Multi-objective Optimization of Scheduling Dataflows on Heterogeneous Cloud Resources

Ilia Pietri[1], Yannis Chronis[1], Yannis Ioannidis[1,2]
[1]*Department of Informatics and Telecommunications, University of Athens, Greece*
[2] *ATHENA Research and Innovation Center, Greece*

Submission Type: Research

May 6, 2017

Abstract

Elasticity makes cloud computing an attractive platform for executing complex large-scale expensive dataflows, as it enables different trade-offs between execution time and monetary cost, by varying the number of resources to be provisioned. With cloud providers offering heterogeneous types of resources with different performance and price characteristics, the problem of identifying the various trade-offs available is a great challenge, as the number of possible alternative configurations increases significantly compared to a homogeneous environment, which is itself already computationally difficult. This paper proposes a novel algorithm for dataflow scheduling on heterogeneous clouds that identifies solutions (schedules) close to the optimal pareto front, by exploring the search space in an efficient way. The results of an experimental comparison with the state of the art show that, in several cases, the proposed algorithm provides a richer, more diverse set of solutions, several of which are characterized by significantly better time-money trade-offs.

1 Introduction

Big data applications may often involve execution of expensive queries and processing of large amounts of data. Such queries are often represented as dataflows that describe large-scale computations (operators) and data flow dependencies between them. Distributed systems have been widely used for the execution of this type of applications. Among these, cloud computing is rather popular as it allows resource provisioning on demand, on a pay-per-use basis. As cloud providers offer to users the flexibility to select from a large number of possible configurations of resources on which to schedule their dataflows, the challenge of optimizing both application performance and monetary cost has emerged.

Distinct optimization objectives may often be conflicting. For example, using a large number of resources to execute independent operators (operators without any data dependencies between them) will usually lead to better performance but higher cost than using a single resource to execute everything. Hence, different configurations can be chosen to obtain different time-money trade-offs. n terms of performance and price characteristics, heterogeneity of resources makes the scheduling problem even more complex; the possible combinations of different resource types, each with a different number of virtual machines (VMs) of each type used, result in a significantly larger space of alternative trade-off solutions than the homogeneous case. Nevertheless, heterogeneous configurations may also lead to faster and/or cheaper execution schedules (plans) compared to homogeneous configurations. Multi-objective query optimization aims at exploring the space of potential solutions to identify schedules that exhibit the best trade-offs between conflicting objectives.

In this paper, we present "Homogeneous to Heterogeneous Dataflow Scheduling" (HHDS), a scheduling algorithm for executing dataflows on heterogeneous clouds. The algorithm tries to identify a set of solutions close to the optimal pareto front with diverse time-money trade-offs exploring the search space in an efficient way. In contrast to related work, the algorithm starts with the space of homogeneous configurations to identify an initial set of schedules with good trade-offs and gradually moves towards heterogeneous configurations to further improve the current pareto front (skyline). To do so, the types of VMs used at each solution are modified to create new schedules when lower execution time or monetary cost can be obtained and the pareto front is updated. As the number of skyline solutions may be large, a pruning method is developed to identify several representative schedules. Pruning aims at distributing the points across

the skyline so that there are more solutions to the knees of the pareto curve and a fewer number of solutions further away from the knees. The efficiency of the proposed approach to provide a better and more divergent skyline is demonstrated through the results of an experimental evaluation and comparison with the state of the art.

The main contributions of this work are the following:

- We develop a two-stage heuristic that addresses the multi-objective problem of dataflow scheduling on heterogeneous cloud resources.
- We propose a novel pruning method that favors large numbers of representative pareto-efficient schedules at sharper parts of the pareto curve and leaves fewer points at flatter parts to distribute them along the pareto curve.
- We provide an experimental evaluation to show the effectiveness of the proposed approach using realistic dataflows.

The remainder of the paper is organized as follows. Related work is included in Section 2. The problem description follows at Section 3 and the proposed algorithm is presented in Section 4. The experimental evaluation and its results are described in Section 5, while Section 6 concludes the paper.

2 Related Work

Dataflow scheduling and resource provisioning on distributed systems like grid and cloud computing is the focus of many studies which address different optimization goals [1, 3, 6, 9, 15, 17, 23, 24]. Among these, execution time and monetary cost minimization has received considerable attention.

