J Grid Computing (2014) 14:245–264
DOI 10.1007/s10723-013-9282-3

Science in the Cloud: Allocation and Execution of Data-Intensive Scientific Workflows

Claudia Szabo · Quan Z. Sheng · Trent Kroeger · Yihong Zhang · Jian Yu

Received: 9 November 2012 / Accepted: 10 October 2013 / Published online: 30 October 2013

Abstract An important challenge for the adoption of cloud computing in the scientific community remains the efficient allocation and execution of data-intensive scientific workflows to reduce execution time and the size of transferred data. The transferred data overhead is becoming significant with emerging scientific workflows that have input/output files and intermediate data products ranging in the hundreds of gigabytes. The allocation of scientific workflows on public clouds can be described through a variety of perspectives and parameters, and has been proved to be NP-complete. This paper proposes an evolutionary approach for task allocation on public clouds considering data transfer and execution time. In our framework, a solution is represented using an *allocation* chromosome that encodes the allocation of tasks to nodes, and an *ordering* chromosome that defines the execution order according to the scientific workflow representation. We propose a multi-objective optimization that relies on a cloud cost model and employs tailored evolution operators. Starting from a population of possible solutions, we employ crossover and mutation operators on both chromosomes aiming at optimizing the data transferred between nodes as well as the total workflow runtime. The crossover operators combine parts of solutions to reduce data overhead, whereas the mutation operators swamp between parts of the same chromosome according to pre-defined rules. Our experimental study compares between the proposed approach and current state-of-the art approaches using synthetic and real-life workflows. Our algorithm performs similarly to existing heuristics for small workflows and shows up to 80 % improvements for larger synthetic workflows. To further validate our approach we compare between the allocation and scheduling obtained by our approach with that obtained by popular scientific workflow managers, when real workflows with hundreds of tasks are executed on a public cloud. The results show a 10 % improvement in runtime over existing schedulers, caused by a 80 % reduction in transferred data and optimized allocation and ordering of tasks. This improved data locality has greater

C. Szabo (✉) · Q. Z. Sheng · T. Kroeger · Y. Zhang
School of Computer Science, The University of Adelaide, Adelaide, SA 5005, Australia
e-mail: claudia.szabo@adelaide.edu.au

Q. Z. Sheng
e-mail: michael.sheng@adelaide.edu.au

T. Kroeger
e-mail: trent.kroeger@adelaide.edu.au

Y. Zhang
e-mail: yihong.zhang@adelaide.edu.au

J. Yu
Faculty of Information and Communication Technologies, Swinburne University of Technology, Melbourne, Victoria 3122, Australia
e-mail: jianyu@swin.edu.au

impact as it can be employed to improve and study data provenance and facilitate data persistence for scientific workflows.

Keywords Data-intensive workflows · Cloud computing · Scheduling · Allocation · Evolutionary computation

1 Introduction

Despite its evident appeal for affordable, easy-to-use computation, largely brought on by advantages such as pay-per-use, elasticity, and availability among others [19], cloud computing has yet to become the computing platform of choice, especially in the scientific community [10, 19, 33], where there is a need for a computational platform to replace High Performance Computing (HPC) for on-demand, responsive, and highly customized computational scientific research [10]. The suitability of cloud computing for scientific computation has been the focus of research in the past years [10, 33], with recent work proposing solutions that schedule and execute scientific workflows in the cloud [5, 32, 35]. This is an important challenge as today's problems are increasingly data and computationally intensive, which results in high computational and data-intensive workflows. However, despite recent efforts, cloud computing is still not widely accepted as a scientific computational platform. Besides initial perception problems that see cloud computing as difficult to access, low-interaction, low-responsiveness HPC [10], issues still arise due to concerns of availability [33], scheduling, and allocation of scientific workflows [5] among others.

The task allocation problem is an NP-complete problem [11] and as such various heuristics have been proposed [5, 23, 24, 35]. Traditional task allocation algorithms look at the problem from a provider perspective, in which the focus is to maximize the utilization of specific *homogeneous* resources in a Grid or cluster, when multiple workflows from different users are submitted [1, 17]. In contrast, in cloud computing, virtual clusters can be easily set up using *heterogeneous* resources [2], and the perspective is shifted to the user that is concerned with reduced runtime at minimal costs [19]. The inherent variety of cloud providers and solutions leads to a different formulation of the task allocation problem, as different types of resources (hardware, software), and utility functions (single or multi-objective, performance, cost, profit, energy consumption) can be devised [19]. A key issue remains providing locality of intermediate data products with the workflow tasks that need them [5]. For example, Montage, an astronomic image mosaic scientific workflow [15], spends 99 % of its time in data operations, sometimes exchanging files up to hundreds of gigabytes in size. A study by Shibata et al. [25] shows that locality-aware scheduling techniques which assign a task to the execution site that has most of its input files, can increase the locality of files by 96 %, depending on the structure of the workflow, and thus improve total execution time. Hence, data placement and task assignment are not independent problems and should be treated together. This is becoming of crucial importance as the size of the exchanged data is increasing dramatically. For example, reported real-life use of the Montage workflow has a workflow of 10,422 tasks that exchange a total of 57 GB of data, whereas the Cybershake workflow has 80 partitions, with each running a total of 24,132 tasks exchanging 58 GB of data [30].

In our previous work, we have proposed an evolutionary approach for the allocation of scientific workflows that optimizes the allocation based on the cost (in US$) of running the workflow [27]. Based on a cost model for running workflows on a set of cloud instances, our multi-objective algorithm employs a two-chromosome representation of the problem, encoding the *allocation* of tasks to particular cloud instances, and the *ordering* of task execution. Our multi-objective algorithm exploits a cost-execution runtime trade-off and achieves significant improvements over simple heuristics and particle swarm implementations using synthetic, generated, scientific workflow descriptions. Recognizing the importance of *data locality* when executing data-intensive scientific workflow, we propose in this paper a framework that considers and optimizes the allocation and ordering of tasks in the workflow such that the data transferred

between tasks and the execution runtime are minimized together. A trivial solution that would reduce the size of data transferred between tasks could be to allocate all tasks on a single cloud instance. This would reduce the overhead incurred by the communication between tasks running on different cloud instances, but the total runtime of the workflows will be maximal. Instead, we propose a multi-objective strategy that optimizes both data transfers and total runtime. In contrast to existing work that only looks at batch allocation of task bundles to specific cloud instances [17], our approach considers both the allocation and the ordering of tasks.

We employ a model based on the Amazon Elastic Compute Cloud (EC2) and Scalable Storage Service (S3), but our framework can be easily extended to other cloud providers. We analyze several scientific workflows of different sizes and characteristics. Firstly, our experiments allocate I/O—intensive workflows such as Montage [4], which creates science-grade astronomic image mosaics using data collected from telescopes. We next analyze computationally-intensive workflows such as Epigenomics [4], which maps short DNA segments to previous existing references. Secondly, we analyze scientific workflows of varying size, in the range of tens, hundreds, and thousands of tasks, and compare our approach with simple heuristics and particle swarm implementations. To further validate our approach, we integrate our solution with the Pegasus workflow management system that executes real-life scientific workflows on cloud instances from various providers [8]. We compare between the total data transferred and runtime obtained by running the workflow using the Pegasus default allocation and scheduling policy and the overhead and runtime obtained using our proposed multi-objective algorithm. To further exploit data locality, we implement a scheduling and allocation routine that shows significant performance gains in terms of runtime and reduced data transfer overhead than default policies. The main contributions of our work are:

- A novel representation of the problem using two chromosomes encoding the *allocation* and the *ordering* of tasks to cloud instances, and the definition and application of two-phase mutation and crossover operators following problem constraints,
- An evolutionary approach for the allocation of data-intensive scientific workflows that optimizes both workflow runtime and the size of transferred data,
- An extensive experimental study that compares our approach with existing approaches using synthetic and real workflows, and
- An integrated prototype that allocates and executes scientific workflows on public clouds using various scheduling strategies with significant runtime and data transferred overhead improvements when exploiting data locality.

This remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the task allocation problem in detail, including its formulation as an optimization problem with specific operators. A detailed experimental study follows in Section 4. Section 5 concludes this paper and discusses future work directions.

2 Related Work

Deelman et al. [8] have proposed the Pegasus workflow management system (WMS) as a framework that maps complex scientific workflows into Grid resources, using the Condor scheduler [28]. Similar to most workflow management systems [18, 20], Pegasus schedules tasks based on their earliest start or finish time, and/or resource utilization, and does not consider other parameters such as placement of data and resource cost. A recent paper investigates the suitability of cloud computing for the execution of Pegasus workflows, but only a single cloud instance (in which the Condor scheduler is installed) is considered [29]. Other approaches include matrix-based heuristics [35], hypergraph-based heuristics [5], particle swarm optimizations [21, 32], and optimizations based on evolutionary approaches [17].

