of the number of iterations of the first *for* loop for the *FLX-PDS* algorithm. At each iteration a new target system is added into the restricted target set if the available time of that system is earlier that the current makespan; thus, the makespan monotonously decreases as we iterate. The iterations end after adding 199 systems into the restricted target set. Figure 4a and b show the first 15 and the last 14

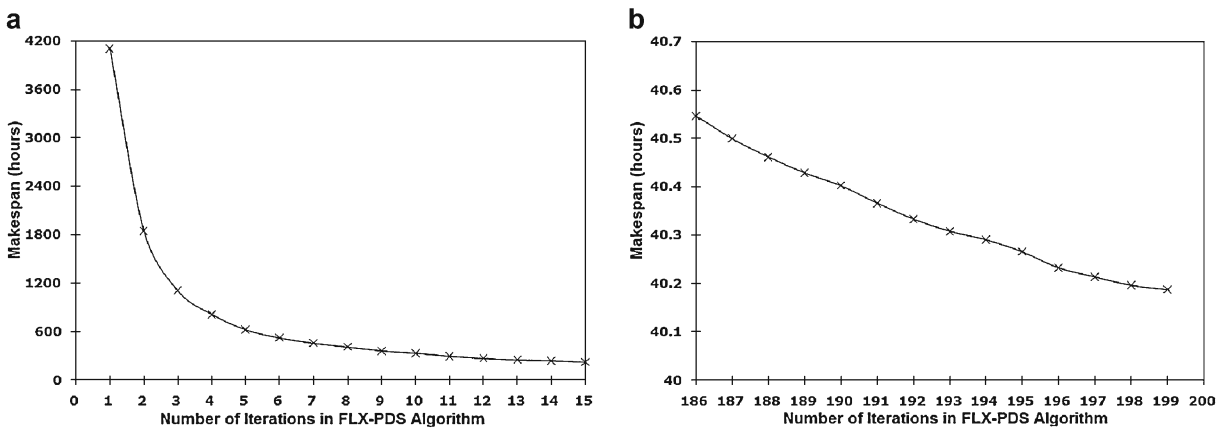

Fig. 4 The makespan (in hours) for the *FLX-PDS* algorithm. **a** The first 15 iterations. **b** The last 14 iterations

iterations, respectively. If we define the speedup as the ratio of the makespan at the first iteration of an interval to the one at the last iteration of the interval, we notice that:

$$Speedup_{1-15} = \frac{4103}{221} = 18.57 \quad while$$

$$Speedup_{186-199} = \frac{40.58}{40.19} = 1.01.$$

This indicates that the benefits of using an increasingly larger number of systems have to be balanced against the additional cost and overhead to coordinate multiple sites. We conclude that the *FLX-PDS* algorithm should have an additional termination condition: `if` $(\tau^{(j)}/\tau^{(j-1)} <$ *MinSpeedupPerIteration*) `terminate`. This conclusion applies to all *FlexMap* algorithms (*FLX-NODS*, *FLX-DSBAT*, *FLX-DSAAT*, and *FLX-PDS*) as they share the same idea in building up the optimal solution.

We now compare the three different data staging strategies for the *FlexMap* family of algorithms, Fig. 5. The three flavors of the algorithm end up using all 200 target systems as soon as the input data set size reaches 11×10^{11} *dtu*, 19×10^{11} *dtu*, and 5×10^{11} *dtu*, respectively. The *FLX-DSAAT* leads to the longer makespan, Fig. 5a, for each input data size and requires more resources. The *FLX-DSBAT* cannot guarantee the existence of the solution when the input data set size is relatively large (in our case when the input data set size reaches 21×10^{11} *dtu*) and the target systems set is relatively small [45], while the other two succeed. The *FLX-PDS* algorithm reduces the makespan substantial comparing to *FLX-DSAAT* algorithm and places a lower load upon resources.

As expected, the number of systems used for a given task increases as the input data set size increases, Fig. 5b. In this experiment, the resource vectors $(\mu_i, \sigma_i, \eta_i, R_i)$, $\forall \mathcal{S}_i \in \mathcal{S}$ are the same, only the data set size increases. For a relatively small input data set size, after a few iterations, the makespan becomes shorter than the available times of the systems outside of the restricted target set. As the input data set size increases, the makespan after the same number of iterations increases and allows us to include more systems in the restricted target set.

The more complex *FixMap* family of algorithms is discussed next. First, we study the the effect of the number p of pipeline stages. As expected, the larger p the shorter is the time to fill-in the pipeline and start the computation and the shorter is the makespan, Fig. 3b.

