Future Generation Computer Systems 37 (2014) 321–334

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Pareto frontier for job execution and data transfer time in hybrid clouds

Javid Taheri [a,*], Albert Y. Zomaya [a], Howard Jay Siegel [b], Zahir Tari [c]

[a] School of Information Technologies, The University of Sydney, Australia

[b] Department of Electrical and Computer Engineering, Colorado State University, United States

[c] School of Computer Science, RMIT University, Australia

HIGHLIGHTS

- Particle Swarm Optimization to find the Pareto frontier of data-aware job scheduling.
- Pareto frontier of execution of jobs vs. transfer time of their required data-files.
- Analysis of the influence of Big-data and/or Private-data presence in hybrid clouds.
- Significant outperformance in comparison with current algorithms.
- Fast convergence speed; usually a few minutes for typical hybrid clouds.

ARTICLE INFO

Article history:
Received 10 July 2013
Received in revised form
14 October 2013
Accepted 3 December 2013
Available online 18 December 2013

Keywords:
Big data
Private data
Cloud bursting
Particle swarm optimization
Pareto frontier

ABSTRACT

This paper proposes a solution to calculate the Pareto frontier for the execution of a batch of jobs versus data transfer time for hybrid clouds. Based on the nature of the cloud application, jobs are assumed to require a number of data-files from either public or private clouds. For example, gene probes can be used to identify various infection agents such as bacteria, viruses, etc. The heavy computational task of aligning probes of a patient's DNA (private-data) with normal sequences (public-data) with various data sizes is the key to this process. Such files have different characteristics – depends on their nature – and could be either allowed for replication or not in the cloud. Files could be too big to replicate (big data), others might be small enough to be replicated but they cannot be replicated as they contain sensitive information (private data). To show the relationship between the execution time of a batch of jobs and the transfer time needed for their required data in hybrid cloud, we first model this problem as a bi-objective optimization problem, and then propose a Particle Swarm Optimization (PSO)-based approach, called here PSO-ParFnt, to find the relevant Pareto frontier. The results are promising and provide new insights into this complex problem.

1. Introduction

Cloud computing is a service oriented computing paradigm that has significantly revolutionized computing through its many services – Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) – as well as some of the recently added ones: Database as a Service and Storage as a Service. A large number of application domains have leveraged such services and provided a variety of cloud-based solutions [1]. As a result of such a shift, data have been produced and consumed at much higher rates when compared to traditional grid or cluster systems. Scalable job scheduling and database management systems for both CPU-intensive workloads as well as data-intensive applications have thus become a critical part of current cloud infrastructures [1].

Along with public clouds (e.g., Microsoft Azure [2], and Amazon EC2 [3]), many companies have constructed their own private cloud infrastructure through transforming many of their legacy systems. Although having a private cloud is an advantage for many organizations, sudden needs for extra computing capabilities might lead some of these organizations to outsource portions of their computation needs. This leads to another application development model, known as *cloud bursting*, where an application is run in a private cloud and bursts into (i.e. expands into) a public cloud should the demand for computing exceed available resources. Experts, however, recommend cloud bursting only for non-sensitive applications, for example, those applications that do not require private/sensitive data to run. Because of such security issues, organizations tend to use their private clouds even when performing all computation in public clouds is cheaper. It is also believed that cloud bursting works best when either an applica-

* Corresponding author. Tel.: +61 290369718.
E-mail addresses: javid.taheri@sydney.edu.au (J. Taheri), albert.zomaya@sydney.edu.au (A.Y. Zomaya), HJ@ColoState.edu (H.J. Siegel), zahir.tari@rmit.edu.au (Z. Tari).

0167-739X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.future.2013.12.020

tion does not have complex interdependency with other applications, or when applications are moved to public clouds so that local resources are spared for more business-critical applications [1].

Big-data is another reason why many computations must be performed externally to one's private cloud. Although the definition of big-data has not been fully agreed upon yet, it is always used to describe a voluminous amount of unstructured or semi-structured data, usually on the order of terabytes and beyond, created from one or multiple sources. Big-data is usually defined using the following "4-V's" [4]: Volume, Variety, Velocity, and Variability. *Volume* refers to data that is large in size; *Variety* refers to a data set composed of many sources and which is probably unstructured; *Velocity* refers to change of the data rate coming to a process; and *Variability* refers to the fact that sometimes it is almost impossible to predict value of information that may come to you tomorrow. All these V's imply that computation must be usually performed where the data resides; Volume of data also further restricts it to a no-replication policy for big-data sometimes. However, from the security point of view, organizations may still decide to replicate big-data on their private clouds for further analysis because they might not be able to tolerate delays of such large transfers from public clouds to their private infrastructure.

From the scheduling point of view, providing solutions that consider all the aforementioned restrictions (i.e. security for private data and size for big-data) and efficiently execute a batch of jobs in a hybrid (private plus public) cloud is far more difficult than the original data-dependent job scheduling problem in grids. In fact, such solutions must consider not only location of data-files in addition to computational capacity of clouds in scheduling decisions, but also the privacy and unusual size of few very large-sized data-files in a system. Because of such extra difficulties in dealing with both complex restrictions, many proposed schedulers of such hybrid systems are usually over-simplified to produce the fastest and mostly the simplest solutions.

After close examination of many already proposed schedulers, we noticed that no proper investigation is ever conducted for hybrid clouds to discover the inter-relationship between the execution time of a batch of jobs and the transfer time required to deliver (cache or replicate) their required data when the size of data is large [5–13,15,16]. Prior investigations for grids showed that these two objectives usually contradict each other where minimizing one usually results in compromising the other [13,15,17]. For example, minimizing the execution time of a batch of jobs requires scheduling jobs to clouds with more computing cores, whereas minimizing the transfer time of data requires scheduling jobs to clouds where the needed data already reside.

We have also realized that most of such techniques are usually tailor-made to either minimize the execution time of jobs or the transfer time of all data-files in a system, with very few exceptions that consider both. We also realized that it is impossible to measure the true performance of such algorithms when migrated to clouds without knowing their optimal (either theoretical or computational) scheduling solutions. PSO-ParFnt is a technique we designed to address this issue, because it is designed to computationally find the Pareto frontier of hybrid clouds and reveal the true performance of different algorithms in various situations. All programs that may require cloud-bursting of some or all of their processes can directly benefit from the outcome of PSO-ParFnt to balance the execution time of their jobs versus the amount of data that must be transfered to/from the cloud. Astronomy applications such as Montage [18], bioinformatics applications such as DNA sequencing [19], and climate modeling applications [20] are among many applications with such nature.

This paper proposes an approach to model such complex relationship and analyze its trade-offs. To this end, we first model the problem as a bi-objective optimization problem and then use our proposed Particle Swarm Optimization (PSO) approach to compute the Pareto frontier of the trade-offs. The Pareto fronts for our case studies are then aligned with several already proposed hybrid scheduling algorithms to (1) validate the quality of our computed Pareto fronts, and (2) validate the quality of a few already proposed solutions by measuring their distance from the calculated Pareto fronts.

This work is organized as follows. Section 2 highlights related work followed by preliminaries of the proposed approach in Section 3. Section 4 details the solution for the computation of the Pareto front. Section 5 overviews the simulation setup and details the results of the simulation studies. Section 6 analyses the results and summarizes the important outcomes. Finally, Section 7 concludes our work and highlights future directions for study.

2. Related work

The work in this paper is closely related to three main aspects of cloud computing: big-data transfer complexities, data privacy, and scheduling data dependent jobs in hybrid clouds. Because comprehensive literature reviews for each of these topics are beyond the scope of this study, we provide sufficient details on each aspect by covering only issues directly related to our proposed solutions.

Big-data: In information technology, big-data [21,22] consists of data sets that usually grow too large and become complex to handle using current database management tools; capture, storage, search, share, analytics, and visualization are among some of the well-known issues [23]. Despite its many challenges, the trend of incorporating big-data is still continuing as it has the potential to provide deeper analysis to detect business trends, prevent diseases, combat crime, and others [24]. Data sets are continually growing in size as they are usually collected from a variety of sources, such as ubiquitous information-sensing mobile devices, aerial sensory technologies (remote sensing), software logs, cameras, microphones, radio-frequency identification readers, and wireless sensor networks [25]. As a result, the world's technological capacity to store information has roughly doubled every 40 months [26]. Along with this trend, database requirements are also vastly different from one organization to another. Greenplum [27] is an example of such databases where the emphasis is to provide very fast data loading to other applications. Fig. 1 conceptually shows how different data-file systems can be categorized according to their structure and scaling capabilities [28]. As capacity needs grow, in scale-up storage systems, disks are added behind an already existing storage controller; in scale-out systems, complete storage elements are added to the system. Loosely structured and scale-out architectures are essential and favored for big-data initiatives.

Private-data: Fujitsu conducted a global survey in October 2010 to study consumer attitudes and concerns about having their personal data in a cloud [29]. The survey revealed that although consumers are excited and intrigued by opportunities that arise from cloud computing, they are also deeply concerned about their data privacy and risks involved in sharing data. There have been several legal studies to properly define "personal data" in the cloud [30]. Serious questions remain as to whether databases containing anonymized, pseudoanonymized, encrypted, and fragmented data in transmission and/or storage should still be considered as "private" or not. As a result of this, many service providers, such as financial institutions, prefer not to take the risk of using cloud bursting in order not to compromise the safety of their data.

Data aware job scheduling: For jobs with file dependencies, especially data-intensive ones, scheduling not only involves computational concerns, but also the data management to access required data-files. Data replication techniques have been around for many

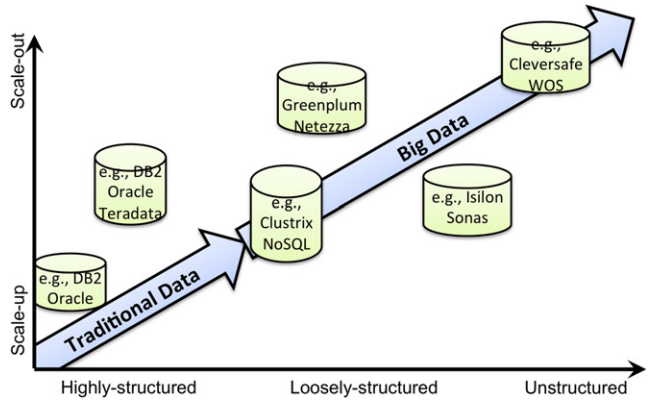


Fig. 1. Positioning of databases for the Big-data Era.

years, for example, to facilitate accessibility of jobs to data. Naturally, clouds can also benefit from the use of efficient algorithms to manage data during out-bursting or anonymizing calculations when performing private computations in public clouds. Because grids and clouds share many underlying concepts, many algorithms to replicate data in grids can be easily extended to work in clouds.