In matrix-based heuristics, a similarity matrix is built for files exchanged in the workflow [35]. In this matrix, entry $d(ij)$ shows the number of

tasks that require both files $f(i)$ and $f(j)$. In the build-time stage, the matrix is constructed for all application data, including those that have fixed locations. Next, the similarity matrix is used to cluster the files such that highly related files are placed in the same site. When a task arrives during the workflow execution, it is greedily assigned to the execution site that contains most of its files. Similarly, when a data file is generated, it is placed in a site that contains files with a high similarity score with the input file. The experiments simulate a cloud environment in a local cluster using VMware, and generates random workflows to run on the simulated environment. The experiment results show that there is a 40 % reduction of data movement. However, while a greedy approach offers an increase in speed, the overhead resulted from the similarity calculation is significant, and similarity scores between data files are difficult to calculate.

Hypergraph-based heuristics [5] model the workflow as a hypergraph considering both data placement and task assignment. The hypergraph is partitioned using a formula that attempts to distribute file transfer workloads among execution sites with respect to some pre-determined ratios. This partitioning can be done with the aid of a modified version of an existing hypergraph partitioning tools, PaToH. The approach permits simultaneous construction of data placement and task assignment schemes that distribute the storage and computational loads among the execution sites with respect to some pre-determined ratios. This static allocation approach assumes that storage and execution capabilities of each cloud instance, as well as the exact number of cloud instances, is provided by the user, together with information about the desirability of particular cloud instances. The approach achieves a 35 % improvement over the matrix-based heuristic in terms of load balancing for synthetic workflows. However, this approach assumes a cost model based only on the data transferred, defined as the number of files exchanged between sites in a static workflow, and does not consider the size of the data files nor the task and workflow runtime.

From a provider perspective, evolutionary algorithms have been used to allocate tasks on cluster nodes to reduce compute node utilization [17]. This approach proposes a single chromosome representation of the problem, in which only the task allocation to specific nodes is defined, excluding the ordering of tasks on each node. This has a significant effect for scientific workflows where there exist order and input/output dependencies between tasks and as such invalid allocations can be generated when a workflow is allocated. Moreover, the approach does not consider other perspectives such as the runtime and cost of the execution, as well as the size of the data files transferred between nodes. Recent work from the creators of the Pegasus workflow management system looks at employing genetic algorithms to optimize the execution costs of scientific workflows while at the same time ensuring that the constructed virtual cluster is optimal [30]. This provider perspective proposes to a genome encoding the tasks that will execute on particular cloud instances but also includes various cloud instance types, with the understanding that an appropriate virtual cluster would need at least one master node, an I/O node and several worker nodes. This represents a possible solution to the heterogeneity problem discussed above. However, no experiments are performed to analyze the feasibility of the proposed algorithm and the encoding does not consider the order of executing tasks.

Other approaches [21, 32, 34] model the task-resource mapping as a particle swarm optimization problem (PSO) [16] to minimize the overall cost of execution. However, this limits the optimization to a single objective such as runtime, as current multi-objective particle swarm optimizers only deal with small problems. As shown in Section 4, this is also the case for our simple PSO implementation, which cannot allocate workflows with size greater than 500 tasks. In contrast, we propose a multi-objective algorithm that optimizes both workflow runtime and data transferred, and that runs in under 5 minutes for workflows with 1,000 tasks. Our multi-objective approach, while obtaining a trade-off between runtime and data transfer, still achieves significant improvements in a synthetic simulated environment as well as when executing real workflows in the Pegasus WMS.

3 Execution of Data-Intensive Scientific Workflows

We model a scientific workflow as a directed acyclic graph (DAG) representation of a sequence of tasks in a program that solves a scientific problem [18].[1] Tasks in the workflow are routines that may require one or more input files and that produce one or more output files. Task dependence is represented by the DAG, in which tasks are nodes connected by arcs representing file dependencies. In this section, we propose a formal definition of scientific workflows catered for the problem of scheduling data-intensive workflows. We present our proposed evolutionary approach and discuss in detail our proposed cloud cost model and fitness evaluation.

3.1 Problem Statement

A scientific workflow can be defined formally as the set of tasks $T = \{t_i\}$, of size $m = |T|$, that are organized based on their dependencies in the set $W = \{(t_i, t_j, F_{ij}) | t_i, t_j \in T, F_{ij} \in \Phi\}$. In the DAG representation, a weighted edge between task t_i and t_j has the weight F_{ij}, as shown in Fig. 1. The set Φ contains the file sizes, in gigabytes (GB), transferred between the tasks in the workflow, with F_{ij} representing the total size of the files produced by task t_i that are needed by its child t_j. The task allocation problem looks at assigning workflow tasks on various cloud instances, as shown in Fig. 1, where the six workflow tasks are allocated to three cloud instances.

Scientific workflows are both data and computationally intensive, with file sizes up to hundreds of gigabytes exchanged between tasks. A key challenge is how to allocate tasks on specific cloud instances to ensure a minimum amount of data is transferred between tasks that exchange large files. This is the case for example for the Montage workflow, which has 99 % execution time taken by data operations [14, 15, 25]. Studies show that locality-aware scheduling techniques, which assign a task to the execution site that has most of its input files, can increase the locality of files by 96 %, depending on the structure of the workflow [25]. This has the potential to significantly improve the runtime of the workflow, depending on its structure and file dependencies between tasks. As such, the location of data and the allocation of tasks to specific cloud nodes where data is located or has been previously produced by other tasks, should be treated together by the optimization heuristic. However, because of the particular structure of the workflows as we will show below, a tradeoff will occur between the workflow execution runtime and the size of transferred data.

The optimization of the allocation of tasks on a number of nodes can be defined from a variety of perspectives and using various parameters. For example, from a *cloud user perspective*, of interest are the inter-task communication, the communication topology, and selecting different instance types depending on the task size. Moreover, a trade-off between runtime, transferred data overhead, and cost might occur for large scientific workflows, as more cloud instances (hence an increase in cost and data overhead) might be employed to reduce the runtime. Thus, from a cloud user perspective, optimization functions could look at runtime, cost, data overhead and the number of cloud instances. From a *cloud provider perspective*, issues of importance are the allocation of heterogeneous resources, a minimized inter-process communication, as well as meeting availability and reliability. For a cloud provider, optimization functions could look at profit, performance, meeting deadlines, ensuring security, and reducing energy consumption. Lastly, when clouds from different providers are aggregated in a federated cloud, network latency, reliability, and availability become important challenges. From this perspective, the objective is to increase profit, to ensure appropriate time-zones for sensitive applications running on geographically distributed clouds, and reduce energy consumption. Moreover, the execution of scientific workflows on a parallel platform can be *static* or *dynamic*. A sta-

[1] This definition does not consider the case where the number of iterations for various constructs is not known in advance and thus loop unfolding cannot be perfomed. In this case, the workflow cannot be executed by schedulers such as Condor [28].

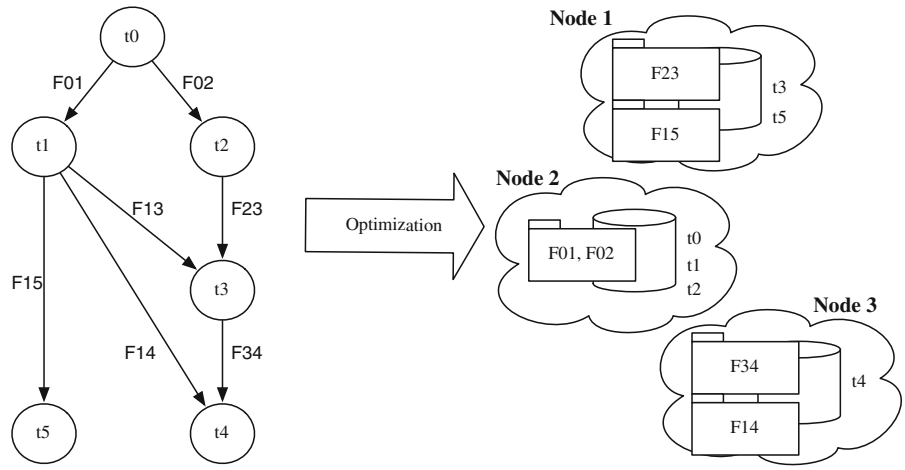

Fig. 1 Allocation of scientific workflows on public clouds

tic allocation considers the initial status of the cloud platform. In the case of dynamic scientific workflows, where the load and output of individual tasks and cloud instances changes over time, a task needs to be moved from one cloud instance to another transparently for the other tasks in the workflow, and at minimal impact on the runtime and cost.