Several algorithms focus on single-objective optimization or constrained single-objective optimization. For example, the Heterogeneous Earliest Finish Time (HEFT) algorithm in [18] is a well known heuristic that maps tasks to heterogeneous resources selecting the slot with the earliest finish time. The objective of the list-scheduling algorithm is to minimize the overall execution time. The cost-based approach in [3] optimizes the monetary cost by determining the minimum resource capacity required to complete all the tasks within the specified deadline. Cost-based scheduling within a deadline is also the subject in [23]. The proposed algorithm supports rescheduling of tasks that are not yet executed to adapt to unpredictable execution delays.

Other studies like [12, 15, 17] focus on multi-objective approaches. The scheduling algorithm in [17] is an extension of HEFT and consists of two stages. It initially maps each task to the most cost efficient VM, combining task execution time and monetary charges of VMs at an objective function. A user preference parameter is incorporated into the objective function to introduce the concept of Pareto dominance and balance the trade-off between the two objectives. The second approach further reduces the cost of non critical tasks by extending their execution time. The algorithm in [12] is another extension of HEFT that uses a cost conscious parameter to balance the trade-off and generate a pareto-optimal set of solutions. Such approaches address the multi-objective optimization problem as a weighted single-objective problem.

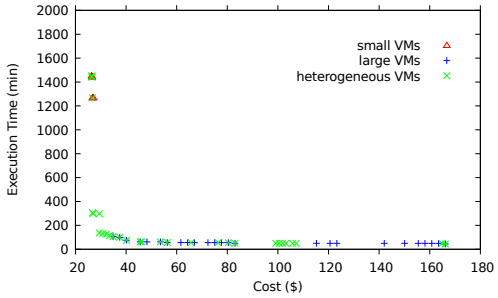
The work proposed in this paper is more related to pareto-based approaches like [1, 4, 6]. Two single-objective approaches which can generate a set of solutions to address the problem of cost or execution time minimization on clouds are proposed in [1]. The two approaches are then combined into a pareto-based approach to select only non-dominated solutions. MOHEFT [6] is a pareto-based algorithm that extends HEFT by keeping a set of partial solutions at each step instead of a single allocation considering heterogeneous resources. The solutions at the skyline are pruned based on a metric called crowding distance which tries to measure the surrounding area of each skyline point where no other skyline point exists.

Multi-objective query optimization (MOQO) [8, 19] focus on determining the join orders of the operators to develop a set of pareto-optimal query plans based on multiple cost metrics in addition to execution time. The work in [19] proposes approximation schemes to solve the multi-objective query optimization problem in cloud environments. However, such MOQO schemes are complementary to our work, as they focus on a different level of optimization for query plans.

Pruning schemes to select a number of appropriate (representative) pareto-efficient solutions has also received attention [14, 20]. The works in [14] proposes an algorithm to efficiently compute the maximum coverage representative skyline which aims to select solutions that maximize the area of dominant solutions (dominance area). The work in [20] proposes a dominance power - inverse dominance power metric which favors solutions that dominate more genuine points in the sense that they are dominated by fewer points.

In contrast to related work, the approach proposed in this work tries to explore the solution space of execution schedules in an efficient way, starting from homogeneous configurations which are then modified using different VM types in order to generate additional solutions and further improve the obtained skyline. The proposed approach assesses the impact of the changes (VM updating) on the execution of the whole dataflow. The solutions are pruned based on a novel metric that considers the sharper areas of the skyline as more important, favoring points at

Figure 1: Homogeneous vs Heterogeneous Scheduling.



the knees of the curve. The method further identifies a number of solutions at the flatter areas to provide a richer and more disperse skyline.