The effects of the data staging when we limit the number of systems included in the restricted target set are summarized in Fig. 6 for the three flavors of the algorithm. The pipelined version

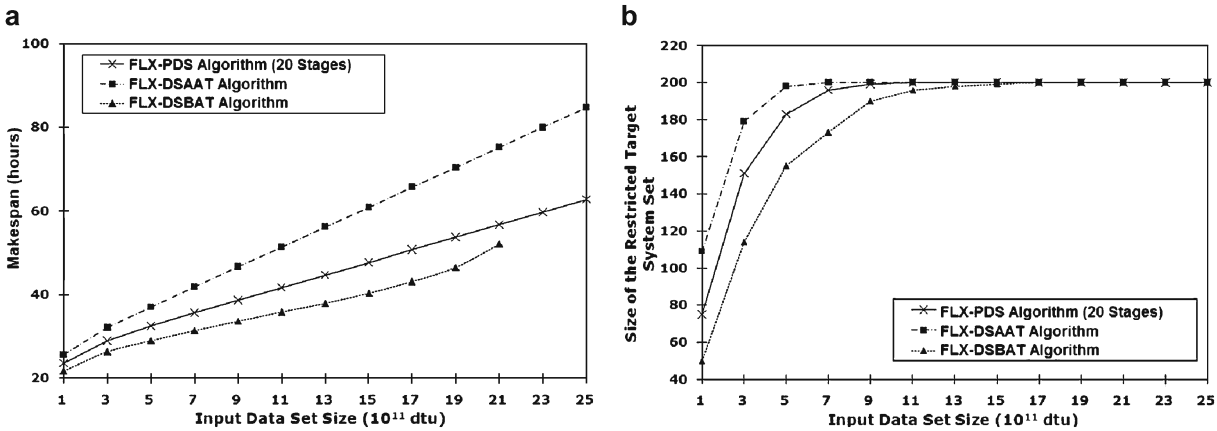

Fig. 5 The effect of the three different data staging strategies when we allow as many systems as possible to be included in the restricted target set (*FlexMap* scenario). **a** The makespan (in hours) function of the input data set size for the *FLX-PDS* with 20 stages, *FLX-DSAAT* and *FLX-DSBAT* algorithms; the input data set size increases in units of 2×10^{11} *dtu*. **b** The size of the restricted target set (the number of systems used) when the input data set size increases in units of 2×10^{11} *dtu* for the three flavors of the algorithm

is better than *FIX-DSBAT* when the number of process groups m is relatively small because the latter chooses systems that allow staging of larger amount of data before their available times in order to guarantee the existence of a solution and the systems with an earlier available time, higher execution rate, and higher duty cycle, could be excluded from the restricted target set [45]. The two versions exhibit the similar behavior when $m \geq 50$, but *FIX-DSBAT* does not produce a solution when $m < 50$. When m is large, the gap between the two algorithms becomes smaller and, eventually, *FIX-DSBAT* outperforms the pipelined version. In order to reduce the makespan, we can either use more systems, or increase the number of pipeline stages. Indeed, the *FIX-PDS* and *FIX-DSBAT* may lead to the same computation completion time of 54 h, but require different numbers of target systems, $m = 90$ and $m = 150$, respectively.

For a relatively small input data size, *FIX-DSBAT* is slightly better than the pipelined version but it fails to produce a solution when the input data set size is larger than 1.3×10^{12} *dtu*. As the input data set size increases, the gap between the algorithms narrows; when the input data set size is larger than 0.9×10^{12} *dtu*, the pipelined version outperforms the *FIX-DSBAT*. The pipelined version is always better than *FIX-DSAAT* and both produce solutions regardless of the input data set size. We also study the speedup when the size of the process groups m increases but the input data set size is fixed, $\omega = 10^{12}$ *dtu*, for the *FIX-PDS* algorithm. The speedup over different ranges of target systems varies:

$$Speedup_{30-110} = \frac{115}{54} = 2.13 \quad while$$

$$Speedup_{110-200} = \frac{54}{40} = 1.35.$$

Again, the benefits of using an increasingly larger number of systems have to be balanced against the additional cost and overhead to coordinate multiple sites.

6 Implementation

The DLS system is implemented as a peer-to-peer system [33]. The DLS code is replicated on all target systems and each system can play the role of Coordinator or Execution Engine. Figure 7 shows the four components of the DLS meta-scheduler: the *Resource Manager (RM)*, the *Monitor (M)*, the *Scheduler (S)* and the *Execution Manager (EM)*. Each system supports bi-directional communication interfaces.