Hereafter, we use the term *node* or *cloud instance* to refer to a virtual machine instance obtained from a cloud provider that offers Infrastructure-as-a-Service [19]. In this paper, we employ the Amazon Elastic Cloud Compute (EC2) service as an example. Our model assumes that tasks are allocated to nodes to be executed in a specific order. In other words, our optimization framework will not only assign tasks to specific nodes but also specify the order in which these tasks are executed, to minimize runtime and data transfers. This is in contrast to existing approaches that only focus on the allocation and do not consider ordering [17].

3.2 Evolutionary Algorithms

In an evolutionary algorithm, a population of chromosomes that encode candidate solutions is evolved or modified in an iterative process to reach better, more optimized solutions that meet as specific goal [12]. The population of chromosomes evolves iteratively as various operators are applied to the population in the current step, which is used in the next step to generate solutions to the problem. The definition of a problem to be solved by evolutionary algorithms involves three main parts, namely, (i) the *representation* of a problem solution as one or more chromosomes, (ii) the *definition* of operators that evolve chromosomes in a population, and (iii) a set of *fitness functions* that determine whether the change performed by the operators brings the population closer to the optimized solution [12]. At each step in the search for an optimizing solution to the problem, an evolutionary framework will apply the operators on two or more chromosomes to evolve the population. The fitness function determines the suitability of new candidates.

In our previous work [27], we have proposed an evolutionary algorithm that optimizes a workflow based on total execution cost that included data transfers as a secondary parameter. Because of this, our algorithm performs well on computationally-intensive workflows but took significant performance hits for data-intensive workflows. Recognizing this, we focus in this paper on obtaining an optimal allocation and ordering of workflow tasks with the aim of reducing the size of transferred data and runtime. A straightforward optimization of transferred data would allocate all workflow tasks on a single cloud instance, which would significantly affect the total execution time. This, together with the correlation between execution time and data transferred as shown below, leads us to propose a multi-objective evolutionary approach that would allow us to optimize both the size of transferred data

and execution time. In contrast, the goal of single objective algorithms is to find the *best* solution, which corresponds to the minimization or maximization of an objective function. To cater for various objectives, the function usually gathers all objectives into a single measure of the goodness of a particular solution. In a single objective algorithm, the solution space is searched towards maximizing the objective function; in contrast, in a multi-objective algorithm, the multiple objectives lead to a compromise solution that highlights the trade-offs between the various objectives, usually in the form of a Pareto front. While single objective algorithms usually give insight into the characteristics of particular problems, they do not offer trade-offs between different objectives and thus are less suited for real-life problems.

Our proposed multi-objective evolutionary algorithm represents the solution to the task allocation of data-intensive workflows using two chromosomes. The first chromosome specifies the allocation strategy and maps individual tasks to cloud nodes. The second chromosome in each candidate solution represents a total ordering over all tasks, as shown in Fig. 2. Specifically, in the first chromosome the index represents the task id in the workflow, from 0 to $m-1$, and the values represent the node id, from 1 to n. The second chromosome contains a task ordering from t_0 to t_{m-1}. For the example in Fig. 1, we have $n = 3, m = 6$, and an encoding as shown in Fig. 2.

With information about both the task allocation and ordering, each solution can be evaluated

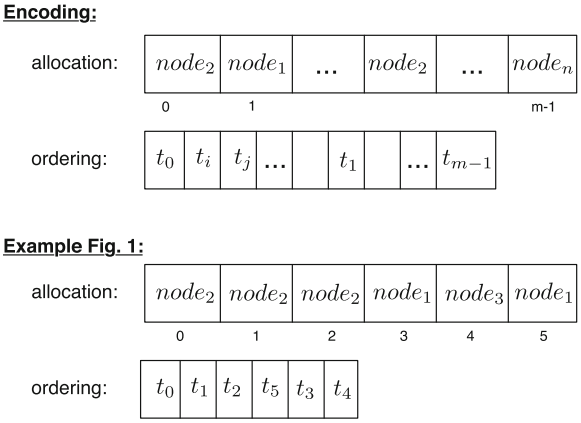

Fig. 2 Chromosome encoding

to estimate the overall execution time and size of data transferred. While there is likely to be a correlation between runtime and data transferred for the execution of scientific workflows, the highly parallel nature of these workflows and data transfers may give rise to significant trade-offs between these two objectives. To handle this, we propose the application of recently developed multi-objective evolutionary algorithms, such as NSGA-II to optimize towards a solution [6]. Rather than evolving solutions towards a single optima, multi-objective evolutionary algorithms maintain a set of optimal solutions—termed collectively a pareto front—that represent different trade-off solutions to be chosen by the scientist. In this case, the NSGA-II algorithm is used to optimize both the allocation and ordering strategies simultaneously by applying appropriate crossover and mutation operators as discussed below.

3.2.1 Fitness Function Evaluation

We propose an evaluation function that returns a two-dimensional array representing the data transferred and runtime respectively of each allocation a. For each allocation a of workflow W on n cloud instances we use the notation

$$a = \{a_1, a_2, , \ldots, a_n\}$$

where a_k represents all the tasks running on the cloud instance k defined as the schedule $a_k = (t_i, \ldots, t_j)$ representing the ordered executions of all tasks between t_i and t_j on cloud instance k. We calculate the data overhead ($DO(a)$) as the sum of the file sizes exchanged by workflow tasks across cloud instances, as below:

$$DO(a) = \sum F_i + \sum F_{ij} \tag{1}$$

where F_i represents all of the output files of task t_i, and $\exists k, l, k \neq l$ such that $t_i \in a_k$ and $t_j \in a_l$, and $(t_i, t_j, F_{ij}) \in W$, that are the files exchanged between task t_i and t_j.

Consider the case shown in Fig. 3, when a parent t_i is allocated to a cloud instance, and all or some of the parent's child tasks t_j are allocated to another cloud instance. In our proposed model, we employ the Scalable Storage Service (S3) model, in which the parent will upload its

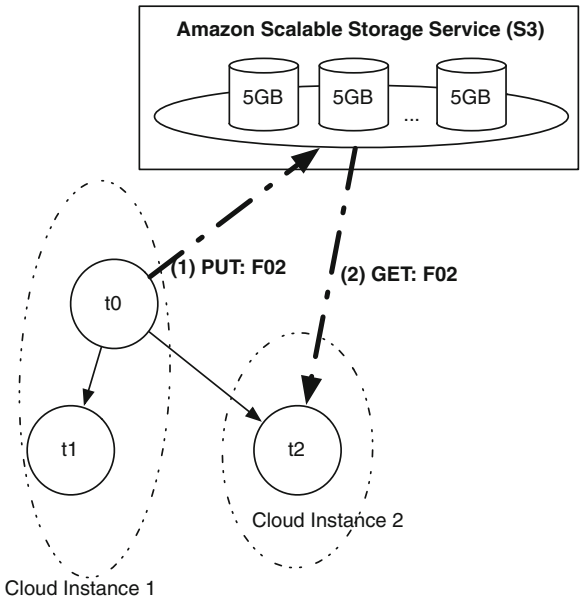

Fig. 3 Using the Amazon scalable storage service (S3)

output files to S3 (the first component in (1)) and the children executing on different instances will download the files from S3 when needed (the second component in (1)), as shown in Fig. 3. The use of S3 is justified by the need of data persistence, parallelized access to buckets, increased bandwidth, and reduced failure rates when compared to virtual disks [10, 31]. Moreover, the scientific workflows we analyzed are characterized by a parent with a large number of children (over 500 in one case) and as such the impact of the single parent upload is minimal.

The runtime of a scientific workflow given a specific allocation a is calculated as the maximum end time of the execution of tasks per each cloud instance ($ET(t_i)$), assuming all input/output files are present, plus the overhead required to transfer the needed files from one cloud instance to another through S3. Thus

$$R(a) = \max_k(\max_i(ET(t_i))) + \frac{DO(a)}{B} \quad (2)$$

where $t_i \in a_k, a_k \in a$ is a task part of the ordered allocation a_k excuting on cloud instance k, and the value of bandwidth B is set to $B = 12$ MB/s as discussed in Section 4. The maximum end time (ET) of tasks per node is calculated using a simple algorithm that at each iteration visits each cloud instance and determines the first task that can be executed. The runtime of that task is then established according to the workflow specification file[2] and the value of the end time variable is increased accordingly. If there is more than one task that can be executed, the algorithm chooses the task with the smallest id. Moreoveor, the calculation of ET only considers scientific workflows without loops, as discussed above. Our future work will extend this calculation to better optimize the next task to be executed as well as consider loop constructs and workflow-specific conditions on task execution. When all tasks have been executed, the maximum end time of all nodes is considered. This follows the assumption that all cloud instances will be shut down only when the last task in the workflow has finished executing.