3 Problem Description

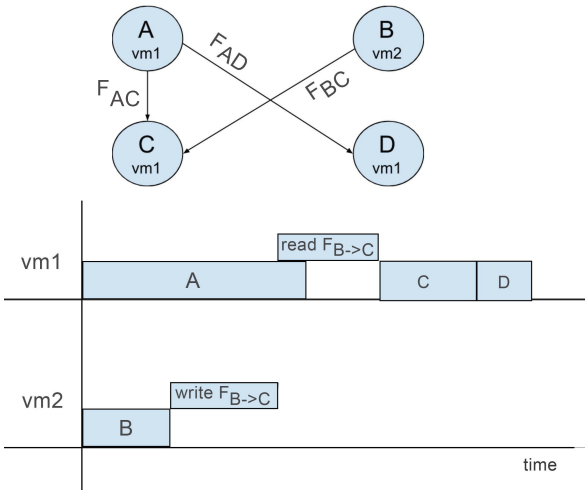
The problem considered in this work is determining the number and type of VMs to provision for the execution of dataflows on a cloud computing platform with the aim to optimize both application performance and the related monetary costs. Although a single VM type can be used for the execution of a dataflow, heterogeneous configurations may result in solutions with lower execution time, monetary costs and richer trade-offs. To illustrate this, Figure 1 shows the different solutions achieved for the execution of an example Montage dataflow of 100 operators when a slow VM type, a faster VM type or multiple VM types are available to use.

Assigning dataflow operators to different VM slots leads to a set of (partial) solutions which differ in execution time and cost. Some solutions may be dominated by others in the sense that lower execution time and monetary cost are achieved. Following the principle of pareto dominance, more efficient solutions can be identified. The set of non-dominated solutions comprises a *pareto front* (*skyline*). The skyline solutions can be presented to users to manually select the option that better fits to their requirements. Alternatively, the best trade-off solution may be automatically selected based on predefined user preferences e.g. weights that show the importance of different objectives [19].

3.1 Applications used

The paper considers dataflow queries modeled as a Directed Acyclic Graph (DAG). The nodes represent operators (computations) and the edges the flows of data transfers between them. An operator can only start after the execution of its predecessors completes. It is assumed that each operator is store-and-forward where all the input data

Figure 2: Operator Execution.



of an operator must be available before its execution. The runtime of operator op at vm type vm is given by:

$$runtime_{op} = \frac{size_{op}}{speed_{vm}}, \quad (1)$$

where the length of the operator $size_{op}$ is given in millions instructions and the processing capability of the VM type used $speed_{vm}$ is given in millions instructions per second.

Cloud Model Cloud providers may offer heterogeneous VM types with different characteristics in terms of performance and price. A model similar to Amazon Elastic Compute Cloud (EC2) is assumed with VMs being provisioned on demand. The user is charged for each VM instance used by time unit (quantum), such as in an hourly billing basis. A VM is considered to be active during a time quantum when one or more operators run on it. Partial time units are rounded to the full time units. For example, a VM is charged as a full hour even if it is only used for several minutes.

In this work, we only consider the computing service, assuming that the storage service is accessible from each VM at the same cost (network bandwidth). Also, it is assumed that a shared cloud storage system, such as Amazon S3, is used to store and retrieve data. Input data required for the execution of an operator are retrieved if needed and stored locally at the assigned VM with data transfer between operators that run on the same VM being 0. After the execution of an operator, its output data are stored to the storage system used. The execution of an operator may overlap with data transfers from/to the storage service. However, operators do not run concurrently on a VM, but have exclusive access to it.

Figure 2 includes a simple example with four operators (A, B, C and D) running at two VMs to show how data

transfers (files F_{AD}, F_{AC}, F_{BC}) occur. It can be seen that file F_{BC} is stored at VM_2 after the execution of operator B finishes and retrieved at VM_1 before the execution of operator C starts. The data transfer of the file overlaps with the execution of operator A at VM_1. On the other hand, communication cost from A to C or D is considered to be 0, as the operators are assigned to the same VM.

4 Algorithm Description

In this section a bi-objective scheduling algorithm that iteratively computes the skyline of potential solutions (schedules) for the mapping of dataflow operators to suitable VMs is described. The output of the algorithm is a set of non-dominated solutions that achieve different trade-offs between execution time and monetary cost. In that way, users can determine the number and type of VMs required in order to achieve configurations that better fit their preferences.