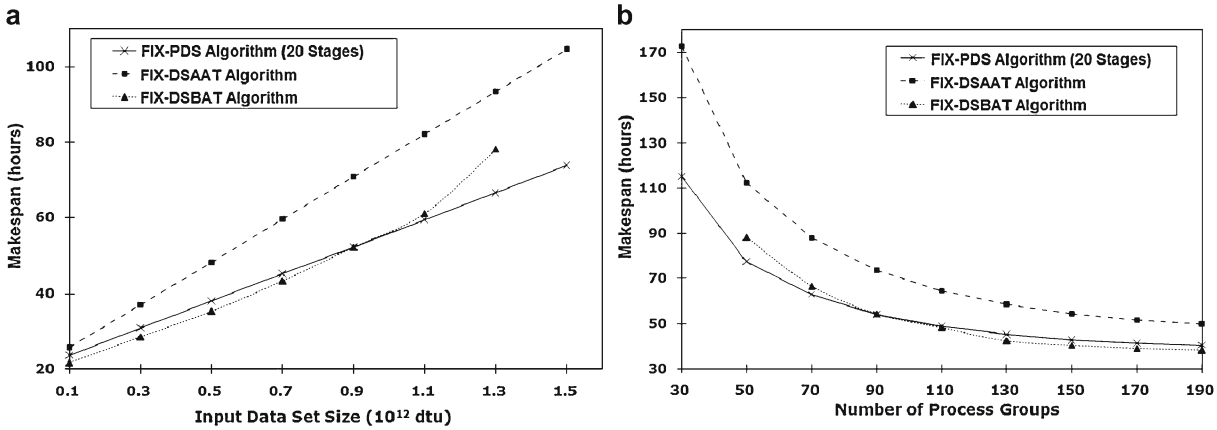

Fig. 6 The effect of the three different data staging strategies when we limit the number of systems included in restricted target set (*FixMap* scenario). We observe the makespan for the *FIX-PDS* with 20 stages, *FIX-DSAAT* and *FIX-DSBAT* algorithms. **a** Function of the input data set size when the number of process groups is 85. **b** Function of the number of the process groups; the input data set size is 10^{12} dtu

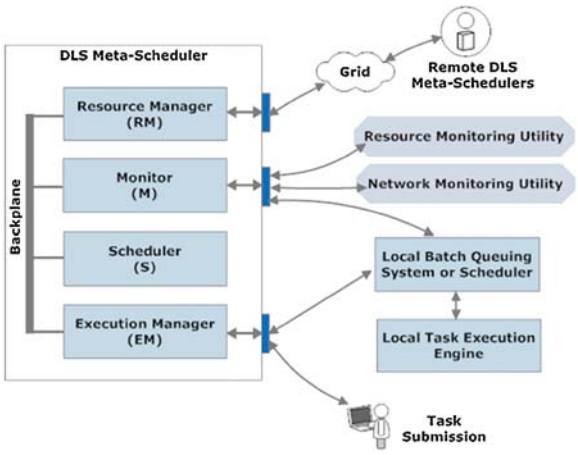

Fig. 7 The architecture of the DLS meta-scheduler with the interface to interact with the external services and devices

The *Resource Manager* initiates the negotiation between the DLS meta-scheduler on the Coordinator site and the DLS meta-schedulers on multiple target systems. Its primary role is to generate the target systems set. The *RM* integrates the security authentication, message communication handling, and the data management functions for the target systems pool. It operates by listening for incoming messages from the remote resources and responding to remote resources on the specific ports through its external interface. The *RM* plays also the role of a message dispatcher and relays messages to the appropriate units.

The role of the *Monitor* is to report on the status of local resources. When a system joins the potential target systems set, the local *Monitor* supplies the Coordinator with information regarding the status of local resources, i.e., the performance evaluators. The *Monitor* uses several utilities to gather the necessary data: (i) the resource status monitoring utilities including the batch queue prediction service on local scheduler such as *PBS* (http://www.pbsgridworks.com/), *Condor* [35], or *Sun Grid Engine* (http://gridengine.sunsource.net/), and the historical execution record analyzer to estimate the system available time, anticipated duty cycle, and the execution rate; (ii) network utilities such as *Network Weather Service (NWS)* [41] and *iperf* (http://dast.nlanr.net/projects/Iperf/) to measure the data transfer rate.