Six major classes of replica strategies presented in [31] can be easily deployed for clouds; they are: (1) no-replication, (2) best-client, (3) cascading, (4) plain-caching, (5) caching plus cascading, and (6) fast-spread. In the no-replication policy, data-files are never replicated or cached; they are always downloaded upon request. In the best-client policy, data-files are replicated on storage nodes that have the highest number of requests for the file. In cascading, once popularity of a data-file exceeds a certain threshold in a given time interval, it is replicated to the best-client storage. In plain-caching, every client that requests a file also stores a copy of it. In caching plus cascading, plain-caching and cascading are simultaneously performed. In fast-spread, data-files are proactively replicated along the path between storage nodes and clients. Also, several techniques have been proposed to not only replicate data but also schedule jobs to grid nodes [31]; the following four classes of algorithms have been identified: (1) JobRandom, (2) JobLeastLoaded, (3) JobDataPresent, and (4) JobLocally. In JobRandom, jobs are randomly distributed among computing nodes. In JobLeastLoaded, jobs are scheduled to nodes with the least queue lengths; i.e., with the least number of jobs waiting to run. In JobDataPresent, jobs are scheduled to nodes which either already have the required data or can download them faster in comparison with others. In JobLocally, each job is run at the cluster where it is submitted regardless of the amount of data to be downloaded.

Based on the above, the following scheduling systems are probably the most comprehensive grid/cloud schedulers to simultaneously assign jobs and replicate data-files. Data Intensive and Network Aware (DIANA) scheduling [32,33] is designed based on the real GILDA [34] and CMS [35] grid systems. In this approach, jobs are first assessed to determine their execution category. For data-intensive applications, jobs are migrated to computing nodes (CNs) with minimum access (download) time to their required data-files. For computationally intensive jobs, on the other hand, data-files are migrated/replicated to storage nodes (SNs) with minimum access (upload) time to their dependent jobs. In both cases, the decision is made based on: (1) capacity of SNs, (2) speed and number of computers/processors in CNs, and (3) network links connecting SNs and CNs. BestMap [17] uses two alternative mechanisms to iteratively (1) minimize execution time of jobs, and (2) minimize delivery time of all data-files through replication. Unlike DIANA, BestMap does not categorize jobs, but always tries to find the most suitable CN or SN to schedule a job or replicate its dependent data-files at each stage. JDS-BC [13] is a heuristic approach to solve the stated problem. JDS-BC uses information about already scheduled jobs or replicated data-files in making decisions for scheduling or replicating current jobs or data-files. JDS-BC, which is based on the bee colony optimization technique, models jobs as bees and CNs as hives. Here, upon scheduling any job, the scheduled job reports its received "benefit" from a given allocation. Such benefit is designed so that schedulers concurrently balance the execution time of jobs and reduce the overall delivery time of all data-files. Chameleon [36], also known as FLOP, targets CNs that can start executing jobs straight away; i.e., it always migrates jobs to CNs that can start executing them faster than others. Although Chameleon/FLOP does not initially consider data-file download times during its scheduling process, it always replicates data-files upon scheduling jobs to provide faster upload times to them.

3. Preliminaries

This section summarizes some preliminary information required to better explain our approach. We first mathematically define the stated bi-objective optimization problem. This is followed by an explanation of the PSO technique before detailing how we used it in our approach. Finally, a Pareto frontier curve is mathematically defined.

3.1. Framework

In this work, we made an attempt to incorporate as many features as possible from previous approaches and design of the proposed framework to consist of heterogeneous: (1) private clouds, (2) public clouds, (3) storage clouds, (4) interconnecting network, (5) schedulers, (6) users, (7) jobs, and (8) data-files. Private clouds are assumed to be independent and do not share information with each other; each private cloud has enough computational power and storage capability to perform its local jobs and store its private data. Public clouds' computational and storage capabilities can be used by private clouds for out-bursting their computations. Private clouds can also replicate their data to public clouds to improve access to their data. Storage clouds do not have computational powers and can only provide data to public/private clouds should they require.

To better model different cloud types, we designed the following two basic elements: cloud computing component (CCs) and cloud storage component (SCs). Public and private clouds are each made of a CC attached to a SC; storage clouds only have a SC. **CCs** in our framework represent heterogeneous computing environments with various CPU types; they are assumed to have enough local cache to hold data-files for executing a batch of jobs. **SCs** host data-files required by jobs. Although from the optimization point of view there is no difference between SCs attached to CCs and the ones isolated, attached SCs can upload data-files to their associated CCs must faster than the isolated ones; whereas isolated SCs usually have more capacity than the attached ones. CCs and SCs are connected through an **interconnection network** comprised of individual links. Each link in this system has its own characteristics and is modeled using two parameters: delay and bandwidth. These links are assumed independent; and thus, different CCs/SCs can simultaneously transfer data-files to each other. Attached CCs and SCs are assumed to have LAN connections; all other links are assumed to have WAN connections. **Schedulers** are decision makers of the whole system; they accept jobs and data-files from users/systems and schedule or replicate them to relevant CCs and SCs, respectively. **Users** generate jobs and submit them to schedulers to be executed by CCs. **Jobs** have heterogeneous execution times along with a heterogeneous list of required data-files.

Table 1
Summary of notation.

N_{CC}, N_{SC}, N_J, N_D	Total number of CCs, SCs, jobs, and data-files in a systems
N_R	Maximum number of replicas for each datafile
D_i^{size}, SC_j^{size}	Size of datafile #i, and SC #j
$J_i^{ST}, J_i^{EX}, J_i^{TT}$	Start, execution, and transfer time to download all data-files required to execute job #i
DS_j^{size}	Total size of data-files addressed by data file-set #j

Data-files are assumed to be owned by SCs and are allowed to have a predefined number of replicas in a system. Schedulers can only delete or move replicas; the original copies are always kept intact. Data-files are also assumed to contain private-, big-, or normal-data (neither private nor big). Private data is stored in SCs of private clouds only; these files cannot be replicated. Big-data and normal-data can be stored on both private and/or public clouds; there is no restriction in replicating these two types of data.

3.2. Problem statement

Data aware job scheduling (DAJS), inspired by grids, is a bi-objective optimization problem and is defined as assigning jobs to CCs and replicating data-files on SCs to concurrently minimize (1) the overall execution time of a batch of jobs as well as (2) the transfer time of all data-files to their dependent jobs [13,15,17,32,33]. Because these two objectives are usually independent, and in many cases even conflicting, minimizing one objective usually results in compromising the other. For example, achieving lower execution time requires scheduling jobs to clouds with more free cores, whereas achieving lower transfer times requires using links with higher bandwidths in a system. Similar to other approaches, we also assume that if several jobs in a CC require the same data-file, the requested data-file will be downloaded once and then stored in a local repository (cache) for further local requests [15–17,33,36,37]. Table 1 summarizes the notation we use throughout this work.

To mathematically formulate DAJS, assume that jobs should be partitioned into several job-sets, $\{JS_1, JS_2, \ldots, JS_{N_{CC}}\}$, to be executed by CCs, and data-files should be partitioned into several data-file-sets, $\{DS_1, DS_2, \ldots, DS_{N_{SC}}\}$, to be replicated onto SCs. For example, if $N_J = 9$ and $N_{CC} = 3$, then *JobSets* $= \{\{1, 5, 7\}, \{2, 4, 8, 9\}, \{3, 6\}\}$ means jobs in $JS_1 = \{J_1, J_5, J_7\}$, $JS_2 = \{J_2, J_4, J_8, J_9\}$, and $JS_3 = \{J_3, J_6\}$ are assigned to be executed in CC_1, CC_2, and CC_3, respectively. Based on this model, DAJS is defined as finding elements of job-sets and data-file-sets to minimize the following two objective functions.

$$\begin{cases} 1.\ MIN\ \underset{i=1}{\overset{N_{CC}}{MAX}}\ JS_i^{EX} \\ 2.\ MIN\ \sum_{i=1}^{N_{CC}} JS_i^{TT} \\ s.t. \\ DS_i^{size} \le SC_i^{size}; \quad i = 1, \ldots, N_{SC}. \end{cases} \tag{1}$$

Here, if $JS_i = \{J_1, J_2, \ldots, J_K\}$ contains K jobs scheduled to be executed by CC_i, then the execution time and the total transfer time of this job-set can be calculated as follows:

$$JS_i^{EX} = \underset{k=1}{\overset{K}{MAX}} \left(J_k^{ST} + J_k^{EX}\right) \tag{2}$$

$$JS_i^{TT} = \sum_{k=1}^{K} J_k^{TT}. \tag{3}$$

In DAJS formulation, the constraint is to guarantee that the total size of all data-files each SC hosts is less than its total capacity. For local schedulers inside each CC, the overall execution time of a set of jobs greatly depends on each CC's local scheduling policy; we however chose First-Come-First-Served with backfilling for this purpose as it usually results in near-optimal deployment of resources when a large number of jobs are submitted [38].

3.3. Particle swarm optimization

Particle Swarm Optimization (PSO) is among the most well-known nature-inspired techniques to find optimal solutions of complex problems [39]. PSO originates from the following two concepts: (1) swarm intelligence usually observed in population based animals such as birds to achieve global interests such as find food sources; and (2) evolutionary computing where several solutions are combined to produce higher quality ones. In PSO, several solutions (particles) of a given problem are usually randomly generated to initialize a "swarm"; then, iteration by iteration, swarm particles try to improve their quality by moving toward better particles/solutions already found—either by themselves or by other particles. Based on such information, each particle calculates its next location by using the following formula:

$$x(t+1) = x(t) + v(t) \tag{4}$$

where $x(t)$ is a vector to represent the current location of a particle, and $v(t)$, also a vector, is the velocity of the particle at time t and is calculated at each iteration as follows:

$$\begin{aligned} v(t) = {} & w_v r_v v(t-1) + w_l r_l (x^{best}(t) - x(t)) \\ & + w_g r_g (g^{best}(t) - x(t)). \end{aligned} \tag{5}$$

Here, $x^{best}(t)$ and $g^{best}(t)$ represent the best solutions found by the particle itself or the swarm as a whole up to time t, respectively. w_v, w_l, and w_g are three weights (scalars) to engage the previous velocity, the best local solution, and the best global solution, respectively. r_v, r_l, and r_g are random numbers in [0.0, 1.0]. Algorithm 1 shows steps of a typical PSO-based solution.

Step 1: Generate the initial swarm
Step 2: Repeat until a termination condition is met
- 2.1 Evaluate fitness value of each particle
- 2.2 Update local and global best solutions
- 2.3 Calculate velocity for each particle and update its location

Algorithm 1: A typical PSO optimization procedure

3.4. Pareto frontier curve

Pareto efficiency, or Pareto optimality, is a concept in economics with applications in engineering and social sciences [15]. It is defined as allocating goods among individuals where no individual can improve his/her situation without worsening another's; a Pareto frontier is a set of Pareto efficient choices. Connecting members of such set generates a Pareto frontier curve (Pareto front for short). The Pareto front is particularly useful in engineering where designers can make trade-offs within sets, rather than considering the full range of every parameter.

To formally define a Pareto front, consider a design space with n real parameters and m measurement criteria for each design. Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be the function that assigns a criteria space point $f(x)$ to each design space point x. Also let X and $Y = f(X)$ be the set of all feasible solutions in $\mathbb{R}^n$ and their measured values in $\mathbb{R}^m$, respectively. The Pareto front (Y^*) is a subset with maximal elements of points in Y where no point is strictly dominates the others. For minimization problems, point y^* dominates y if $\{\forall i : y_i^* \le y_i\}$ and $\{\exists i : y_i^* < y_i\}$. Fig. 2 shows an example of such frontier curve for a two-dimensional criteria space.