The approach proposed in this paper is based on the idea that the mapping of dataflow operators to VMs and the provisioning of proper VM types can be considered as two complementary problems to be addressed. The algorithm works in two stages. It initially tries to explore homogeneous configurations close to the optimal pareto front for each available VM type (Section 4.1), assigning operators to idle slots. In the second stage (Section 4.2), the algorithm combines the pareto-efficient solutions identified for each VM type. Based on their unified skyline, heterogeneous configurations with proper VM characteristics are then explored to create additional schedules which can improve the skyline. The algorithm iteratively upgrades or degrades the type of VMs at each schedule, taking into account the slack time of dataflow operators, in order to generate and add new solutions when the skyline can be improved. Decision making is based on the impact of the changes on the whole dataflow schedule, gradually exploring heterogeneous configurations with lower execution time or monetary cost.

4.1 Computation of Homogeneous Skyline Stage

In the first stage, shown in Algorithm 1, the algorithm initially assigns weights (ranks) to the operators to determine the order in which they are scheduled to the VMs. The approach used to order the operators is described in Section 4.1.1. Then, the algorithm iteratively builds the skyline of the solutions (schedules) for each VM type separately and keeps their unified skyline. To do so, the algorithm builds the skyline of solutions for a single VM type by adding at each step the next ready operator to be assigned to all the possible available slots at each partial solution. These include all the idle slots in the existing resources (VMs) that meet the data dependency constraints (supporting an insertion-based policy to fill idle slots between operators) but also potential slots at newly created resources (lines 7-11). At each step the partial schedules built are stored in order to compute the current pareto front (line 14). When the number of solutions is large only the k most representative solutions are kept using the pruning policy described in Section 4.1.2. Also, the algorithm assigns a parameter for each schedule (plan), $vmUpgrading$, depending on the VM type assigned which indicates whether the type of each VM used at the plan can be later upgraded/degraded or both (line 7). Depending on the parameter values (ascending, descending or ascending/descending), the algorithm explores heterogeneous configurations in the next stage by upgrading and/or degrading the type of the VMs used at each schedule accordingly.

4.1.1 Operator Ordering

Dataflow operators can be divided into levels (stages) according to data dependencies between them. Also, several operators may be more critical than others. Mean slack on different VM types, the time the execution of an operator can be delayed without extending its critical path, is used to indicate the criticality of an operator. The operators are ranked based on their level and mean slack so that data dependencies are met and operators with smaller slack (more critical operators) are prioritized like [2, 22]. The algorithm assigns the operators to available slots using a top-down approach with each operator being ready for scheduling when all of its predecessors are already assigned. When more than one operator is ready for scheduling, the operator with the highest ranking is selected.

To do so, the operators are divided into levels based on their longest path from an entry node of the dataflow. The level of each operator is computed as the maximum level of its predecessors, $preds_{op}$, increased by 1:

$$level_{op} = max_{p \in preds_{op}} level_p + 1, \quad (2)$$

with the level of each entry node (operators without any predecessors) being 0.

For operators of the same level, operators with smaller slack are prioritized, by assigning weights based on the summation of their upward and downward ranking, the longest path from an exit and entry node, respectively. The upward rank is computed recursively starting from an exit node as:

$$urank_{op} = \overline{w}_{op} + max_{s \in succs_{op}} (urank_p + comCost_{op \to s}), \quad (3)$$

where $comCost_{op \to s}$ is the communication cost between operators op and s. The upward rank of an exit node is

Algorithm 1 Homogeneous Skyline Computation Algorithm

```
1: procedure GENERATEHOMOSKYLINE(vmType)
2:     rankedOps: operators sorted based on their topological level and mean slack
3:     while readyOps ≠ ∅ do
4:         readyOps: operators with no data dependencies
5:         op ← operator with the highest ranking from readyOps
                                  ▷ Between operators of the same level, more critical operators are prioritized
6:         for p ∈ skylinePlans do
7:             candPlans ← allocateNewVM(vmType, vmUpgrading)
8:             for ∀vm ∈ p do
9:                 p' ← p + assign(op, vm)
10:                candPlans ← candPlans + p'
11:            end for
12:        end for
13:        Sort candPlans based on time, money and average VM utilization
14:        skylinePlans ← computeAndPruneSkyline(candPlans)
15:    end while
16:    return skylinePlans
17: end procedure
```