The *Scheduler* executes the DLS algorithms. It generates the mapping, stores it in the Schedule Table and informs the *Resource Manager* to send resource reservation messages to the selected sites.

The *Execution Manager* processes task submission requests and responses to the task execution status queries from the user. It also acts as the interface between the DLS meta-scheduler and the local batch-queuing system or local scheduler on the same site. It can make resource reservation, submit the task execution request, monitor the execution by inquiring the local system, and handle the execution exceptions.

7 Conclusions and Future Work

In this paper we define the *Divisible Load Scheduling* (DLS) problem under multi-port communication model and propose a set of divisible load scheduling algorithms for different data staging strategies. We define four performance evaluators: the available time, the execution rate, the duty cycle, and the data transfer rate, to characterize the heterogeneity of the computing systems. The DLS algorithms we introduced run in polynomial time and generate the optimal mapping from process groups to parallel systems, compute the data allocated to each system, and guarantee the shortest makespan. The conditions for the optimality of the solution for each DLS algorithm are proven.

While the DLS algorithms discussed in this paper are optimal, they require information that is rarely accurate. Indeed, neither the available time and duty cycle, nor the execution rate of a program on a specific target system, or the data transfer rate, can be determined with outmost accuracy. An interesting question is how sensitive are the algorithms introduced in this paper to the input parameters. A quantitative sensitivity analysis is non-trivial due to the interplay among these parameters; we will only present a qualitative analysis.

All algorithms are sensitive to the information regarding the available time. A delay in the

available time of a single system will increase the makespan for DSAAT/PDS strategies. It will also increase the makespan for the DSBAT strategy if this system finishes the computation after the time when all process groups were expected to finish. The fact that a system becomes available early leads to an earlier completion time on this system for the DSAAT and PDS strategies, but will not reduce the makespan if other systems finish according to the original schedule. An earlier available time for the DSBAT strategy may not allow the data staging to complete.

An underestimation of the data transfer rate on a target system does not affect the DSBAT strategy; for the DSAAT and PDS strategies it may reduce the completion time on the system affected by this estimation. On the other hand, an overestimation of the data transfer rate will affect all three strategies. The DSAAT and PDS strategies will experience a longer makespan; the makespan of the DSBAT strategy may not be affected but it is also possible that the algorithm will fail because the data staging could not finish.

Changes in the duty cycle and the execution rate do not have any effect, as long as their product is the same. If their product is underestimated, the makespan will decrease for the DSBAT and DSAAT strategies. It will also decrease for the PDS strategy provided that the data transfer rate is high enough to ensure that the data segment is available before the next stage. As a general rule, an overestimation of the product of the duty cycle and the execution rate will lead to an increase of the makespan for all three strategies.

Although the DSBAT strategy leads to the shortest makespan, it is the most sensitive to input parameters; possible solutions are to: (i) use conservative estimation of the data transfer rate; (ii) do not allow target systems to start the computations before the scheduled available time; (iii) resort to fault-tolerant coordination when insufficient data staging time can be compensated.

The PDS strategy is the least sensitive to the input parameters and it is fault tolerant. After each pipeline stage the Coordinator could use up-to-date information about each system and adjust dynamically the workload assigned. In this case, each execution stage becomes a *test process* for the performance evaluator estimation.

We plan to design a history subsystem to automatically monitor the resource status and record post-mortem data about individual executions. An analysis of the history records will allow us to better estimate the performance evaluators for each target system. Our future work also includes the additional simulation studies to investigate the effectiveness of the fault-tolerant coordination and developing a self-organization model for the DLS system.

Acknowledgements This research was supported in part by National Science Foundation grants ACI0296035, EIA0296179, and by I2Lab Graduate Fellowship at the College of Engineering & Computer Science at University of Central Florida. We also acknowledge the support from a 2007 Walton award from the Science Foundation of Ireland. The authors are grateful to the anonymous reviewers for their suggestions to improve the paper.