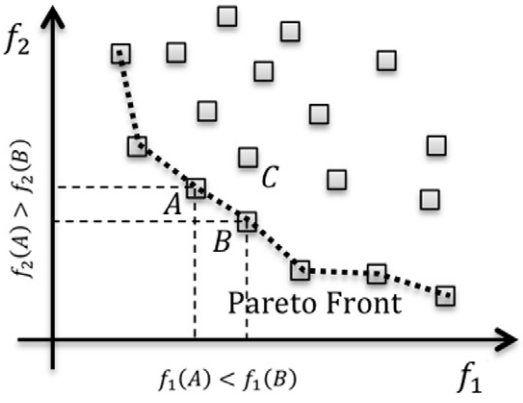


Fig. 2. A Pareto frontier curve.

4. PSO for finding DAJS's Pareto front

4.1. Overview

This section provides details of the proposed algorithm, namely PSO-ParFnt, to find the Pareto front for the execution time of a batch of data dependent jobs versus the transfer time of their required data in hybrid clouds. Because the original PSO is designed to optimize one objective only, we carefully modified it so that it suits our specific needs for solving the bi-objective DAJS problem here. Also, to obtain a well-distributed Pareto front across both objectives of our DAJS problem, we designed several swarms to collaboratively explore different sections of such Pareto frontier curve. Each swarm works independently and regularly exchanges particles with other swarms. The Pareto frontier of a DAJS problem is computed based on the union of all particles from all swarms. Algorithm 2 and Fig. 3 describe/show the PSO-ParFnt procedure.

4.2. Generate random particles

To initialize swarms (Step 1 in Algorithm 2), we first need to design particles able to represent all possible solutions of the stated problem. In PSO-ParFnt, each particle consists of two parts for jobs and data-files. The first part is to represent where each job must be executed; the second part determines replica locations for each data-file. It is defined as follows:

$$Prtcl = \left(x_1, x_2, \ldots, x_{NJ} \;\middle|\; \begin{matrix} \langle y_1^1, y_2^1, \ldots, y_{NR}^1 \rangle, \\ \langle y_1^2, y_2^2, \ldots, y_{NR}^2 \rangle, \\ \ldots, \\ \langle y_1^{ND}, y_2^{ND}, \ldots, y_{NR}^{ND} \rangle, \end{matrix} \right) \quad (6)$$

where, $x_i \in [1, N_{CC}]$ represent the CC that is responsible to execute the ith job (J_i) and $y_j^k \in [1, N_{SC}] \bigcup \{-1\}$ addresses where the jth replica of the kth data-file (D_k) must be stored; $y_j^k = -1$ means no replication. Note that the original copy of each data-file is always accessible even if no replica is made for it. For example,

$$Prtcl = \left(1, 2, 2, 1, 2, 3, 4, 4, 3, 1 \;\middle|\; \begin{matrix} \langle 1, -1 \rangle, \langle 1, 3 \rangle, \langle 2, 3 \rangle, \\ \langle 2, -1 \rangle, \langle 3, -1 \rangle \end{matrix} \right) \quad (7)$$

implies that job sets $\{J_1, J_4, J_{10}\}$, $\{J_2, J_3, J_5\}$, $\{J_6, J_9\}$, and $\{J_7, J_8\}$ must be executed by CC_1, CC_2, CC_3, and CC_4, respectively; also, D_1 must have only one replica on SC_1 (two copies in total to be accessed by all jobs), D_2 must have two replicas on SC_1 and SC_3, D_3 must have two replicas on SC_2 and SC_3, D_4 must have only one replica on SC_2, and D_5 must have one replica at SC_3.

Input: (1) Bag-of-Jobs with full description for each job, (2) Detailed characteristic of location and size of all big-/private-/normal-data-files, (3) Network characteristics with full description of each link, and (4) Detailed characteristics of all private and public clouds
Output: The Pareto frontier curve of the execution time of jobs versus the transfer time of their required data-files

Step 1: Generate random particles (Algorithm 3), namely $RndPrtcls$
Step 2: Initialize swarms, $\{Sw_1, Sw_2, \ldots, Sw_{Nsw}\}$, with $RndPrtcls$ and set their targets. (Algorithm 4)
Step 3: Repeat for a predefined number of iterations
3.1 Combine particles from all swarms and make $AllPrtcls = \bigcup_{i=1}^{N_{sw}} SW_i$
3.2 Find the Pareto front from $AllPrtcls$
3.3 Update swarms' targets (Algorithm 4)
3.3 For each swarm
Update all particles (Algorithm 6)
Remove duplicate particles
Fill and trim swarm particles (Algorithm 5)
Update the best particle

Algorithm 2: PSO-ParFnt optimization procedure

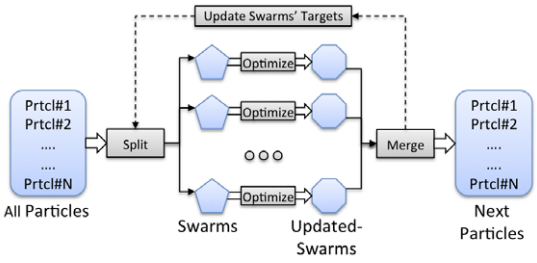


Fig. 3. PSO-ParFnt's procedure.

The initial random population (Step 1 in Algorithm 2) consists of several particles; each generated through Algorithm 3, where Step 1 allocates enough memory to save a particle. Step 2 randomly assigns/schedule jobs to CCs. Step 3 randomly replicates data-files on SCs; here, if the size of a data-file is larger than its randomly selected SC or it is non-replicable, then no replication is performed.

Input: (1) Detailed characteristics of all private and public clouds, (2) Detailed characteristics of all jobs and big-/private-/normal-data-files
Output: A random feasible solution, $Prtcl$.

Step 1: Allocate memory to store a vector with size $N_J + N_D \times N_R$, namely $Prtcl$.
Step 2: For $job = 1$ to N_J
Generate a uniformly distributed random number $1 \le Rnd \le N_{CC}$
Let $Prtcl(job) = Rnd$
Next job
Step 3: For $df = 1$ to N_D
For $rep = 1$ to N_R
Generate a uniformly distributed random number $0 \le Rnd \le N_{SC}$
If ($Rnd \neq 0$ AND $SC_{Rnd}^{size} \ge D_{df}^{size}$ AND df is replicable)
then
$Prtcl(df \times N_R + rep) = Rnd$
$SC_{Rnd}^{size} = SC_{Rnd}^{size} - D_{df}^{size}$
else
$Prtcl(df \times N_R + rep) = -1$
Endif
Next rep
Next df

Algorithm 3: Generating a sample particle

Table 2
Notation for swarms and particles.

$Sw_s^{TgET}, Sw_s^{TgTT}, Sw_s^{TgW}$	Execution time target, transfer time target, and target's bandwidth for swarm #s
$Sw_s^{TgType}, Sw_s^{size}$	Type and size of swarm #s
$Sw_s = \{P_1^t, \ldots, P_K^t\}$	Locations of ($K = Sw_s^{size}$) particles compose swarm #s at time t
$\|P_1, P_2\|$	Distance between two particles
$\|P\|$	Length of a particle equals to $N_J + N_D \times N_R$
$P(x)$	xth element of a particle

4.3. Swarms

Because of the complexity of the stated problem, we decided to design several swarms to cooperatively work in finding the desired Pareto front. The various experiments we conducted showed that having multiple less-crowded swarms with individual targets are more efficient than a single crowded one to find the Pareto frontier of the stated DAJS. These experiments also showed that for the stated DAJS problem in particular, a single swarm could not homogeneously explore all sections of the desired Pareto front. As a result, we observed that most particles are superfluously targeting specific regions of a Pareto front, whilst its other sections were harshly ignored. Table 2 summarizes the notation we use to describe different characteristics of swarms and particles in this work.

There are three type of swarms in PSO-ParFnt: *ExeTm*, *TransTm*, and *Both*. The first swarm type (*ExeTm*), only focuses on reducing the execution time of all particles of a swarm, while keeping their data-file transfer times within a predefined band; i.e., minimizing the first objective of DAJS, while restricting the other. This can be mathematically formulated as follows:

$$\begin{cases} MIN \ \underset{i=1}{\overset{N_{CC}}{MAX}} \ JS_i^{EX} \\ s.t. \\ TT^{MIN} \leq \sum_{i=1}^{N_{CN}} JS_i^{TT} \leq TT^{MAX} \\ DS_i^{size} \leq SC_i^{size}; \quad i = 1, \ldots, N_{SC}. \end{cases} \tag{8}$$

Fig. 4(a) shows five SW^{TgET} swarms with such individual targets. As can be seen, these swarms were able to efficiently push back their targeted section of the Pareto front across several iterations. The second swarm type (*TransTm*) follows the same procedure, but to reduce the transfer time of all particles, while keeping their execution times within a predefined band. The third swarm type (*Both*) aims to reduce the summation of execution time and transfer time of all particles in a swarm. Fig. 4(b)–(c) shows how *TransTm* and *Both* swarm types push back their targeted sections of a Pareto front over several iterations.

Algorithm 4 shows how the randomly generated particles (*RndPrtcls*) are used to initialize several swarms and set their targets. In this algorithm, Steps 1–2 find the minimum and the maximum execution time and transfer time of all particles in a list of randomly generated particles. Step 3 restricts the maximum value to be at most three times the minimum value. Doing this will help the generated swarms to avoid outlier particles; the ratio of three is set empirically. Step 4 finds two bandwidths to homogeneously split sections of a desired Pareto front among the requested number of swarms of each type. Steps 5–7 set targets for the generated swarms and add them to the swarm list. After setting the targets for each swarm, they must be filled with appropriate particles to help them achieve their desired targets. Algorithm 5 details how each swarm is filled and trimmed with most suitable particles from a list of available particles.