equal to its mean execution time $\overline{w}_{op}$. The downward rank is computed recursively starting from an entry node as:

$$drank_{op} = \overline{w}_{op} + max_{p \in preds_{op}}(urank_p + comCost_{p \to op}), \quad (4)$$

where $\overline{w}_{op}$ is the mean execution time of operator op at each VM type [18]. The downward rank of an entry node is equal to its mean execution time. The weight (rank) of operator op is then given by its mean slack:

$$rank_{op} = urankop + drankop - \overline{w}_{op}. \quad (5)$$

4.1.2 Skyline Pruning

The number of partial solutions generated at each step may be large and keeping all the possible schedules which result in an exhaustive search may be infeasible. To reduce the space of solutions, at each step the skyline of the partial solutions is computed keeping only solutions that are not dominated by any other solution. When two solutions do not differ in any of the targeted objectives (i.e. execution time and monetary cost), the schedule with the highest resource utilization is kept. The skyline of the partial solutions is further pruned when the number of skyline points is large (more than the predefined parameter k), selecting the most representative ones.

Pruning is based on the discrete second derivative, θ''_{sp}, of each skyline point sp from its adjacent skyline points and its euclidean distance, $dist_{sp,kn}$, from its nearest knee kn of the pareto curve. The discrete second order derivative for a skyline point sp is approximated as the difference between the first order partial derivative of sp and its adjacent left and right skyline points, p_l and p_r, respectively:

$$\theta''_{sp} = |\theta_l - \theta_r|, \quad (6)$$

With the term *knee* we refer to skyline points which differ significantly from solutions close to them offering more interesting trade-offs. These solutions are considered to be important as high savings in terms of time or money can be achieved. A skyline point is considered to be a knee if its discrete second derivative is more than or equal to the mean discrete second derivative of all the skyline points. In each step the two extreme skyline points which correspond to plans with minimum time or money at the pareto front are kept and $k-2$ solutions between the two extreme points are selected based on the scoring function of Equation 7:

$$score_{sp} = \theta''_{sp} \cdot dist_{sp,kn}. \quad (7)$$

The idea is that skyline points at the knees of the pareto curve have a higher probability of being selected as representative points. Such points can result in high savings compared with other solutions. For example, a significant decrease on execution time may be achieved with a small increase in the monetary cost. Additionally, points that are distant from the knees have a higher probability to be kept. In that way, the representative skyline may cover a wider area of solutions.

4.2 Computation of Heterogeneous Skyline Stage

The second stage of the algorithm is shown in Algorithm 2. After generating the unified skyline from homogeneous configurations (line 7), the algorithm tries to ex-

Algorithm 2 Heterogeneous Skyline Computation Algorithm

```
1: procedure GENERATEHETEROSKYLINE(paretoPlans, upgrading)
2:     rankedOps: operators sorted based on their topological level and mean slack time
3:     paretoPerVMType ← ∅
4:     for vmType ∈ vmTypesAvailalble do
5:         paretoVMType ← paretoVMType + generateHomoSkyline(vmType)
6:     end for
7:     solutions ← computeSkyline(paretoVMType, vmUpgrading)
                                   ▷ vmUpgrading: ascending and/or descending depending on vmType
8:     for plan ∈ solutions do
9:         if plan.vmUpgrading ≡ ascending/descending then
10:            Create two copies of the plan where vmUpgrading ← ascending, descending, resp.
11:        end if
12:    end for
13:    plansToUpgrade ← solutions
14:    for oldPlan ∈ plansToUpgrade do
15:        for op ∈ p do
16:            computeSlackTime(op)
17:        end for
18:        for vm ∈ p do
19:            Compute the average slack time of vm, avgSlack_vm
20:        end for
21:        vmSorted ← sort vms by avgSlack_vm ascending/descending based on oldPlan.vmUpgrading
22:        while vmSorted ≠ ∅ do
23:            vmToUpgrade ← vm with the largest/smallest slack from vmSorted
24:            newPlan ← getnextVMType(vmToUpgrade), update schedule
25:            if getCost(newPlan) > getCost(oldPlan)&&getTime(newPlan) > getTime(oldPlan) then
26:                break
27:            else
28:                modifiedPlans ← modifiedPlans + newPlan
29:            end if
30:        end while
31:    end for
32:    paretoPlans ← computeAndPruneSkyline(paretoPlans + modifiedPlans)
33:    plansToUpgrade ← each plan in modifiedPlans belonging to paretoPlans
34: end procedure
```