References

1. Altilar, D., Paker, Y.: An optimal scheduling algorithm for parallel video processing. In: IEEE Int. Conference on Multimedia Computing and Systems. IEEE Computer Society, Silver Spring (1998)
2. Atallah, M.J., Black, C.L., Marinescu, D.C., Siegel, H.J., Casavant, T.L.: Models and algorithms for co-scheduling compute-intensive tasks on a network of workstations. J. Parallel Distrib. Comput. **16**(4), 319–327 (1992)
3. Baraglia, R., Ferrini, R., Tonellotto, N., Ricci, L., Yahyapour, R.: A launch-time scheduling heuristics for parallel applications on wide area Grids. J. Grid Computing **6**(2), 159–175 (2008)
4. Bataineh, S., Robertazzi, T.G.: Distributed computation for a bus network with communication delays. In: Proc. Conf. Information Sciences and Systems, Baltimore, MD (1991)
5. Beaumont, O., Casanova, H., Legrand, A., Robert, Y., Yang, Y.: Scheduling divisible loads on star and tree networks: results and open problems. IEEE Trans. Parallel Distrib. Syst. **16**(3), 207–218 (2005)
6. Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.: Scheduling Divisible Loads in Parallel and Distributed Systems. IEEE Computer Society, Silver Spring (1996)
7. Bharadwaj, V., Ghose, D., Robertazzi, T.G.: Divisible Load Theory: a new paradigm for load scheduling in distributed systems. In: Cluster Computing on Divisible Load Scheduling, vol, 6, no. 1, pp. 7–18 (2003)
8. Blazewicz, J., Drozdowski, M., Markiewicz, M.: Divisible task scheduling—concept and verification. Parallel Comput. **25**, 87–98 (1999)
9. Blazewicz, J., Drozdowski, M.: Scheduling divisible jobs on hypercubes. Parallel Comput. **21**, 1945–1956 (1995)

10. Blazewicz, J., Drozdowski, M.: The performance limits of a two-dimensional network of load-sharing processors. Found. Comput. Decis. Sci. **21**(1), 3–15 (1996)
11. Braun, T.D., Siegel, H.J., Beck, N., Boloni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B., Hensgen, D., Freund, R.F.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **61**(6), 810–837 (2001)
12. Casanova, H., Legrand, A., Zagorodnov, D., Berman, F.: Heuristics for scheduling parameter sweep applications in Grid environments. In: Proceedings of the 9th Heterogeneous Computing Workshop (HCW00), pp. 349–363 (2000)
13. Cheng, Y.-C., Robertazzi, T.G.: Distributed computation with communication delay. IEEE Trans. Aerosp. Electron. Syst. **24**, 700–712 (1988)
14. Cheng, Y.-C., Robertazzi, T.G.: Distributed computation for a tree network with communication delays. IEEE Trans. Aerosp. Electron. Syst. **26**(3), 511–516 (1990)
15. Cohen, B.: BitTorrent Protocol Specification. http://www.bittorrent.org/protocol.html (2008)
16. Darema-Rodgers, F., Norton, V.A., Pfister, G.F.: Using a single-program-multiple-data computational model for parallel execution of scientific applications. Technical Report RC11552, IBM T.J Watson Research Center (1985)
17. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, ISBN 1-55860-475-8 (2000)
18. Grid Infrastructure Group: TeraGrid. http://www.teragrid.org/ (2009)
19. Hong, Q., Ju, J.: Cooperative task scheduling on workstations network. J. Softw. **9**(1), 14–17 (1998)
20. Jacobson, V.: Congestion avoidance and control. In: Proceedings of ACM SIGCOMM '88 (1988)
21. Ji, Y., Marinescu, D.C., Zhang, W., Zhang, X., Yan, X., Baker, T.S.: A model-based parallel origin and orientation refinement algorithm for CryoTEM and its application to the study of virus structures. J. Struct. Biol. **154**(1), 1–19 (2006)
22. Karatza, H.D.: Gang scheduling and I/O scheduling in a multiprocessor system. In: Proc. Symp. on Performance Evaluation of Computer and Telecommunication Systems (SCSI), pp. 245–252 (2000)
23. Kim, S., Weissman, J.B.: A genetic algorithm-based approach for scheduling decomposable data Grid applications. In: Proc. 33rd Int'l Conf. Parallel Processing (ICPP04), vol. 1, pp. 406–413 (2004)
24. Lee, C., Hamdi, M.: Parallel image processing applications on a network of workstations. Parallel Comput. **21**, 137–160 (1995)
25. Legrand, A., Su, A., Vivien, F.: Minimizing the stretch when scheduling flows of biological requests. Research Report RR2005-48. Ecole Normale Superieure de Lyon (2005)
26. Matthews, W., Cottrell, L.: Achieving high data throughput in research networks. In: CHEP 2001, China (2001)
27. Mathis, M., Semke, J., Mahdavi, J.: The macroscopic behaviour of the TCP congestion avoidance algorithm. Comput. Commun. Rev. **27**(3), 62–82 (1997)
28. McClatchey, R., Anjum, A., Stockinger, H., Ali, A., Willers, I., Thomas, M.: Data intensive and network aware (DIANA) Grid scheduling. J. Grid Comput. **5**, 43–64 (2007)
29. Moges, M.A., Robertazzi, T.G.: Grid scheduling divisible loads from multiple sources via linear programming. In: IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2004). Cambridge, MA (2004)
30. Plastino, A., Ribeiro, C.C., Rodriguez, N.: Developing SPMD applications with load balancing. Parallel Comput. **29**(6), 743–766 (2003)
31. Renard, H., Robert, Y., Vivien, F.: Static load-balancing techniques for iterative computations on heterogeneous clusters. Technical Report RR-2003-12, LIP, ENS Lyon, France (2003)
32. Smallen, S., Casanova, H., Berman, F.: Tunable on-line parallel tomography. In: Proceedings of SuperComputing '01, Denver, CO (2001)
33. Steinmetz, R., Wehrle, K.: Peer-to-peer systems and applications. In: Lecture Notes in Computer Science, vol. 3485. ISBN 3-540-29192-X (2005)
34. Stevens, W.R.: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. The Internet Society (RFC2001) (1997)
35. Thain, D., Tannenbaum, T., Livny, M. (2003) Condor and the Grid. In: Grid Computing: Making the Global Infrastructure a Reality. Wiley, New York (2003)
36. Topcuouglu, H., Hariri, S., Wu, M.-Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002)
37. van der Raadt, K., Yang, Y., Casanova, H.: APSTDV: divisible load scheduling and deployment on the Grid. Technical Report CS2004-0785, Dept. of Computer Science and Engineering, University of California, San Diego (2004)
38. Viswanathan, S., Veeravalli, B., Robertazzi, T.G.: Resource-aware distributed scheduling strategies for large-scale computational cluster/Grid systems. IEEE Trans. Parallel Distrib. Syst. **18**, 1450–1461 (2007)
39. Weissman, J.B.: Prophet: automated scheduling of SPMD programs in workstation networks. In: Concurrency: Practice and Experience, vol. 11, pp. 301–321 (1999)
40. Wolski, R., Spring, N., Hayes, J.: Predicting the CPU availability of time-shared unix systems. In: Proceedings of 8th IEEE High Performance Distributed Computing Conference (HPDC8) (1999)
41. Wolski, R., Spring, N.T., Hayes, J.: The network weather service: a distributed resource performance forecasting service for metacomputing. Future Gener. Comput. Syst. **15**(5,6), 757–768 (1999)
42. Wong, H.M., Yu, D., Veeravalli, B., Robertazzi, T.G.: Data-intensive Grid scheduling: multiple sources with capacity constraints. In: Proc. 16th Int'l Conf. Parallel and Distributed Computing and Systems (PDCS03), pp. 7–11 (2003)