3.3 Crossover Operators

To evolve the population of chromosomes encoded as shown in Fig. 2, we apply two separate crossover operators, corresponding to the two chromosomes of the parent solutions, with a specified probability. Firstly, a standard single-point crossover [12] is applied to the *allocation* chromosomes of the solutions, where a crossover point is chosen uniformly at random to divide the chromosome. The first segment of the chromosome from one parent is then combined with the second segment of the other parent to yield valid allocation chromosomes for two offspring solutions, as shown in Fig. 4. For clarity, we refer to $node_i$ as the cloud node with id i, where i takes values from 1 to n, the total number of nodes.

In a second step, an order crossover operator is applied to the ordering chromosomes of the parent solutions, as shown in Fig. 5. Such an operator is necessary since the ordering chromosome is constrained by the workflow task-graph and the application of a standard single-point crossover may lead to the generation of unfeasible solutions. For this operator, a crossover point is chosen uniformly at random for each parent solution and the first segment of the chromosomes for the offspring

[2]The runtime of each task varies depending on the workload of the system. We employ both benchmark values and baseline runs as described in Section 4.

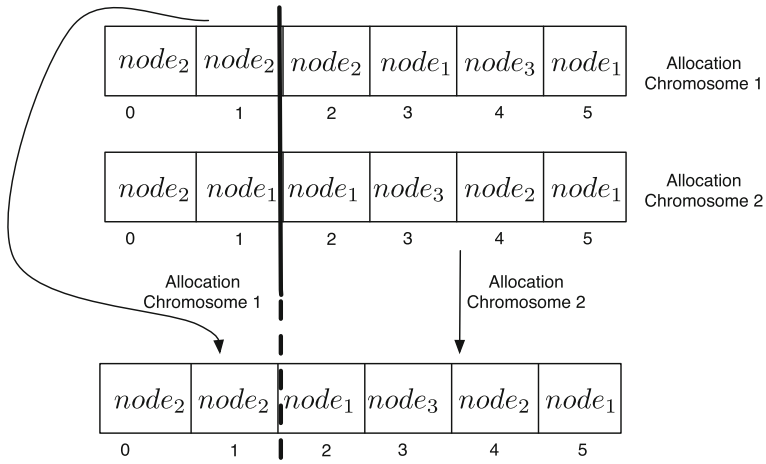

Fig. 4 Single-point crossover operator

solutions are populated as per the standard single point crossover operator. Parts that were not included in the first segment are then added in the same order as they appear in the alternate parent chromosome.

3.4 Mutation Operators

Mutation operators typically involve the changing of parts (also called genes) of the chromosome in some way [12]. We apply a mutation operator with specified probability, to the offspring solutions in a two-step process caused by the task dependencies within the workflow graph. If a solution is chosen for mutation, then each gene within its *allocation* chromosome is randomly flipped, with probability $1/m$, where m is the number of tasks. The flipped gene is assigned a random integer value from the interval $[1, n]$, representing a cloud node id, as shown in Fig. 6.

We cannot apply the same type of mutation to the *ordering* chromosome because it will often lead to solutions that invalidate workflow constraints as represented in the workflow DAG. Instead, the second step of the process applies a *swap* mutation operator to the ordering chromosome. Our implementation pre-processes the input workflow to identify a list of all possible pairs

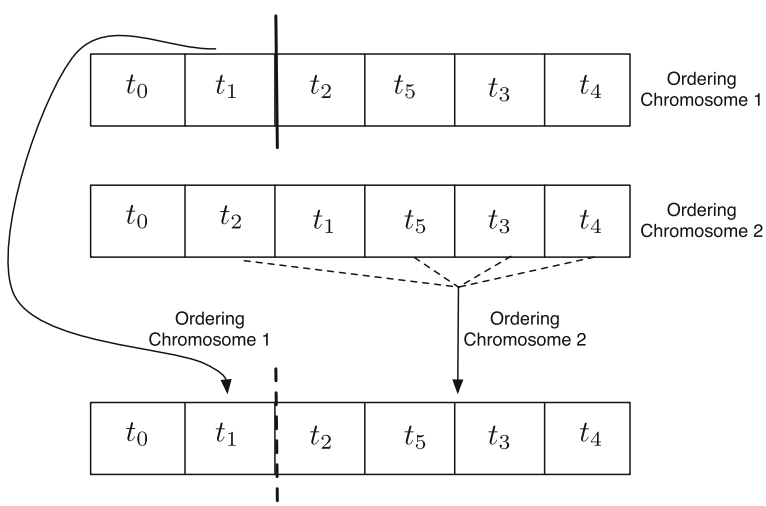

Fig. 5 Order crossover operator

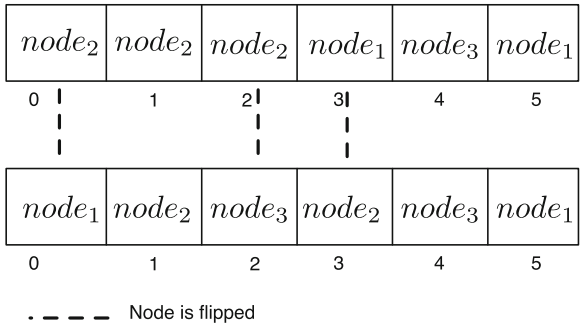

Fig. 6 Mutating the allocation chromosome

of tasks that may be swapped, without violating precedence constraints based on the topological sorts of the workflow. The swap mutation operator randomly chooses one pair from this list and exchanges this pair of values within the ordering chromosome (Fig. 7).

Our multi-objective evolutionary algorithm optimizes the task allocation and ordering simultaneously with respect to both data transfer and runtime. Formulating this problem as a multi-objective optimization allows users of scientific workflows to consider tradeoffs between transferred data and execution time. We embed the problem formulation and operators described above within the standard NSGA-II algorithm, implemented within the jMetal optimization framework. Our experiments show that an optimal configuration is given by a population size of 10, mutation and crossover probabilities of 0.9 and a maximum number of evaluations set to 100,000.

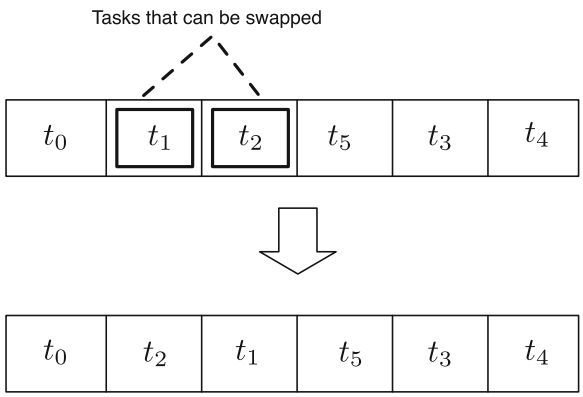

Fig. 7 Swap mutation for the ordering chromosome

4 Experimental Analysis

Our experimental analysis is organized in two main parts. Firstly, a simulation study compares our approach with existing state-of-the-art approaches using both computationally and data-intensive scientific workflows. Our study uses synthetic workflow definitions of varying sizes of up to 1,000 tasks that have been widely used as benchmarks in a variety of projects. Secondly, to further validate our approach, we allocate and execute the Montage workflow obtained from the California Institute of Technology on a cloud made from Amazon EC2 instances. The workflow size varies from 60 to 387 tasks. We compare between the allocation and ordering provided by our multi-objective approach and the allocation employed by the Condor scheduler, based on the total execution time and the size of transferred data. The workflow execution is performed using the Pegasus workflow management system [8], on a cloud of Amazon EC2 instances that use the Amazon Scalable Storage Service. This analysis and comparison of real-life execution of the workflow is paramount for analyzing the feasibility of our approach. This is largely because cloud parameters are not publicized, e.g., bandwidth, migration policies, etc., and the status of a cloud and even specific cloud instances can vary during the execution of the workflow and from run to run. These variations can influence the final runtime and size of transferred data for both approaches, i.e., greedy in Condor, and multi-objective optimization in our proposed approach.