plore heterogeneous configurations with different trade-offs in the pareto front (skyline). For plans where the type of the VMs can be either upgraded or degraded (indicated using the value *ascending*/*descending* for parameter *vmUpgrading*), the algorithm keeps two copies of the plan in the skyline and updates the parameter *vmUpgrading* for each plan to *ascending* and *descending*, respectively, so that plans with heterogeneous configurations (using larger VMs or smaller VMs accordingly) will be explored.

The procedure followed for each plan to be upgraded is described next. The slack time of each operator at the plan, which indicates the delay the execution time of an operator can be stretched without extending overall dataflow execution time, is computed as:

$$slackTime_{op} = lst_{op} - est_{op}. \tag{8}$$

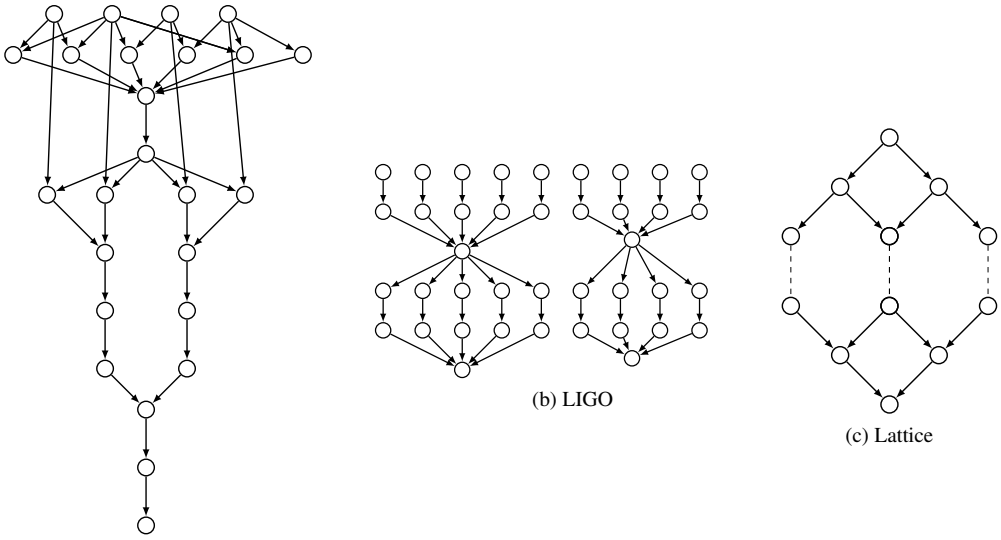
The earliest start time of each operator op computed recursively is given by Equation 9:

$$est_{op} = max_{p \in preds_{op}}(est_p + runtime_p + comCost_{p \to op}), \tag{9}$$

The predecessors $preds_{op}$ include all the predecessors in the DAG and the VM assigned. The latest start time is given by Equation 10:

$$lst_{op} = min_{s \in succs_{op}}(lst_s - comCost_{op \to s} - runtime_{op}). \tag{10}$$

The successors $succs_{op}$ of the operator considered include both the successors in the DAG and the successor at the VM assigned. Then, for each VM of the plan the mean slack time of the operators assigned to the VM is computed and the VMs are sorted accordingly (line 21). The VM with the largest/smallest mean slack time (for descending/ascending *vmUpgrading*), is selected to be degraded or upgraded, respectively. The slots of the operators are updated and the new plan is generated. The monetary cost and execution time for the new plan are com-



(a) Montage (b) LIGO (c) Lattice

Figure 3: The structures of dataflows used.

puted and the plan is kept if the monetary cost or the execution time is reduced (line 28). Otherwise, the plan is rejected and the algorithm continues with the next plan in the list (line 26). After updating all the plans in the list, the algorithm computes (and prunes if needed) the new skyline rejecting dominated (or not representative) solutions (line 32). The same procedure continues for the newly created plans kept at the representative skyline with the aim to further upgrade/degrade the VMs at the next iteration and generate new solutions. The algorithm terminates and returns the latest computed skyline when there are no new solutions kept at the skyline or the smallest/largest VM type is reached for every plan.