Input: (1) Randomly generated particles ($RndPrtcls$), (2) SW^{size}, N_{TgET}, N_{TgTT}, and, N_{TgB}
Output: A list of targeted swarms, $SwLst$

Step 1: Find $\widehat{ET}$, $\widetilde{ET}$ as the maximum and minimum execution time of all particles in $RndPrtcls$, respectively.
Step 2: Find $\widehat{TT}$, $\widetilde{TT}$ as the maximum and minimum execution transfer time of all particles in $RndPrtcls$, respectively.
Step 3: Let
$\widehat{ET} = \min(\widehat{ET}, 3\times\widetilde{ET})$, $\widehat{TT} = \min(\widehat{TT}, 3\times\widetilde{TT})$
Step 4: Let
$B^{ET} = \left(\widehat{ET} - \widetilde{ET}\right)/N_{TgET}$,
$B^{TT} = \left(\widehat{TT} - \widetilde{TT}\right)/N_{TgTT}$
Step 5: For $s = 1$ to N_{TgET}
Create a new swarm, Sw
$Sw^{TgType} = "ExeTm"$
$Sw^{TgTT} = (s-1)\times B^{TT} + \widetilde{TT} + B^{TT}/2$
$Sw^{TgET} = 0$
$Sw^{TgW} = B^{TT}$
Add Sw to $SwLst$
Next s
Step 6: For $s = 1$ to N_{TgTT}
Create a new swarm, Sw
$Sw^{TgType} = "TransTm"$
$Sw^{TgTT} = 0$
$Sw^{TgET} = (s-1)\times B^{ET} + \widetilde{ET} + B^{ET}/2$
$Sw^{TgW} = B^{ET}$
Add Sw to $SwLst$
Next s
Step 7: For $s = 1$ to N_{TgB}
Create a new swarm, Sw
$Sw^{TgType} = "Both"$
$Sw^{TgET} = Sw^{TgTT} = Sw^{TgW} = 0$
Add Sw to $SwLst$
Next s

Algorithm 4: Generate targeted swarms

Input: (1) A list of all available particles $AvlPrtcls$, (2), All empty swarms, $\{Sw_1, \ldots, Sw_N\}$
Output: Swarms filled with particles

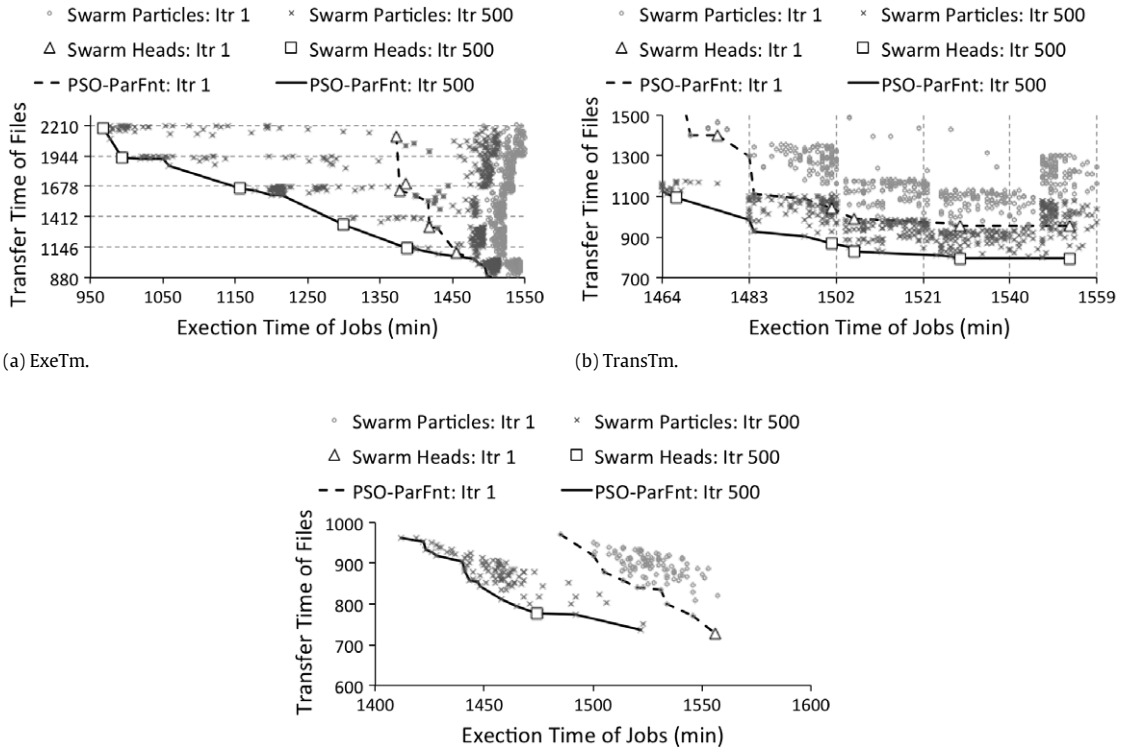
For $s = 1$ to N
Switch Sw_s^{Type}
Case "$ExeTm$"
$QlfPrtcls = \left\{ P_i \in AvlPrtcls;\ where; \left| P_i^{ET} - Sw_s^{TgET} \right| \leq Sw_s^{TgW} \right\}$
Sort $QlfPrtcls$ in ascending order of the execution time of its particles
Copy up to Sw_s^{size} particles from $QlfPrtcls$ to Sw_s
Case "$TransTm$"
$QlfPrtcls = \left\{ P_i \in AvlPrtcls;\ where; \left| P_i^{TT} - Sw_s^{TgTT} \right| \leq Sw_s^{TgW} \right\}$
Sort $QlfPrtcls$ in ascending order of the transfer time its particles
Copy up to Sw_s^{size} particles from $QlfPrtcls$ to Sw_s
Case "$Both$"
Sort $AvlPrtcls$ in ascending order of Execution time plus transfer time of its particles
Copy up to Sw_s^{size} particles from $AvlPrtcls$ to Sw_s
Endswitch

If Sw_s does not have enough particles, **then**, fill it with random particles from $AvlPrtcls$
Next s

Algorithm 5: Fill and Trim procedure

4.4. PSO-ParFnt optimization cycle

Step 3 of Algorithm 2 performs the main optimization of PSO-ParFnt after its initialization. In this step, which is repeated for



(a) ExeTm.

(b) TransTm.

(c) Both.

Fig. 4. Pareto frontier shift of ExeTm-/TransTm-/Both-Targeted swarms after 500 iterations.

a predefined number of iterations, particles of all swarms are combined first, and then used to find the Pareto front. In Step 3.1, the combined particles are also used to set new targets for each swarm. Here, because a Pareto front is usually shifted toward lower values, for both objectives, after several iterations, it is necessary to adjust new targets. Here, we follow the same steps already described in Algorithm 4 to set a new target, but without adding new swarms; i.e., the range of all particles is used to readjust targets of already existing swarms.

Step 3.3 of Algorithm 2 highlights how each swarm is updated iteration by iteration to achieve its target. As already explained, first all particles are updates, then possible redundant particles are removed to avoid dominant solutions. After that, updated particles are filled and trimmed again to avoid diverging; finally, the best global particle of each swarm is updated.

Algorithm 6 explains how particles of a swarm are updated. It is also worth noting that because of the discrete nature of DAJS, the velocity of particles in PSO-ParFnt is calculated (Step 2) and used differently (Step 3) to update particles of a swarm. Here, based on the velocity for each particle, elements of a particle are either copied from the best particle of a swarm (Step 3.1) or randomly set to other values (Step 3.2). The lower the velocity of a particle, the greater chance it has for its elements to being filled with random values. Such a reverse mechanism to always force particles toward more movements allows generation of more variety of particles and consequently a greater chance of finding higher quality particles. Steps 3.3 and 3.4 make certain that the resulted particles are feasible. In Step 2, the distance between two particles, $\|P_1, P_2\|$, is calculated as

$$\|P_1, P_2\| = \frac{\sum_i \begin{Bmatrix} 1 & \text{if } P_1(i) = P_2(i) \\ 0 & \text{otherwise} \end{Bmatrix}}{|P_1|}, \tag{9}$$

where $|P_1|$ is the length of a particle; i.e., $N_J + N_D \times N_R$.

Input: (1) All particles in a swarm at iteration t, $\{P_1^t, \ldots, P_K^t\}$, their previous locations, $\{P_1^{t-1}, \ldots, P_K^{t-1}\}$, last velocity for each particle, $\{v_1^{t-1}, \ldots, v_K^{t-1}\}$, and the global best P_{best}^t
(2) Updating rates: w_1, w_2, w_3
Output: Next location of all particles $\{P_1^{t+1}, \ldots, P_K^{t+1}\}$

Step 1: Generate four random numbers
$0 < r_0, r_1, r_2, r_3 \leq 1$
Step 2: For $k = 1$ to K
$v_k^t = \frac{r_0 + r_1 w_1 v_k^{t-1} + r_2 w_2 \|P_k^t - P_k^{t-1}\| + r_3 w_3 \|P_k^t - P_{best}^t\|}{r_0 + r_1 w_1 + r_2 w_2 + r_3 w_3}$
Next k
Step 3: For $k = 1$ to K
For $i = 1$ to $|P_k^t|$
Generate three random number
$0 < r \leq 1$
$1 \leq r_{CC} \leq N_{CC}$
$0 \leq r_{SC} \leq N_{SC}$
3.1 If $(r < v_k^t)$ then
$P_k^{t+1}(i) = P_{best}^t(i)$
3.2 elseif $(x < N_J)$
$P_k^{t+1}(i) = r_{CC}$
elseif $(RndSC \neq 0)$
$P_k^{t+1}(i) = r_{SC}$
else
$P_k^{t+1}(i) = -1$
Endif
Next i
3.3 If P_k^{t+1} replicated a non-replicable files, then delete the replica
3.4 If P_k^{t+1}'s allocated files to SCs exceed their capacity; then, randomly delete replicas from P_k^{t+1} to make it feasible
Next k

Algorithm 6: Update particles in a swarm

5. Simulation results

Three artificial hybrid clouds are generated to check the performance of PSO-ParFnt using our exclusively designed/modified

Table 3
Characteristics of the generated test clouds.

	Test-Cloud-1	Test-Cloud-2	Test-Cloud-3
Number of Public/Private/Storage clouds	1/1/1	2/2/2	3/3/3
Number of CCs/SCs	2/3	4/6	6/9
Number of jobs	200	400	600
Execution time of all jobs	67 362 s (0 d:18 h:42 min: 42 s)	140 287 s (1 d:14 h:58 min:07 s)	209 035 s (2 d:10 h: 03 min:55 s)
Number of data-files	Total: 224; B[a]: 4, P: 20, N: 200	Total: 448; B: 8, P: 40, N: 400	Total: 672; B: 12, P: 60, N: 600
Size of data-files	Total: 47.5 TB; B: 23.0 TB, P: 13.1 TB, N: 11.4 TB	Total: 93.1 TB; B: 48 TB, P: 22.6 TB, N: 22.5 TB	Total: 135 TB; B: 70 TB, P: 31.4 TB, N: 33.7 TB
Public storage clouds	54 TB storage each	Each: 54 TB storage each	Each: 54 TB storage each
Public clouds	512 cores, 48 TB storage	Each: 512 cores, 48 TB storage	Each: 512 cores, 48 TB storage
Private clouds	128 cores, 30 TB storage	Each: 128 cores, 30 TB storage	Each: 128 cores, 30 TB storage

[a] B: Big-data, P: Private-Data, N: Normal-data.

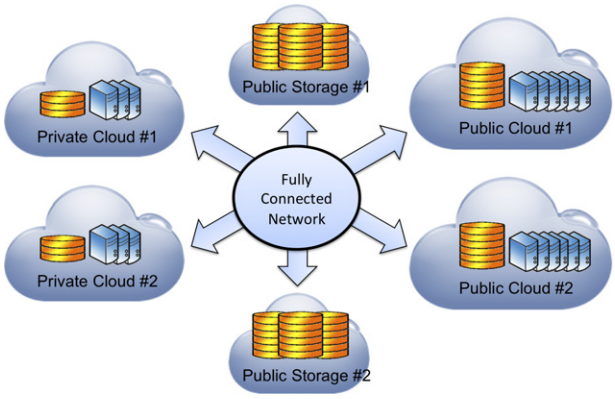


Fig. 5. Test-Cloud-2.

simulator also used in [12,13,15–17] and properly documented in [14]. These clouds are generated based on the direct observations from [32,33,35,40]. In this simulator, different clouds are generated through setting proper characteristics of their environmental parameters. Table 3 shows the characteristics of these systems. Fig. 5 shows the overall structure of the second test-cloud in our system. For example, Test-Cloud-1 presents a hybrid cloud environment consisting of one public computational cloud with 512 cores and 48 TB of storage, one private cloud with 128 cores and 30 TB of storage, and one public storage cloud with 54 TB of storage. 200 jobs with overall execution time of 18 h:42 min:42 s are created for this setup. These jobs required a total of 47.5 TB of data-files to be executed: 23 TB as big-data, 13.1 TB as private-data and 11.4 TB as normal-data.