4.1 Experimental Parameters

For our simulation experiments, we use the characteristics of Amazon `c1.xlarge` instances, which are suited for computationally intensive applications as they have high I/O performance and a large memory capacity. The Pegasus project publishes the specifications of a number of scientific workflows shown in Table 1 [22]. The published information includes workflow size, workflow DAG, as well as details about one execution of the workflows (runtime, files transferred, etc.) on a cluster composed of instances equivalent to Amazon `c1.xlarge` instances [14,

Table 1 Scientific workflow characteristics (adapted from [22])

Workflow	Type	Number of tasks	Number of cloud instances
	I/O; memory; CPU		
Montage	High; low; low	25; 50; 100; 1,000	5; 6; 10; 77
CyberShake	Low; low; medium	30; 50; 100; 1,000	3; 5; 9; 5
Inspiral	Low; medium; medium	30; 50; 100; 1,000	3; 3; 9; 50
Sipht	Low; medium; high	30; 60; 100; 1,000	4; 8; 12; 128
Epigenomics	Low; medium; high	24; 46; 97; 997	2; 5; 2; 15

22]. These are synthetic workflows in that the workflow description does not have associated executables and thus cannot be executed. These workflows have been used as benchmarks in a variety of projects and as such are employed in our simulation study [5, 32]. An Amazon `c1.xlarge` instance has a CPU of `20 EC2 Compute Units` and a memory of 7 GB, and a cost of $0.68/h. For storage, we employ Amazon S3 buckets. In our experiments, we employ a number of cloud parameters deduced from various studies, e.g., bandwidth $B = 12$ MB/s [10, 31].

Our framework determines an optimal configuration of tasks executing on a number of cloud instances (n) that form a virtual cloud cluster. It is important to highlight here that determining the optimal value of n for a particular workflow can impact the performance and results of the optimization. An approach would be to set a pre-defined value of n for all scientific workflows. However, as it can be seen in Table 1, the size of the workflows varies from 25 to 1,000 and a single value of n might not be optimal for all. In contrast, we propose a simple heuristic to determine a good value of the number of cloud instances the workflow will be executed on. Our heuristic determines, for each workflow DAG, n as the maximum number of task sets that can be executed in parallel, as shown in Table 1. Our experiments show that the heuristic values of n lead to better results than a fixed value of n, say $n = 10$, for all workflows.

4.2 Comparison with Existing Approaches

We compare between our approach and a simple allocation heuristic and a particle swarm optimization. Our proposed heuristic tries to allocate parent-child pairs of tasks that have large files exchanged between them on the same cloud instance, to reduce the size of data transferred. The heuristic proposes the concepts of *task bundle* and task *parallel sets*. Firstly, tasks t_{i+2}, t_{i+3} in a chain as shown in Fig. 8, e.g. $t_i, t_{i+2}, t_{i+3}, t_{i+4}$ with no children and no execution constraints in the tree, and in which $F_{i+2,i+3}$ has a large size, can be aggregated as a *task bundle*, i.e. executed on the same cloud instance. The heuristic then attempts to determine tasks, either bundled or single, that can be executed in parallel and puts them in *parallel sets*. The tasks in the parallel sets either have no parents or have the same parent and will be assigned to different cloud instances. Based on the task bundles and the parallel sets, the heuristic then goes through the entire workflow tree and allocates tasks to cloud instances to minimize overhead as discussed above. The allocation is then evaluated using the evaluation function discussed in Section 3.2.1. As shown in Tables 2 and 3, this heuristic obtains good results for workflows of small size, but does not perform well for large workflows, because of the large number of children (more than 500) for each parent that are allocated on different cloud instances and thus incur a high overhead.

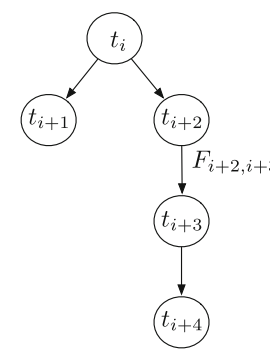

Fig. 8 Task bundle heuristic

Table 2 Comparison of optimization methods with respect to runtime (h)

Workflow	Heuristic allocation	PSO				Multi-objective			
		Mean	Std. dev.	Max	Median	Mean	Std. dev.	Max	Median
Montage 25	1.46	8.76	0.04	8.77	8.77	0.02	0.003	0.037	0.027
Montage 50	3.15	18.97	0.09	19.03	19.02	0.24	0.35	0.98	0.03
Montage 100	7.34	38.42	0.09	38.49	38.47	1.54	0.93	3.37	1.38
Montage 1,000	79.72	–	–	–	–	52.94	2.17	58.95	52.13
CyberShake 30	37.73	203.36	0.002	203.36	203.36	0.068	0.01	0.08	0.06
CyberShake 50	52.30	281.65	0.002	281.65	281.65	0.08	0.01	0.09	0.07
CyberShake 100	100.30	540.50	0.02	540.54	540.50	0.11	0.001	0.12	0.11
CyberShake 1,000	1,252.89	–	–	–	–	0.14	0.01	0.17	0.15
Inspiral 30	0.18	1.22	0.03	1.32	1.23	0.37	0.06	0.37	0.37
Inspiral 50	0.50	1.58	0.05	1.70	1.57	0.39	0.03	0.41	0.37
Inspiral 100	0.60	2.12	0.07	2.29	2.11	0.37	0.05	0.41	0.36
Inspiral 1,000	2.94	–	–	–	–	2.03	0.19	2.45	2.04
Sipht 30	1.67	3.87	0.009	3.88	3.88	0.99	0.09	1.01	1.01
Sipht 60	2.78	7.65	0.05	7.66	7.63	1.20	0.01	1.42	1.20
Sipht 100	4.00	10.93	0.28	11.23	10.93	1.35	0.13	157	1.35
Sipht 1,000	29.06	–	–	–	–	15.06	0.88	16.53	14.97
Epigenomics 24	13.63	73.6	6.20	79.19	76.18	1.72	0.02	1.94	1.55
Epigenomics 46	18.21	114.18	19.60	156.85	100.62	1.96	0.001	1.94	1.03
Epigenomics 97	35.84	156.49	21.73	170.16	167.68	8.42	0.11	8.53	8.38
Epigenomics 997	443.80	–	–	–	–	21.05	3.64	31.90	22.72

We implemented a simple particle swarm optimization (PSO) algorithm, as proposed in [32]. The algorithm was implemented using jSwarm [13] and constructs a particle with size equal to the number of tasks. The PSO uses the same evaluation functions described in (1) and (2). We exe-

Table 3 Comparison of optimization methods with respect to data overhead (GB)

Workflow	Heuristic allocation	PSO				Multi-objective			
		Mean	Std. dev.	Max	Median	Mean	Std. dev.	Max	Median
Montage 25	2.26	53.43	0.001	53.44	53.42	0.77	0.23	1.36	0.68
Montage 50	3.8	44.47	0.001	45.82	44.45	1.17	0.36	1.51	1.36
Montage 100	6.95	62.15	0.001	63.29	62.15	2.44	0.70	3.81	2.35
Montage 1,000	64.55	–	–	–	–	45.32	2.00	50.45	44.85
CyberShake 30	31.01	70.74	0.001	70.74	70.74	1.20	0.21	1.36	1.36
CyberShake 50	43.99	57.93	0.001	57.93	57.93	1.36	0.005	1.36	1.36
CyberShake 100	82.52	153.93	0.001	82.52	82.52	1.36	0.001	1.36	1.36
CyberShake 1,000	910.35	–	–	–	–	1.37	0.003	1.38	1.37
Inspiral 30	2.74	128.53	0.27	129.76	129.20	1.08	0.33	1.36	1.36
Inspiral 50	4.11	148.82	0.12	150.32	147.64	1.31	0.17	1.36	1.36
Inspiral 100	4.13	379.12	0.48	379.87	358.70	2.05	0.001	2.05	2.04
Inspiral 1,000	7.35	–	–	–	–	11.39	1.08	13.98	11.25
Sipht 30	4.22	17.13	0.001	17.80	16.96	1.15	0.31	1.36	1.20
Sipht 60	11.12	81.91	0.73	82.64	81.47	2.14	0.37	2.77	2.07
Sipht 100	13.32	145.23	2.38	146.96	144.23	2.98	0.58	4.15	2.79
Sipht 1,000	98.59	–	–	–	–	57.31	2.88	61.72	57.85
Epigenomics 24	15.12	157.15	0.38	15.78	15.09	1.69	0.39	2.13	1.36
Epigenomics 46	22.45	325.26	5.71	23.25	22.20	2.13	0.001	2.13	2.13
Epigenomics 97	62.88	350.28	14.65	351.96	349.42	6.51	0.37	6.85	6.83
Epigenomics 997	372.53	–	–	–	–	19.81	3.88	27.68	19.40

cuted 30 independent optimization runs for each workflow-objective combination and recorded the results. The mean and standard deviation together with maximum and median values for these runs are shown in Tables 2 and 3. Our implementation obtains results similar to the above heuristic for workflows of small size, below 100 tasks. However, the particle swarm implementation does not scale beyond workflows with more than 500 tasks. A solution for this known problem is suggested in [21] and involves performing a pre-ordering of tasks followed by allocation of workflow tasks on a first-come first-served basis. Since this contradicts our problem formulation, we have not implemented this solution here.