43. Wong, H.M., Veeravalli, B., Barlas, G.: Design and performance evaluation of load distribution strategies for multiple divisible loads on heterogeneous linear daisy chain networks. J. Parallel Distrib. Comput. **65**(12), 1558–1577 (2005)
44. Yang, Y., Casanova, H.: Multi-round algorithm for scheduling divisible workload applications: analysis and experimental evaluation. Technical Report CS2002-0721, Dept. of Computer Science and Engineering, University of California, San Diego (2002)
45. Yu, C., Marinescu, D.C., Siegel, H.J., Morrison, J.P.: A simulation study of data partitioning algorithms for multiple clusters. In: 7th IEEE Int. Symp. on Cluster Computing and the Grid (CCGrid 2007), Brazil (2007)
46. Yu, C., Marinescu, D.C., Morrison, J.P., Clayton, B.C., Power, D.A.: An automated data processing pipeline for virus structure determination at high resolution. In: 6th Int. Workshop on High Performance Structural Biology (HiCOMB), Long Beach, CA, USA (2007)
47. Yu, C., Marinescu, D.C.: Load distribution and co-termination scheduling algorithms for large-scale distributed applications. In; ISCA 21st International Conference on Parallel and Distributed Computing and Communication Systems (PDCCS 2008), New Orlean, LA (2008)
48. Yu, D., Robertazzi, T.: Divisible load scheduling for Grid computing. In: 15th Int'l Conf. Parallel and Distributed Computing and Systems (PDCS2003). IASTED, Anaheim (2003)
49. Zhu, T., Wu, Y., Yang, G.: Scheduling divisible loads in the dynamic heterogeneous Grid environment. In: Proceedings of the 1st International Conference on Scalable Information Systems, Hong Kong (2006)