It is also worth noting that these parameters are chosen based on the initial empirical studies we performed in our group during preparation of this work. We intentionally chose the parameters to observe the severe cases where different scheduling decisions for jobs or replication of data made significant differences in the overall performance of each system. For example, if the number of jobs were increased – with the same size and distribution of data – the system would be more skewed toward job scheduling and less caring about the location of data. The opposite observation would have also been noticed when the amount of data was doubled for the same number of jobs. In that case, the system was more skewed toward replication of data rather than scheduling of jobs. With this balance of settings however, both objectives of the stated problem would be equally important for the system where none of them could be either ignored or favored over the other. The configuration of the other two is similar but in different scales (details of these clouds for further analysis of this work and/or designing similar approaches can be obtained by contacting the authors).

5.1. Algorithm comparison

To the best of our knowledge, except for GA-ParFnt [15,16] which is designed for grids, such a Pareto front has never been computed for clouds with the detailed complications explained in previous sections. Thus, we could only compare our results with GA-ParFnt which uses a Genetic Algorithm to find the Pareto frontier of executing data dependent jobs versus transfer time of data-files in grid environments. Unlike PSO-ParFnt, GA-ParFnt does not consider privacy constraints of data-files as it assumes all data-files are replicable. Besides direct comparison of PSO-ParFnt with GA-ParFnt, we could also compare its results with other scheduling algorithms to (1) check the performance of PSO-ParFnt, and (2) verify the quality of answers found by the already designed algorithm for the DAJS problem. BestMap [17], DIANA [32,33], JDS-BC [13], Chameleon/FLOP [36], MinTrans [37,41–43], and MinExe [41,42] are six approaches that support the batch mode for scheduling, and thus are selected to also evaluate the performance of PSO-ParFnt.

In summary, BestMap [17] checks the detailed status of all CCs and SCs to decide where each job or data-file must be executed or replicated. DIANA [32,33] categorizes the submitted jobs as either computationally intensive or data intensive. For a computationally intensive job, DIANA migrates it to a CC that can provide the lowest execution time compared with other CCs; for a data intensive job, DIANA either migrates the job to a CC with the fastest data-file download time, or replicates the data-files to SCs with faster upload times compared with other SCs. Chameleon/FLOP [36] targets CCs that can start executing jobs straight away; it then replicates data-files upon scheduling jobs to provide the fastest upload times to them. MinTrans represent a collection of approaches that schedule jobs to CCs with already cached data-files, including JobDataPresent in [41], Data-Present in [42], TLSS + TLRS in [43], and an extension made to SAMGrid using Condor-G in [37]. These approaches are inspired by the fact that obtaining data-files is usually the costlier portion of executing a job, and thus if jobs are sent to CCs with already cached data-files, the overall performance of a system should improve. MinExe represents another group of approaches that schedule jobs to CCs that can execute them faster, including JobLeastLoaded in [41] and Shortest-Turnaround-Time in [42]. Such approaches are motivated by the fact that cache repositories of powerful CCs are gradually enriched as more types of jobs are scheduled on them, and thus it is the running portion of jobs that would eventually dominate the overall performance of a system. Achieving lower execution time and transfer time is the second priority in MinTrans and MinExe, respectively. All aforementioned scheduling algorithms and GA-ParFnt are carefully modified to not copy/replicate private data.

5.2. Test clouds

Three artificial test clouds are generated to represent various scenarios, ranging from single out-bursting hybrid cloud to three

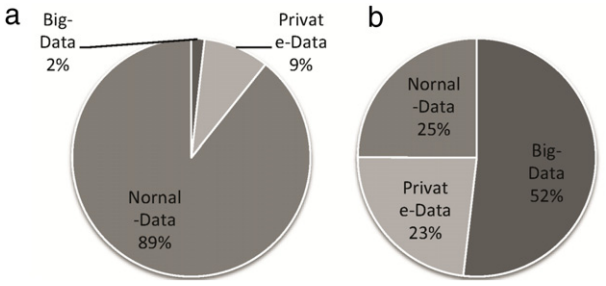


Fig. 6. Distribution of data files according to their (a) number and (b) size.

concurrent ones. Table 3 shows characteristics of these clouds as well as the overall characteristics of their jobs and big-/private-/normal-data-files.

For these test clouds, we assumed that cloud clients have private clouds with 128 cores and 30 terabyte (TB) of storage. They may also decide to out-burst their computation to (1) public clouds and rent 512 cores and 48 TB of storage, and (2) public cloud storage and rent 54 TB of capacity. We also assume that each submitted job to the cloud takes between 2 and 30 min to run and needs between 0 and 5 data-files. Data-files were comprised of three groups: big, private, and normal. Big and normal data-files can be stored on either public or private clouds, whereas private data-files are only stored on private storage (storage attached to a private cloud). We also assumed that size of big, private, and normal data is between 1–10 TB, 100–1000 MB, and 10–100 MB, respectively. During the generation of our hybrid clouds, we also assumed that the total size of all big-data-files in a system is almost half of the total size of all data-files; the other half is roughly split between private and normal data. Fig. 6 shows the distribution pattern of data-files according to their size and number for Test-Cloud-1; the distribution patterns for the other two larger cloud scenarios are very similar to that of Test-Cloud-1. For better simulation of our work we also assumed that a computing cloud is connected through LAN links to its local storage, and through WAN links to other clouds/storage.

6. Discussion and analysis

For Test-Clouds-1/2/3, Figs. 7–9 show the performance of the proposed algorithm (PSO-ParFnt) to find the Pareto frontier of executing a batch of jobs versus transferring time of their required data-files when replication of big-data is allowed or not-allowed. The results for our test clouds are further analyzed to extract vital information about (1) execution behavior of data-file dependent jobs in hybrid clouds as well as (2) execution performance of PSO-ParFnt compared with other techniques.

Table 4
Estimation functions for test clouds.

	Estimation function	MSE
Test-Cloud-1	$0.03 + 0.96e^{-63.56x}$	0.0141
	$0.07e^{-2.19x} + 0.93e^{-68.19x}$	0.0152
Test-Cloud-2	$0.03 + 0.85e^{-27.13x}$	0.0041
	$0.74e^{-17.49x} + 0.26e^{-12544.97x}$	0.0014
Test-Cloud-3	$0.03 + 0.81e^{-13.38x}$	0.0024
	$0.70e^{-8.78x} + 0.23e^{-16132.76x}$	0.0006

6.1. The shape of Pareto front for hybrid clouds

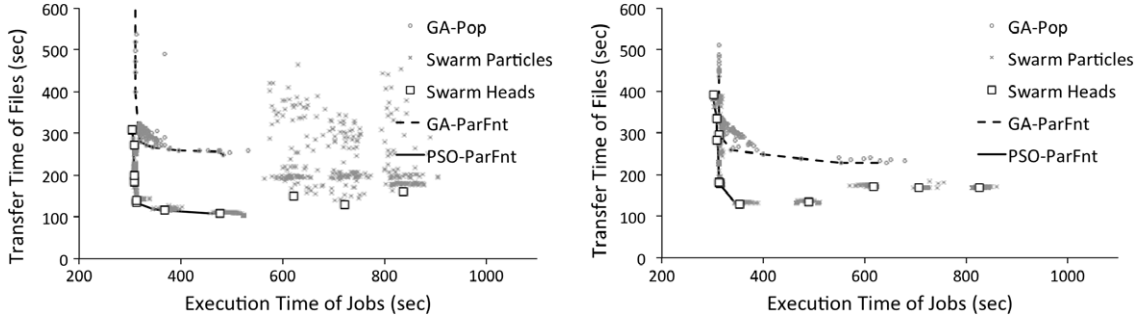
Figs. 7–9 show subtle differences among Pareto frontiers for these test clouds despite their very similar overall shape. For better comparative analysis, we first scale all points to be between [0, 1] and then fit two exponential functions to them: Func1Exp and Func2Exp with the overall shape of $\alpha + \beta \times e^{-\omega t}$ and $\alpha \times e^{-\omega_1 t} + \beta \times e^{-\omega_2 t}$, respectively. We empirically chose exponential functions as they showed lower mean square error (MSE) compared with other regression functions such as linear or logarithmic. Table 4 presents estimated functions for each test cloud along with its measured MSE. Fig. 10 graphically shows the estimated functions in Table 4 along with the original curve found by PSO-ParFnt for all test clouds.

Results in Fig. 10 and Table 4 show sharply curved Pareto fronts for Test-Cloud-1, consisting of one private plus one public cloud; this test cloud is very similar to the out-bursting case of an individual company. As more companies collaborate in making their hybrid clouds (Test-clouds-2/3) such Pareto frontier becomes smoother. As shown in Table 4, the dominant decay factor (Func1Exp) for Test-Cloud-1 is 63.56, almost double the decay factor for Test-Cloud-2 (27.13), and the decay factor of Test-Cloud-2 is also almost double that of Test-Cloud-3 (13.38). This observation shows that should more companies collaborate in sharing their computing infrastructure, such Pareto frontier will become smoother and consequently provide them more options to gradually switch from one priority such as execution time of jobs to another such as transfer time of data-files and vice versa. Note that such sharing does not violate the privacy issues of each company as private data are still kept in private storage elements of each individual.

Fig. 10 and Table 4 also show that the Pareto frontier curve of hybrid clouds are most probably smoother than those of grids previously studied in GA-ParFnt [15,16]; in [15,16] it has been shown that Fun1Exp type functions are generally not adequate for estimating the overall Pareto frontier shape of grids, whereas they are quite accurate for hybrid clouds.