Our implemented multi-objective evolutionary algorithm optimizes the task allocation and ordering simultaneously with respect to both runtime and data transfer. We embed the problem formulation and operators described above within the standard NSGA-II algorithm, implemented within the jMetal optimization framework [9]. We configure this algorithm with a population size of 10, mutation and crossover probabilities of 0.9 and a maximum number of evaluations set to 100,000. We executed 30 independent optimization runs for each workflow to minimize, simultaneously, the overall runtime and size of transferred data on a commodity desktop Intel Core i5 at 2.7 GHz with 16 GB RAM.

Tables 2 and 3 summarize the experimental results and list the mean runtime and transferred data optimization results for each algorithm-workflow combination and the standard deviation across all runs for the stochastic algorithms trialled. An examination of this data shows that the multi-objective algorithm clearly out-performs the heuristic allocation and particle swarm optimization approaches, particularly for workflows with larger number of tasks. The multi-objective algorithm out-performs the heuristic and particle swarm optimization approaches even in the worst case, when considering the maximum values obtained by the multi-objective algorithm. This is also confirmed statistically through the application of Mann–Whitney–Wilcoxon tests, in which results show, with a significance level that is less than 0.001, that the multi-objective algorithm achieves better results for both runtime and overhead optimization.

4.3 Workflow Execution on Public Clouds

The architecture for scheduling scientific workflows on public clouds has a supporting system comprising of two layers, namely, the workflow management system and the task distribution system [3, 7]. The workflow management system is used for designing and constructing scheduled workflows that are then sent to the task distribution system layer, which does the actual distribution and execution of tasks on the computing resources. In this experiment we employ the Pegasus workflow management system [8] that runs on top of the Condor scheduler [28] to schedule workflows on Amazon EC2 instances. Condor has a powerful and flexible resource selection mechanism called requirement matching, which allows user to define for each task what kind of resource it should be run on. Requirements include CPU architecture, operating system on the resource, free disk space and memory size. A task submitted to Condor can only be run on a resource that meets the defined requirements. Other requirements include security level, domain and specific host-names. If no requirement is specified, Condor allocates tasks to available resources, using a mechanism that ranks resources based on their processor power, available memory, free disk space, and recent activities, and allocates queuing task to the resource with the highest rank. Besides being a popular Grid scheduler, Condor is also used by Pegasus WMS to schedule scientific workflows on the cloud, and as such we use it as a benchmark in our experiments.

The Pegasus WMS is a widely used workflow management system in the field of scientific workflow management [8] and supports a wide range of platforms, from desktop environments to public clouds. Pegasus has been tested with a variety of workflows with sizes varying from tens to millions of tasks. It has a set of predefined heuristics to schedule workflows into Grid or cloud resources, but it can be configured to produce a scheduled workflow as the user instructs. The combination of Pegasus and Condor as a

supporting framework for uploading workflows into the cloud has been previously evidenced in [3]. However, in those initial experiments, a single cloud instance was considered, and was used to host the workflow management system and run the workflow processes. Improving on this, we employ Pegasus 4.0 and Condor 7.8 that are running on Amazon EC2 `c1.xlarge` instances and employ Amazon S3 as cloud storage.

Our experimental setup is shown in Fig. 9. In our setup, a local client submits the workflow to a submit host that runs on an EC2 instance. The submit host runs the Pegasus planner and the Condor scheduler. The submit host is configured such that all the task executables and initial input files are sent to S3 buckets, and that tasks dispatched to worker instances are configured to read input files from the S3 buckets. While the workflow is executed, the executables are uploaded to worker instances from the S3 buckets, along with the input files. This is the default Condor behavior, and to our knowledge, cannot be changed, e.g., to allow intermediate data products to remain on the cloud instance for future tasks that might need them. When the workflow completes execution, the submit host obtains the final output from S3 and sends it back to the local machine.

In this experiment, we employ a real Montage workflow obtained from the California Institute of Technology. The workflow consists of several layers of parallel processing (*mProjectPP*, *mDiffFit*, and *mBackground* tasks), data aggregation (*mConcatFit*, *mImgTbl* task), data partitioning, and pipelining tasks as shown in Fig. 10. The workflow employed in this experiment has the same structure as the one showed in Fig. 10, but has more parallel processing tasks, resulting in three workflows of sizes 60, 157, and 387 respectively. It's I/O type and the large number of processing tasks that can be executed in parallel, i.e., tasks of type *mDiffFit* and *mBackground*, makes Montage suitable for our experiments and our Amazon S3 model, as the large number of children that use the output from the same parent imply that a) the intermediate data product can be saved into a shared storage and b) reduced execution time can be obtained by parallelizing the execution of the children tasks over more than one cloud instance. We further discuss this in Section 4.4.

4.3.1 Baseline Run

Our proposed multi-objective algorithm requires runtime and input/output file size estimations to be included in the workflow definition. Towards this, we perform a baseline run by executing the workflow on an Amazon EC2 `c1.xlarge` instance and collecting the runtime and input/output file sizes for each task, that are then appended

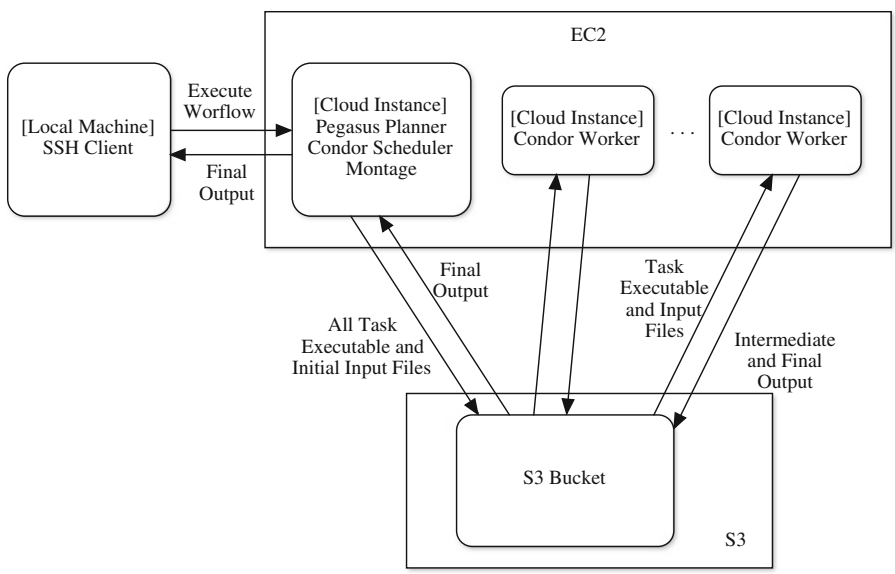

Fig. 9 Experiment setup

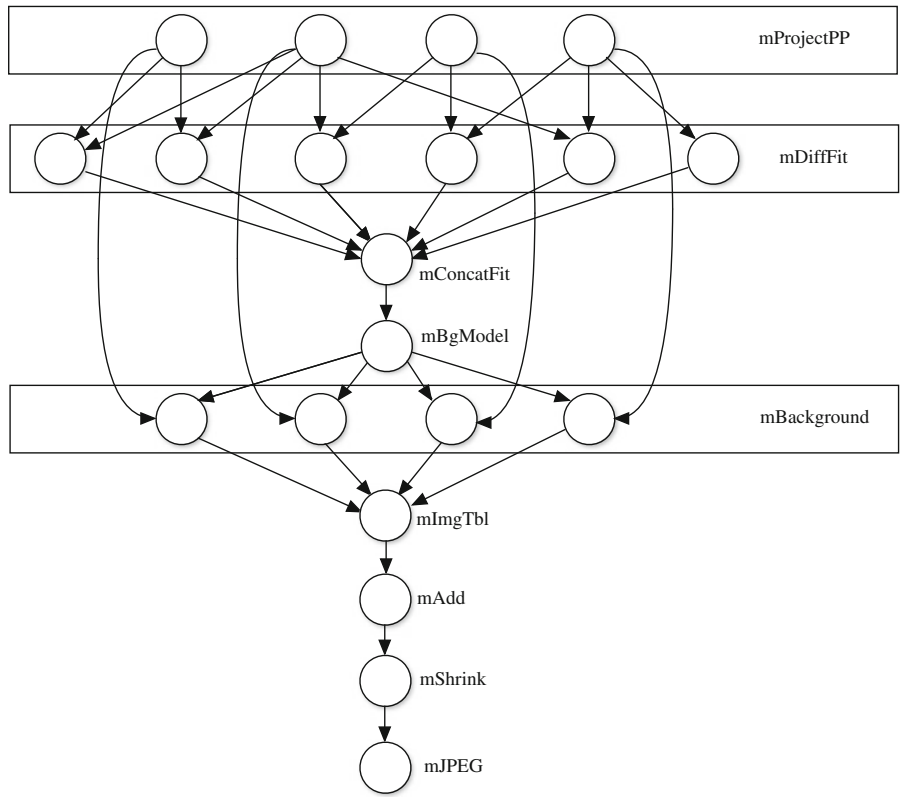

Fig. 10 Structure of the Montage workflow

to the workflow description that is used for our algorithm. It is important to note here that, while this baseline run collects values for the execution times and file sizes exchanged, only the relative sizes of the execution times are important. The execution time may vary depending on the status of the cloud and of the specific cloud instances. However, in the optimization problem, only the relative execution time of a task compared to other tasks in the workflow is relevant when minimizing runtime.