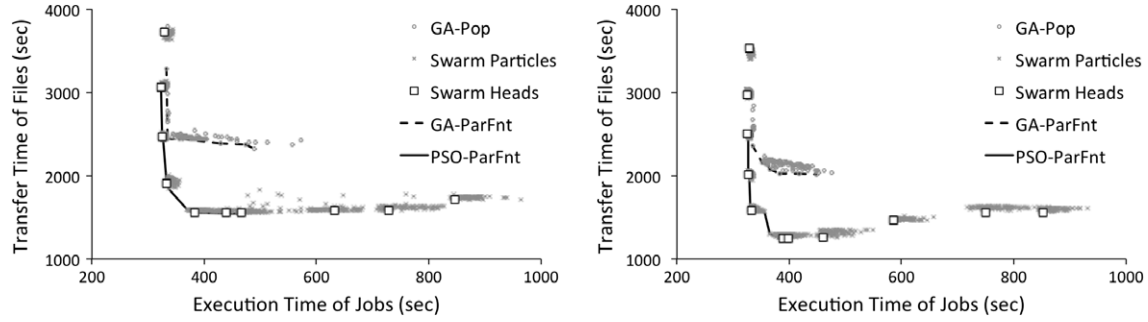
6.2. Comparing PSO-ParFnt with GA-ParFnt

Figs. 7–9 show results of GA-/PSO-ParFnt techniques, both designed to find the Pareto frontier of executing jobs versus transferring time of their required data-files. These figures clearly show



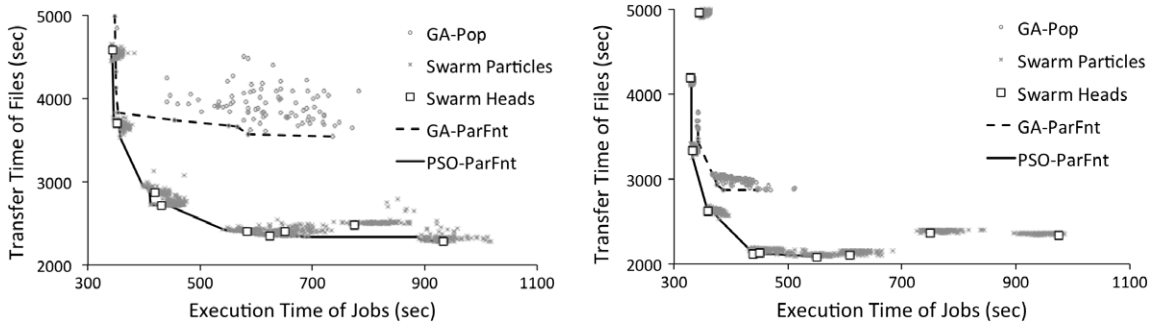
(a) Allowed. (b) Not allowed.

Fig. 7. Pareto frontier of Test-Cloud-1 when replication of Big-data is either allowed (a) or not allowed (b).



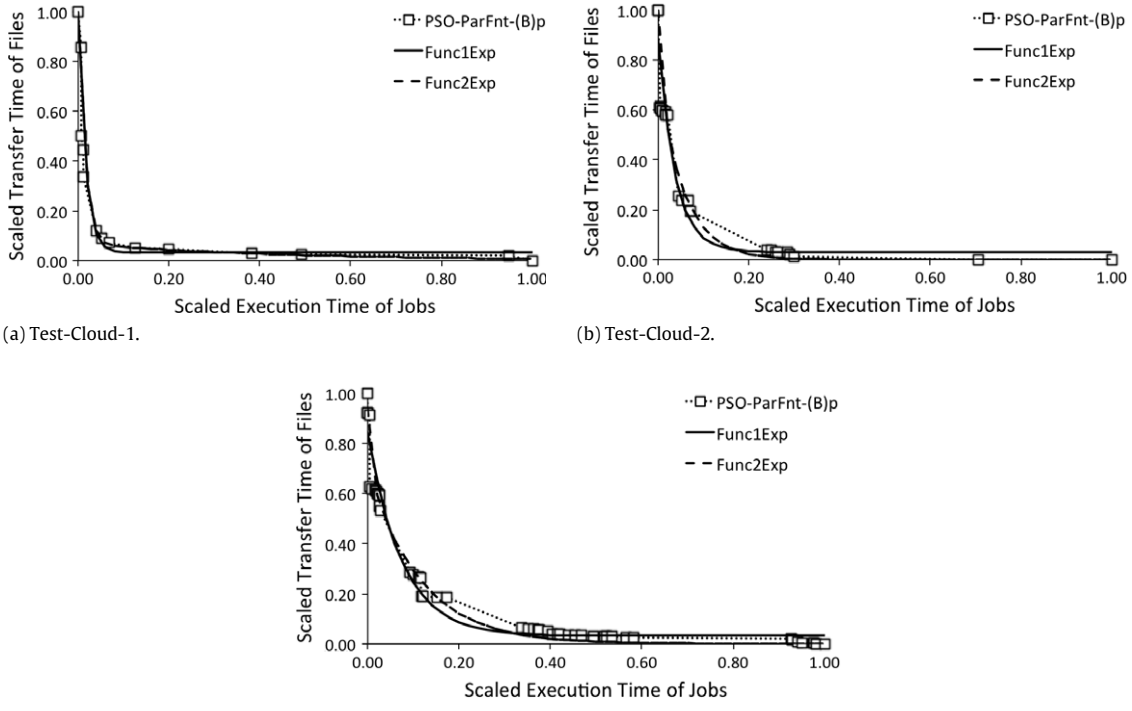
(a) Allowed. (b) Not allowed.

Fig. 8. Pareto frontier of Test-Cloud-2 when replication of Big-data is either allowed (a) or not allowed (b).



(a) Allowed. (b) Not allowed.

Fig. 9. Pareto frontier of Test-Cloud-3 when replication of Big-data is either allowed (a) or not allowed (b).



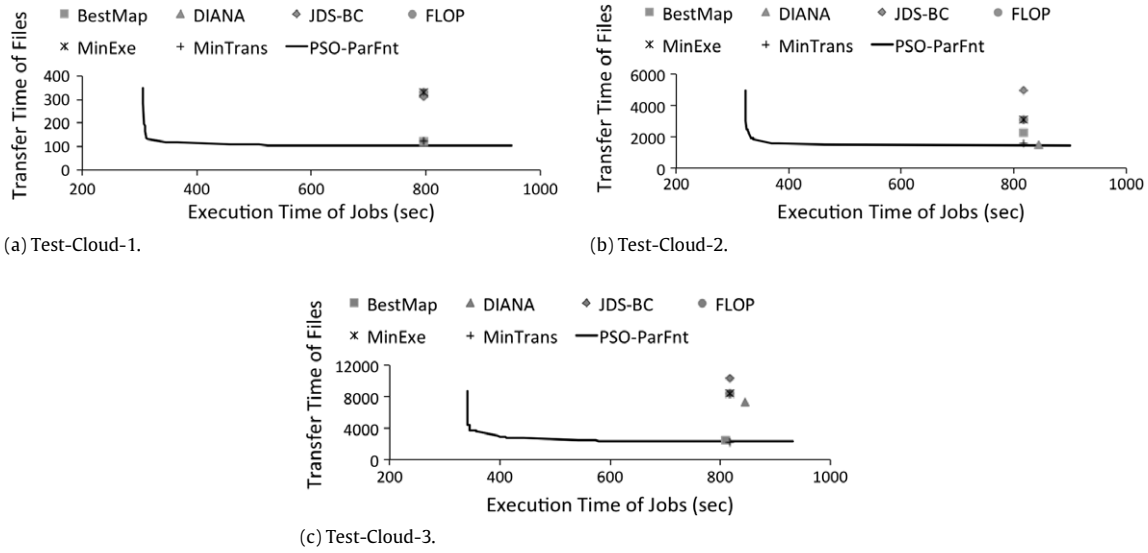
(a) Test-Cloud-1. (b) Test-Cloud-2.

(c) Test-Cloud-3.

Fig. 10. PSO-ParFnt-(B)p aligned with exponential estimation functions.

that both techniques have similar performance in finding the lowest possible execution time of jobs, whereas the lowest possible transfer time of data-file GA-ParFnt estimates is almost double of that of PSO-ParFnt.

To be more specific, Fig. 7(a) shows that the lowest possible execution time of jobs for all jobs in Test-Cloud-1 is almost 300 s – found by both techniques – whereas GA-ParFnt approximates 300 s as the lowest possible transfer time of data-files against PSO-ParFnt that approximates this value to be around 150 s. Such observation



(a) Test-Cloud-1.

(b) Test-Cloud-2.

(c) Test-Cloud-3.

Fig. 11. Pareto frontier of Test-Cloud-1/2/3 aligned with solutions found by scheduling algorithms.

can also be seen in other curves in Figs. 8–9, namely that PSO-ParFnt's estimation of the lowest possible transfer time of all data-files is almost half that of GA-ParFnt.

The other observation relates to the distribution of chromosomes in GA-ParFnt and swarm particles in PSO-ParFnt as the exploring agents in finding accurate Pareto frontiers for these test clouds. As can be seen in Figs. 7–9, GA-ParFnt's chromosomes are mostly populated over the "knee" section of each Pareto front, while swarm particles of PSO-ParFnt are almost homogeneously spread over all sections of these curves. This is the main reason why PSO-ParFnt was more effective in exploring all sections of such Pareto front and consequently produced superior results with almost half the data-file transfer time compared with GA-PatFnt.

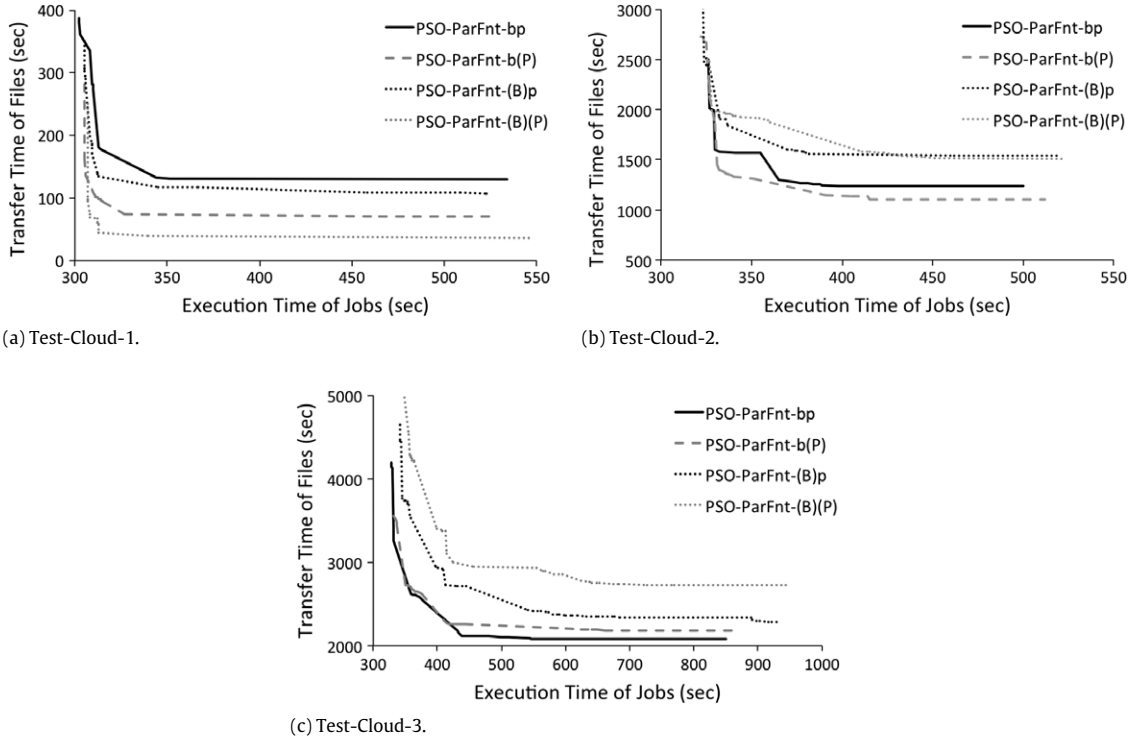
6.3. Comparing PSO-ParFnt with scheduling algorithms

Although PSO-ParFnt's overall aim is different than that of the aforementioned scheduling algorithms, we improvised such comparison to reveal two facts: (1) the accuracy of the Pareto frontier curves calculated by PSO-ParFnt, and (2) the efficiency of scheduling algorithms. Finding any solution by these scheduling algorithms as a single point in these plates below the calculated Pareto front conflicts with the concept of a Pareto frontier and thus devalues the PSO-ParFnt altogether. Furthermore, comparing the distance between each scheduling algorithm to the calculated Pareto front can also reveal the efficiency of each scheduling policy in finding the optimal solution for the stated DAJS problem. Fig. 11 shows such solutions, aligned with the Pareto frontier curve found by PSO-ParFnt for each test cloud.