4.3.2 Experiment Results

In this experiment, we compare between the workflow runtime and transferred data overhead of the Condor allocation and that obtained using our framework. When analyzing our evolutionary approach, we look at a single objective optimization of runtime, and the multi-objective optimization of runtime and overhead discussed above. The single objective optimization employs the same operators as the multi-objective optimization, but aims at minimizing runtime only. We are using the comparison with a single objective optimization of runtime instead of transferred data overhead because a single objective algorithm that would maximize data locality will allocate all the tasks to be executed on a single cloud instance. The results shown in Table 4 compare between the runtime and overhead when using Condor to schedule the workflow using its default scheduling and allocation policy, i.e., greedy, when using the allocation obtained from the single-objective optimization, and when using the allocation obtained from our proposed multi-objective algorithm.

The results present the average and the best for the execution of the workflows on a cluster with n instances, where n is determined as above using our heuristic, as 7, 10, and 13 for workflows of size 60, 157, and 387 respectively. For comparison, the runtime of executing the Montage workflow on a single cloud instance is around 3, 7, and 23 h respectively.

Table 4 shows that our multi-objective approach achieves comparable results to the Con-

Table 4 Execution of Montage workflow on an EC2 cloud

Workflow size	Algorithm	Runtime (s)			Transferred data overhead (GB)
		Average	Best	Stdev	
60	Single-objective	385	341	25	5.45
	Multi-objective	366	338	19	
	Condor	352	340	9	
157	Single-objective	576	521	23	10.06
	Multi-objective	572	541	19	
	Condor	518	510	9	
387	Single-objective	1,129	1,011	82	28.5
	Multi-objective	1,009	982	77	
	Condor	1,002	965	31	

dor default scheduling and allocation policy in terms of runtime. This is surprising, considering the promising results of the simulation study. On closer scrutiny, the Condor task manager transfers *all* intermediate data products to the S3 buckets and out of the cloud instance, regardless of whether the next task to be executed needs the intermediate data product to continue execution. If the next task to be executed on the cloud instance, e.g., task `t2` in Fig. 3, needs the output file produced by the previous task, e.g., task `t1` in Fig. 3, this will incur twice the cost of transferring F_{12} to and from the S3 storage. The fact that all files are transferred to and from S3 by the Condor scheduler means that in this case, the data transferred is the same for all algorithms, as shown in the last column in Table 4.

To prevent this, we implement our own task execution routine that, following the allocation and schedule produced by the multi-objective algorithm, executes the workflow tasks on the specific cloud instance and facilitates data locality by transferring intermediate data products to the S3 storage only when they are needed by child tasks running on other cloud instances, e.g., task `t3` in Fig. 3. We compare the data transferred using our routine (`MO-Routine`) with the data overhead obtained when using the default Condor scheduling and allocation policy but executed using our routine (`Condor-Routine`). Table 5 shows the results.

As it can be seen, the execution of the allocation using our task execution routine and the multi-objective solution results in significantly less data transferred to and from the S3 storage. This has a significant impact as the scientific workflows increase in size, with an increase in overhead reduction from 25 % to 45 %. The reduction also has a positive effect on the runtime, as shown in Fig. 11. Unsurprinsingly, for all workflow sizes, the single-objective optimization that only considers the workflow runtime achieves the worst performance improvement when using our scheduling routine. In contrast, the multi-objective optimization obtains a 10 % decrease in runtime compared to the `Condor-Routine`, which is caused by an 80 % reduction in the data transferred.

4.4 Discussion

It is important to highlight that our proposed approach is suitable for cases where the scientific workflow sees a large number of children tasks that require the same output file from the same parent tasks, and the children tasks are performing computations on different parts of the same input. This is a case that is frequently encountered

Table 5 Size of transferred data when exploiting data locality

Algorithm	Transferred data overhead (GB)		
	Montage 60	Montage 157	Montage 387
SO-Routine	0.194	0.842	3.334
MO-Routine	0.185	0.682	2.954
Condor-Routine	0.874	2.121	4.293

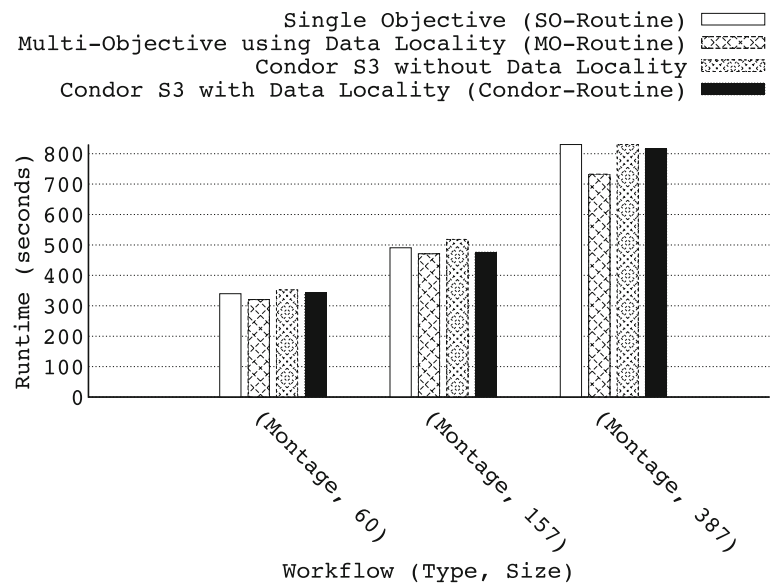

Fig. 11 Execution runtime when exploiting locality

in practice [22]. This scenario also justifies the use of a scalable storage separate from the cloud instance that would permit, similar to S3 in the case of the Amazon EC2 model, data persistence, parallelized access to buckets, increased bandwidth, and reduced failure rates when compared to virtual disks [10, 26, 31]. This scenario also imposes a trade-off between the data overhead and the runtime and cost of executing the scientific workflow. Of course, the entire workflow could be executed on a single cloud instance, disregarding its execution time and inherent cost. We envisage a wide adoption in the scientific community of the execution of scientific workflows using public clouds where the reduced cost and runtime of executing the children tasks in parallel would fully benefit the entire community, e.g., would allow students to execute large-scale experiments at reduced costs.

Another important point to note is that our proposed approach is a static allocation approach that calculates the optimal schedule and ordering of tasks based on our proposed model, using information obtained from a baseline execution of the workflow on a single cloud instance. However, cloud characteristics such as bandwidth and computational capability can vary drastically during the execution of the workflow. In our future work, we propose to look at a dynamic allocation approach that would integrate real-time cloud monitoring information. This implies the design and implementation of a multi-objective allocation algorithm capable of receiving cloud status information and adapting or changing the search direction in real time, which poses a significant research challenge. Next, the wide adoption of cloud computing implies that more than one cloud provider could be considered by our approach, and that the scientific workflow could run on a hybrid cloud. Our approach could be extended in two ways to cater for this scenario. Firstly, the varying cloud characteristics could be modeled in our system as a function of the cloud type. This would imply that constants such as bandwidth, etc., would become functions of the cloud provider, meaning that the multi-objective algorithm could be modified in a similar manner as for the dynamic allocation scenario. Secondly, the cloud provider could become an optimization parameter in the multi-objective algorithm: this would result in a bi-level multi-objective algorithm, in which the choices of various cloud providers could represent one level of optimization, and from this choice, the modified multi-objective algorithm could execute on the second level. Finally, different cloud providers may provide different types of cloud instances.

To cater for this heterogeneity, the instance type could be encoded in the genome similar to previous approaches [30].