Results in Fig. 11 reveal intriguing facts regarding the accuracy of these techniques for different hybrid clouds. For Test-Cloud-1 in Fig. 11(a), solutions can be split into two groups: (1) MinTrans and BestMap that produced optimal solutions very close to the Pareto front, and (2) the rest of the scheduling algorithms with identical execution time of jobs. The transfer time of the second group is almost triple that of the first group. For Test-Cloud-2 in Fig. 11(b), solutions of these scheduling algorithms are not forming any group, but distributing along the transfer time axis with almost identical execution time of jobs. In this case, MinTrans and DIANA produce optimal solutions, while BestMap slightly distances from the Pareto frontier curve. For this test cloud, JDS-BC produces the lowest quality solution with almost triple the transfer time of data-files. For Test-Cloud-3 in Fig. 11(c), MinTrans still produced an optimal solution; BestMap eliminates its distance with the Pareto front and also produced an optimal solution, DIANA noticeably distances from the Pareto front with almost double the transfer time of files, the rest produce answers with at least quadruple transfer time of files. JDS-BC still produced the worse solution!

Further investigation into how each of these scheduling algorithms produces its solution reveals invaluable facts for better design of scheduling algorithms for clouds. The most noteworthy observation in all cases was the almost identical execution time of jobs produced by all scheduling techniques regardless of the size of cloud: almost triple the lowest possible execution time of jobs found by PSO-ParFnt. This proves that single-element scheduling of jobs/data-files (as the common factor among all these algorithms) is probably not effective in hybrid clouds. We chose to refer to all these algorithms "single-element" scheduling as they all try to find the best computing element for executing a job or the best storage element for replicating a data-file one-at-a-time during their scheduling procedure. It is worth noting that this procedure differs from dynamic scheduling where a made decision is irreversible. These algorithms in fact, run a series of decision making procedures where, in each step, the scheduling decision of their previous step could be altered; however, they make such decisions for a single element (job or data-file) at a time. Their transfer time of data-files also greatly differs from one algorithm to another as well as across different clouds.

BestMap always tries to find the best location of each job/data-file one at a time. This procedure seems also to work for clouds as it usually resulted in very close proximity to the Pareto fronts. DIANA categorizes jobs/data-files; this policy seems to be ineffective where a large number of jobs/data-files exist in a system. DIANA shows very similar inefficient results when deployed for large grids [12,13,15–17]. JDS-BC schedules jobs/data-files based on their similarities with the ones already scheduled/replicated. Although this policy showed acceptable results for grid environments where more variety of computing/storage elements is generally available, it proved to be ineffective for clouds. The main reason for this observation is most probably behind the different topological nature of grids and clouds: grids generally consist of more grid nodes with smaller number of computing elements in each node versus clouds that generally consists of less number of nodes with much higher computing elements in each node. For example, 1000 computing cores in a grid could be provided by 20 grid nodes, each including 50 cores, whereas in clouds, it



(a) Test-Cloud-1.

(b) Test-Cloud-2.

(c) Test-Cloud-3.

Fig. 12. Pareto frontier of Test-Cloud-1/2/3 for relaxed replication policies.

could be provided by two clouds with 500 cores each. As can be rationalized, in grids, the similarity between a job/data-file and the ones already scheduled/replicated could provide crucial information about many possible grid nodes and their possible performance for the job/data-file in hand; whereas in clouds with only very few options, the deduced information as a result of these comparisons could be less informative as also shown in Fig. 11. FLOP always tries to find the first computing element that can start execution of a job. This policy also seems inefficient for cloud environment where availability of cores seems less important as compared with availability of the required data-files. MinExe also showed very similar results to FLOP as they both aim to execute jobs as fast as possible. Although MinExe and FLOP produced different solutions for grids, they seem to produce identical results for clouds. MinTrans always schedules jobs and replicates data-files to achieve the lowest possible transfer time of files; this procedure also seems to be very effective in clouds where the main bottleneck seems to be the data transfer rather than the computation power.

6.4. Effect of data nature on the Pareto front

In this section we further study the relationship between the nature of data and the Pareto frontier of a cloud. To this end, we first relaxed the privacy policy for private-data in clouds and ran four different replication policies for each cloud. Fig. 12 shows the resulting Pareto frontier curves for these policies. In this figure, the XY section of a PSO-ParFnt-XY represents its replication policy: capital letters in parentheses mean the replication was allowed; small letters mean the opposite. Table 5 explains all legends. Among these four policies, only PSO-ParFnt-bp/(B)p policies are feasible and hence are shown in darker colors in this figure; the lighter color lines are only to investigate hypothetical cases where private-data are also allowed to be replicated.

Fig. 12 shows that the size of the hybrid cloud has a great impact on the shape and relative location of Pareto frontiers for these four replication policies against each other. For Test-Cloud-1, as expected, PSO-ParFnt-(B)(P) where both big and private data are allowed to have replicas in the system, transfer time of all data-files could be as low as 50 s. For PSO-ParFnt-b(P)/(B)p where only one of these data-files classes is allowed to have replicas, allowing replication of private data reduces the overall transfer time of data-files almost twice that of only allowing replication of big-data. It is also worth noting that the amount of data transfer time that can be saved by allowing each of these two classes to replicate is almost linear!; i.e., the lowest possible transfer time of data-file for PSO-ParFnt-b(P), PSO-ParFnt-(B)p, and PSO-ParFnt-(B)(P) is almost 30 s, 60 s, and 85 s ($\approx$30 s + 60 s) lower than that of PSO-ParFnt-bp, respectively.

For larger hybrid clouds that consist of two or more individual hybrid clouds, a different phenomenon is observed. For Test-Cloud-2, PSO-ParFnt-bp/b(P) achieved the lowest possible transfer time; this shows that despite the general impression that copying/replicating big-data could help reduce the overall transfer time of all data-files in a system, it might also worsen it! Here for example, through such replication, some precious capacity of storage nodes is probably inefficiently occupied and eventually resulted in downloading more data-files in the system. For Test-Cloud-2, Fig. 12(b) also shows that PSO-ParFnt-(B)p/(B)(P) are also fairly close to each other, implying that replicating private-data – even if allowed – has not much effect on the overall transfer time of the system when big-data is allowed to be replicated: a policy that most probably results in unproductive occupation of many valuable storage capacities. For Test-Cloud-3, these differences could be more clearly observed. Here, for example, no replication policy (PSO-ParFnt-bp) for both data classes (big and private) results in the lowest possible transfer time of files. It is very intriguing to see that adding replication policies would actually increase – instead of decrease – the overall transfer time of data-files; i.e., about 100 s and 200 s for PSO-ParFnt-b(P) and PSO-ParFnt-(B)p, respectively. It is also noteworthy to see that allowing both policies at the same time (PSO-ParFnt-(B)(P)) increases the lowest possible transfer time of data-files to almost 700 s and not (100 s + 200 s)!

Table 5
Legends for Fig. 12.

Legend	Big-data replication policy	Private-data replication policy
PSO-ParFnt-bp	Not-allowed	Not-allowed
PSO-ParFnt-b(P)	Not-allowed	Allowed
PSO-ParFnt-(B)p	Allowed	Not-allowed
PSO-ParFnt-(B)(P)	Allowed	Allowed

Table 6
PSO-ParFnt's execution time.

Cloud	Time
Test-Cloud-1	0 h:32 min:25 s
Test-Cloud-2	1 h:15 min:45 s
Test-Cloud-3	2 h:48 min:12 s

Based on further analysis of the results in Fig. 12, we can also hypothesize that allowing replication of Big-data could only be effective in lowering the transfer time of all data-files when simple hybrid clouds are implied. For larger/more complicated hybrid clouds where more than one entity share resources with each other as well as take advantage of public cloud facilities, replicating big-data has an inverse effect for better scheduling of jobs/data-files. On that note, we can also conclude that although replication of private data – even if allowed – can reduce the overall transfer time of systems, it could also slow it down for cooperative hybrid clouds sometimes.

6.5. Execution time of PSO-ParFnt

PSO-ParFnt was developed as an extension to our already designed simulator documented in [14]. For these test clouds, PSO-ParFnt was run on a typical dual-core i7 desktop PC with 16 GB of RAM. Table 6 shows the convergence time for each test cloud. It is worth noting that the SchMng program we used to perform our evaluation is written to only use one core of a PC – even if it has many to offer. As a result, SchMng could not take full advantage of the i7 technology and/or the large RAM provided to it. This table also shows that our designed algorithm is able to find the Pareto frontier curve even for our largest test cloud with 600 jobs, 672 data-files, and a hybrid system with 1920 cores in less than 3 h.

7. Conclusion and future work

This paper introduced a particle swarm optimization technique (PSO-ParFnt) to explore the Pareto frontier curve of the execution time of a batch of jobs versus the transfer time of their required data-files in hybrid clouds. To this end, we first modified the generic PSO technique to have collections of targeted swarms instead of a generic large swarm as usually used in PSO-based approaches. Three test clouds were designed to gauge the efficiency of PSO-ParFnt through comparing it with other techniques. Results were intriguing and revealed invaluable insights into the complex DAJS problem for hybrid clouds. In this study, we particularly found that, despite general impressions that replication of data-files always reduces the overall transfer time of files in a system, it sometimes worsens it; mainly because it may waste precious capacity of storages nodes sometimes. We also observed that the scheduling dynamic of individual hybrid clouds differs from those consisting of several hybrid clouds shared by different entities.

Our future work will focus on the following two directions. Firstly, we wish to modify the developed simulator (SchMng) to leverage parallelization techniques and converge to its solutions faster. Secondly, using direct observation of this study, we will design exclusive cloud schedulers for hybrid clouds and experimentally test them using the private cloud at our university.

References

[1] D. Agrawal, S. Das, A.E. Abbadi, Big data and cloud computing: current state and future opportunities, in: Proceedings of the 14th International Conference on Extending Database Technology, Uppsala, Sweden, 2011, pp. 530–533.
[2] Microsoft. (Visited 2013). http://www.windowsazure.com/.
[3] Amazon. (Visited 2013). www.aws.amazon.com/ec2/.
[4] myNoSQL. (Visited 2013). http://nosql.mypopescu.com/post/6361838342/bigdata-volume-velocity-variability-variety.
[5] A. Delgado Peris, J. Hernandez, E. Huedo, I.M. Llorente, Data location-aware job scheduling in the grid. Application to the GridWay metascheduler, J. Phys. Conf. Ser. 219 (2010) 062043.
[6] J. Kołodziej, S. Khan, Data scheduling in data grids and data centers: a short taxonomy of problems and intelligent resolution techniques, in: N.-T. Nguyen, J. Kołodziej, T. Burczyński, M. Biba (Eds.), Transactions on Computational Collective Intelligence X. vol. 7776, Springer, Berlin, Heidelberg, 2013, pp. 103–119.
[7] C. Augonnet, J. Clet-Ortega, S. Thibault, R. Namyst, Data-aware task scheduling on multi-accelerator based platforms, in: 2010 IEEE 16th International Conference on Parallel and Distributed Systems, ICPADS, 2010, pp. 291–298.
[8] G. Lee, B.-G. Chun, H. Katz, Heterogeneity-aware resource allocation and scheduling in the cloud, in: Presented at the Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing, Portland, OR, 2011.
[9] T. Kosar, Data-aware distributed batch scheduling, in: Handbook of Research on Grid Technologies and Utility Computing: Concepts for Managing Large-Scale Applications, IGI Global, 2009, pp. 41–48.
[10] K. Hasham, A.D. Peris, A. Anjum, D. Evans, S. Gowdy, J.M. Hernandez, E. Huedo, D. Hufnagel, F. Van Lingen, R. McClatchey, S. Metson, CMS workflow execution using intelligent job scheduling and data access strategies, IEEE Trans. Nucl. Sci. 58 (2011) 1221–1232.
[11] J. Kolodziej, M. Szmajduch, S.U. Khan, L. Wang, D. Chen, Genetic-based solutions for independent batch scheduling in data grids, in: European Council for Modeling and Simulation, ECMS 2013, 2013.
[12] J. Taheri, A.Y. Zomaya, P. Bouvry, S.U. Khan, Hopfield neural network for simultaneous job scheduling and data replication in grids, Future Gener. Comput. Syst. 29 (2013) 1885–1900.
[13] J. Taheri, Y. Choon Lee, A.Y. Zomaya, H.J. Siegel, A bee colony based optimization approach for simultaneous job scheduling and data replication in grid environments, Comput. Oper. Res. 40 (2013) 1564–1578.
[14] J. Taheri, A. Zomaya, S.U. Khan, Grid simulation tools for job scheduling and datafile replication, in: Samee U. Khan, A.Y. Zomaya, L. Wang (Eds.), Scalable Computing and Communications: Theory and Practice, John Wiley & Sons, Inc., Hoboken, New Jersey, 2013.
[15] J. Taheri, A.Y. Zomaya, A Pareto frontier for optimizing data transfer and job execution in grids, in: Presented at the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & Ph.D. Forum, IPDPSW, 2012.
[16] J. Taheri, A.Y. Zomaya, S.U. Khan, Genetic algorithm in finding Pareto frontier of optimizing data transfer versus job execution in grids, in: Concurrency and Computation: Practice and Experience, 2012, pp. n/a–n/a.
[17] J. Taheri, Y.C. Lee, A.Y. Zomaya, Simultaneous job and data allocation in grid environments, The University of Sydney, Sydney, Australia, TR 6712011.
[18] Montage. (visited 2013). http://montage.ipac.caltech.edu/.
[19] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B.P. Berman, P. Maechling, Scientific workflow applications on Amazon EC2, in: 2009 5th IEEE International Conference on E-Science Workshops, 2009, pp. 59–66.
[20] V. Nefedova, R. Jacob, I. Foster, Z. Liu, Y. Liu, E. Deelman, G. Mehta, M.-H. Su, K. Vahi, Automating climate science: large ensemble simulations on the teragrid with the GriPhyN virtual data system, in: Presented at the Proceedings of the Second IEEE International Conference on E-Science and Grid Computing, 2006.
[21] MIKE2.0. (Visited 2013). Big Data Definition (http://mike2.openmethodology.org/wiki/Big_Data_Definition).
[22] T. White, Hadoop: The Definitive Guide, Original edition, O'Reilly Media, 2009.
[23] ZDNET. (Visited 2013). http://www.zdnet.com/blog/virtualization/what-is-big-data/1708.
[24] The-Economist. (Visited 2013). http://www.economist.com/node/15557443?story_id=15557443.
[25] T. Segaran, J. Hammerbacher, Beautiful Data, O'Reilly Media, Inc., 2009.
[26] IBM. (Visited 2013). What is big data? http://www-01.ibm.com/software/data/bigdata/.
[27] GREENPLUM. (Visited 2013). http://www.greenplum.com/.
[28] Wikibon. (Visited 2013). http://wikibon.org/wiki/v/Enterprise_Big-data.
[29] FUJITSU, Personal data on the cloud: a global survey of customer attitudes (http://www.fujitsu.com/global/news/publications/dataprivacy.html), Visited 2013.
[30] W.K. Hon, C. Millard, I. Walden, The problem of 'personal data' in cloud computing: what information is regulated?—the cloud of unknowing, Int. Data Priv. Law 1 (2011) 211–228.
[31] R.-S. Chang, J.-S. Chang, S.-Y. Lin, Job scheduling and data replication on data grids, Future Gener. Comput. Syst. 23 (2007) 846–860.
[32] A. Anjum, R. McClatchey, A. Ali, I. Willers, Bulk scheduling with the DIANA scheduler, IEEE Trans. Nucl. Sci. 53 (2006) 3818–3829.
[33] R. McClatchey, A. Anjum, H. Stockinger, A. Ali, I. Willers, M. Thomas, Data Intensive and Network Aware (DIANA) grid scheduling, J. Grid Comput. 5 (2007) 43–64.
[34] GILDA. (Visited 2013). https://gilda.ct.infn.it/.

[35] CERN. (Visited 2013). Compact Muon Solenoid (CMS) http://public.web.cern.ch/public/en/lhc/CMS-en.html.
[36] P. Sang-Min, K. Jair-Hoom, Chameleon: a resource scheduler in a data grid environment, in: Proceedings of 3rd IEEE International Symposium on Cluster Computing and the Grid, CCGRID'03, 2003, pp. 258–265.
[37] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, K. Stockinger, Data management in an international data grid project, in: R. Buyya, M. Baker (Eds.), Grid Computing, vol. 1971, Springer-Verlag, New York, 2000, pp. 77–90.
[38] S. Hongzhang, L. Oliker, R. Biswas, Job superscheduler architecture and performance in computational grid environments, in: Proceedings of the ACM/IEEE SC2003 Conference, SC'03, 2003, pp. 44–58.
[39] J. Blondin, Particle swarm optimization: a tutorial, (www.cs.armstrong.edu/saad/csci8100/pso_tutorial.pdf), 2009.
[40] X.C. Liu, X.G. Qiu, B. Chen, Q. He, K.D. Huang, Scheduling parallel discrete event simulation jobs in the cloud, in: IET Conference Publications, vol. 2012, 2012, pp. 72–72.
[41] K. Ranganathan, I. Foster, Decoupling computation and data scheduling in distributed data-intensive applications, in: Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing, HPDC'02, 2002, pp. 352–358.
[42] M. Tang, B.-S. Lee, X. Tang, C.-K. Yeo, The impact of data replication on job scheduling performance in the data grid, Future Gener. Comput. Syst. 22 (2006) 254–268.
[43] S. Abdi, S. Mohamadi, Two level job scheduling and data replication in data grid, Int. J. Grid Comput. Appl. 1 (2010) 23–37.

Javid Taheri received his Bachelor and Masters of Electrical Engineering from Sharif University of Technology, Tehran, Iran in 1998 and 2000, respectively. He received his Ph.D. in the field of Mobile Computing from the School of Information Technologies (SIT) in the University of Sydney, Australia. Since 2006, he has been actively working in several fields, including networking, bioinformatics, and parallel computing. He is currently working as a Postdoctoral research fellow at SIT/USyd in designing scheduling algorithms for cloud and green computing.

Albert Y. Zomaya is currently the *Chair Professor of High Performance Computing & Networking* in the School of Information Technologies, The University of Sydney. He is also the *Director of the Centre for Distributed and High Performance Computing* which was established in late 2009. Professor Zomaya is the author/co-author of seven books, more than 450 papers, and the editor of 14 books and 20 conference proceedings. He is the Editor in Chief of the *IEEE Transactions on Computers* and serves as an associate editor for 20 leading journals, such as, the *ACM Computing Surveys, IEEE Transactions on Cloud Computing*, and *Journal of Parallel and Distributed Computing*. Professor Zomaya served as General and Program Chair for more than 60 events and served on the committees of more than 600 ACM and IEEE conferences. He delivered more than 130 keynote addresses, invited seminars and media briefings. Professor Zomaya is the recipient of the *IEEE Technical Committee on Parallel Processing Outstanding Service Award* and the *IEEE Technical Committee on Scalable Computing Medal for Excellence in Scalable Computing*, both in 2011. He is a Chartered Engineer, a Fellow of AAAS, IEEE, IET (UK). Professor Zomaya's research interests are in the areas of parallel and distributed computing and complex systems.

Howard Jay ("H.J.") Siegel has been the Abell Endowed Chair Distinguished Professor of Electrical and Computer Engineering at Colorado State University (CSU) since 2001, where he is also a Professor of Computer Science. From 2002 to 2013, he was the first Director of the CSU Information Science and Technology Center (ISTeC), a university-wide organization for promoting, facilitating, and enhancing CSU's research, education, and outreach activities pertaining to the design and innovative application of computer, communication, and information systems. From 1976 to 2001, he was a professor at Purdue University. Prof. Siegel is a Fellow of the IEEE and a Fellow of the ACM. He received a B.S. degree in electrical engineering and a B.S. degree in management from the Massachusetts Institute of Technology (MIT), and the M.A., M.S.E., and Ph.D. degrees from the Department of Electrical Engineering and Computer Science at Princeton University. He has co-authored over 400 technical papers. His research interests include robust computing systems, resource management in computing systems, energy-aware computing, heterogeneous parallel and distributed computing and communications, parallel algorithms, and parallel machine interconnection networks. He was a Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing*, and was on the Editorial Boards of both the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He has been an international keynote speaker and tutorial lecturer, and has consulted for industry and government. For more information, please see www.engr.colostate.edu/~hj.

Zahir Tari is a full professor in Distributed Systems at RMIT University (Australia). He received a bachelor degree in Mathematics from University of Algiers (USTHB, Algeria) in 1984, MSc in Operational Research from University of Grenoble (France) in 1985 and Ph.D. degree in Computer Science from University of Grenoble (France) in 1989. Tari's expertise is in the areas of system performance (e.g. Web servers, P2P, Cloud) and system security (e.g. SCADA security). He is the co-author of six books (John Wiley, Springer) and he has edited over twenty five conference proceedings.He has been Program Committee chair of several international conferences. Prof Tari is also a recipient of over 5M$in funding from theARC and various Australian software industries.Prof Tari is an associate editor of the IEEE Transactions on Computers (TC) and IEEE Transactions on Parallel and Distributed Systems (TPDS).