5 Conclusion and Future Work

The allocation of scientific workflows on public clouds is an NP Complete problem that can be addressed from a variety of perspectives. In this paper, we propose a multi-objective evolutionary algorithm that optimizes the *workflow runtime* and *size of transferred data* of data-intensive scientific workflows. We embed within this algorithm a simple and extensible cost model to evaluate a specific allocation and ordering of tasks executed on a cluster of cloud instances, and employ problem-specific mutation and crossover operators to generate new solutions. Our two-stage crossover and mutation operators follow problem and optimization constraints. Specifically, we encode the solution to the allocation problem using an *allocation* chromosome that defines the task allocation to particular cloud instances, and an *ordering* chromosome that defines the order in which tasks are executed on the cloud instances. We employ *single-point crossover* operators for the allocation chromosomes and *order* crossover operators for the ordering chromosome that is constrained by the definition of the scientific workflow. In a similar manner, the allocation chromosome uses a traditional mutation operator, whereas a swap mutation is employed for the ordering chromosome. Contrary to existing work, our approach proposes both the allocation and the ordering of the tasks to be executed on the cloud instances to minimize runtime and exploit locality, by ensuring that intermediate data products remain the cloud instances that have scheduled tasks that need them.

We validate our implementation using simulations, as well as experiments on the Amazon EC2 public cloud. The simulations employ synthetic workflow data describing both computationally and data-intensive workflows of medium to large sizes for up to 1,000 tasks. Our results show that our multi-objective approach significantly outperforms solutions generated by particle swarm implementation and simple heuristic algorithms. Specifically, the multi-objective approach offers an 70 % improvement in data transferred for larger workflows, e.g., 1,000 tasks, when compared to a heuristic allocation. In contrast, a particle swarm implementation has been shown not to scale beyond workflows with more than 500 tasks. As the simulation experiments use descriptions of synthetic workflows of sizes that are not available for execution, and employs estimated values for cloud parameters such as bandwidth, it is important to further validate our results using a real-life workflow deployed on an existing cloud platform.

Our experiments on the Amazon EC2 public cloud employ a data-intensive scientific workflow used by astronomers at the California Institute of Technology. The workflow size ranges from 30 to 400 tasks. Our experiments compare between the runtime and data transferred obtained when running the workflow using the Pegasus Workflow Management System that employs the Condor scheduler and our own scheduling and allocation routine that follows the solution proposed by the multi-objective algorithm. Our results show that the data locality optimization performed by the multi-objective algorithm outperforms the Condor scheduler by 10 % for both small and large workflows. Specifically, for workflows of 387 tasks, our multi-objective algorithm executed using our scheduling and execution routine achieves a reduction in the size of transferred data of 26 GB when executed on a cluster of Amazon S3 instances over the Pegasus execution.

The allocation of scientific workflow tasks on public clouds can be defined using a variety of parameters. In this paper, we have only considered execution runtime and data transferred, but other parameters, such as heterogeneous cloud instance types and optimizing the number of cloud instances, can be considered in future work. Moreover, we are confident that the integration of real-time monitoring of the cloud instances with the multi-objective algorithm will result in significant performance gains. This dynamic allocation requires the integration of cloud monitoring services within the evolutionary framework and poses significant design and implementation challenges that will be addressed in future work.

References

1. Abramson, D., Enticott, C., Altinas, I.: Nimrod/K: towards massively parallel dynamic grid workflows. In: Proceedings of the ACM/IEEE Conference on Supercomputing, pp. 24:1–24:11 (2008)
2. Arstechnica: $1,279 per hour, 30,000-core cluster built on Amazon EC2 cloud. http://arstechnica.com/business/news/2011/09/30000-core-cluster-built-on-amazon-ec2-cloud.ars. Last retrieved Oct. 2012
3. Berriman, G.B., Deelman, E., Groth, P.T., Juve, G.: The application of cloud computing to the creation of image Mosaics and management of their provenance. In: CoRR (2010)
4. Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.-H., Vahi, K.: Characterization of scientific workflows. In: Proceedings of the Third Workshop on Workflows in Support of Large-Scale Science, pp. 1–10 (2008)
5. Catalyuek, U.V., Kaya, K., Ucar, B.: Integrated data placement and task assignment for scientific workflows in clouds. In: Proceedings of the Fourth International Workshop on Data Intensive Distributed Computing (2009)
6. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
7. Deelman, E., Singh, G., Livny, M., Berriman, B., Good, J.: The cost of doing science on the cloud: the Montage example. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–12 (2008)
8. Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A., Jacob, J.C., Katz, D.S.: Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Sci. Program. **13**, 219–237 (2005)
9. Durillo, J., Nebro, A., Luna, F., Dorronsoro, B., Alba, E.: jMetal: a java framework for developing multi-objective optimization metaheuristics. University of Málaga, Technical Report ITI-2006-10 (2006)
10. Evangelinos, C., Hill, C.N.: Cloud computing for parallel scientific HPC applications: feasibility of running coupled atmosphere-ocean climate models on Amazon's EC2. In: Cloud Computing and its Applications (2008)
11. Fernandez-Baca, D.: Allocating modules to processors in a distributed system. IEEE Trans. Softw. Eng. **15**, 1427–1436 (1989)
12. Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)
13. JSwarm: PSO optimization package. http://jswarm-pso.sourceforge.net/. Last retrieved Oct. 2012
14. Juve, G., Deelman, E., Vahi, K., Mehta, G., Berman, B.P., Berriman, B., Maechling, P.: Scientific workflow applications on Amazon EC2. In: Proceedings of the International Conference on E-Science, pp. 59–66 (2009)
15. Katz, D.S., Jacob, J.C., Berriman, G., Good, J., Laity, A.C., Deelman, E., Kesselman, C., Singh, G., Su, M.-H., Prince, T.A.: A comparison of two methods for building astronomical image mosaics on a Grid. In: Proceedings of the International Conference on Parallel Processing, pp. 85–94 (2005)
16. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks, pp. 1942–1948 (1995)
17. Kwok, Y.-K., Ahmad, I.: Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. J. Parallel Distrib. Comput. **47**, 58–77 (1997)
18. Ludascher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E.A., Tao, J., Zhao, Y.: Scientific workflow management and the Kepler system. Concurr. Comput. Pract. Exp. **18**, 1039–1065 (2006)
19. National Institutes of Standards and Technology: Cloud computing synopsis and recommendations. http://csrc.nist.gov/publications/drafts/800-146/Draft-NIST-SP800-146.pdf. Last retrieved Oct. 2012
20. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., Li, P.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics **20**, 3045–3054 (2004)
21. Pandey, S., Wu, L., Guru, S., Buyya, R.: A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications, pp. 400–407 (2010)
22. Pegasus Project: Workflow generator. https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator. Last retrieved Oct. 2012
23. Prodan, R., Wieczorek, M.: Negotiation-based scheduling of scientific Grid workflows through advance reservations. J. Grid Comput. **8**(4), 493–510 (2010)
24. Prodan, R., Wieczorek, M., Fard, H.M.: Double auction-based scheduling of scientific applications in distributed Grid and cloud environments. J. Grid Comput. **9**(4), 531–548 (2011)
25. Shibata, T., Choi, S., Taura, K.: File-access patterns of data-intensive workflow applications. In: Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp. 522–525 (2010)
26. Singh, G., Livny, M., Berriman, B., Good, J.: The cost of doing science on the cloud: the Montage example. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–12 (2008)
27. Szabo, C., Kroeger, T.: Evolving multi-objective strategies for task allocation of scientific workflows on public clouds. In: IEEE Congress on Evolutionary Computation, pp. 1–8 (2012)
28. Thains, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the condor experience. Concurr. Comput. Pract. Exp. **17**(2-4), 323–356 (2005)

29. Vockler, J.-S., Juve, G., Deelman, E., Rynge, M., Berriman, G.B.: Experiences using cloud computing for a scientific workflow application. In: Proceedings of the 2nd Workshop on Scientific Cloud Computing (2011)
30. Vockler, J.-S., Juve, G., Deelman, E., Rynge, M., Berriman, G.B.: Integration of workflow partitioning and resource provisioning. In: Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 764–768 (2012)
31. Walker, E.: Benchmarking Amazon EC2 for high-performance scientific computing. USENIX Login **33**, 18–23 (2008)
32. Wu, Z., Ni, Z., Gu, L., Liu, X.: A revised discrete particle swarm optimization for cloud workflow scheduling. In: International Conference on Computational Intelligence and Security, pp. 184–188 (2010)
33. Yigitbasi, N., Iosup, A., Epema, D.: C-meter: a framework for performance analysis of computing clouds. In: Cluster Computing and the Grid, pp. 472–477 (2009)
34. Yu, J., Buyya, R.: A taxonomy of workflow management systems for grid computing. J. Grid Comput. **3**(3–4), 171–200 (2005)
35. Yuan, D., Yang, Y., Liu, X., Chen, J.: A data placement strategy in scientific cloud workflows. Futur. Gener. Comput. Syst. **26**, 1200–1214 (2010)