5 Experimental Evaluation

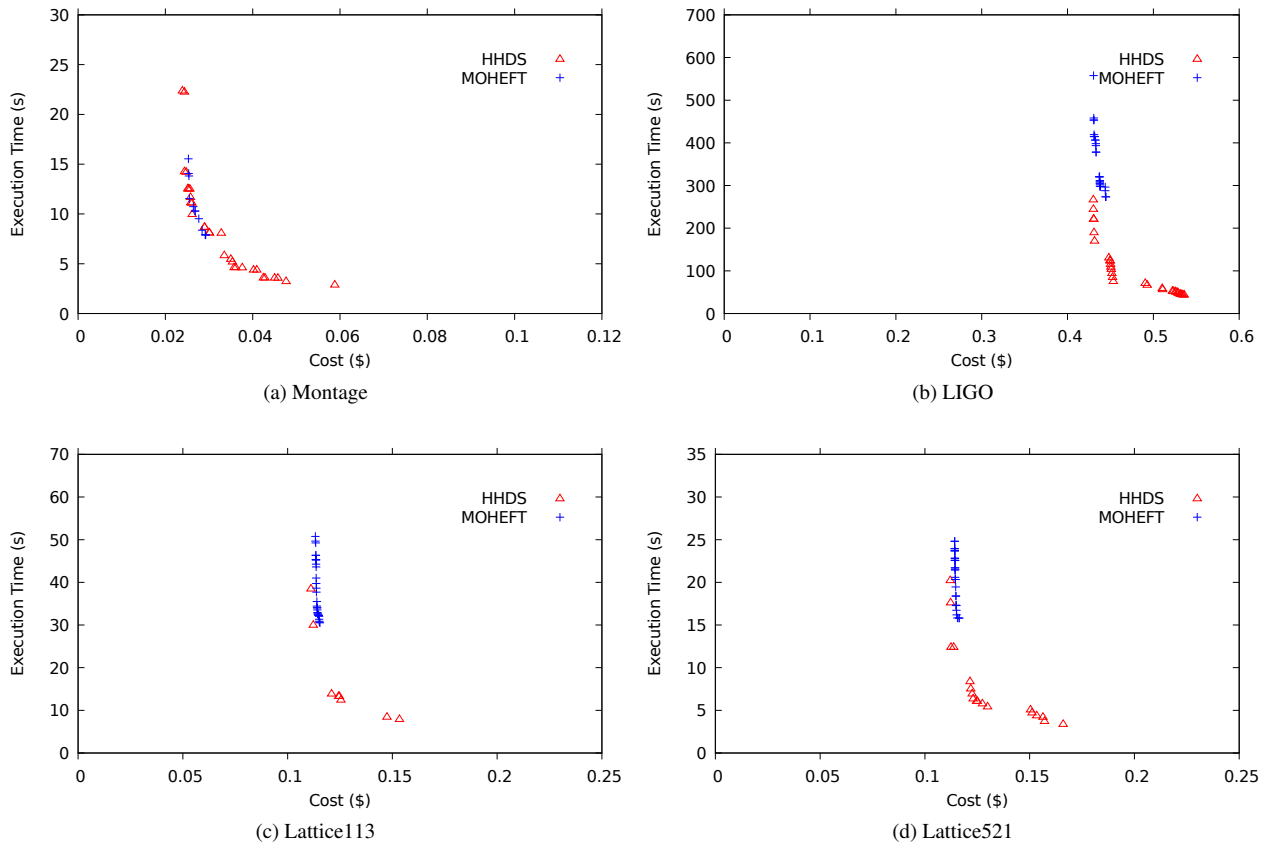
In this section, the efficiency of the proposed algorithm is evaluated and compared with the state of the art using three different dataflows. The dataflow characteristics and cloud model parameters used are described below.

5.1 Methodology

The ADP dataflow simulator [11] incorporated at Exareme [7], a distributed dataflow processing system, was modified and used to implement and validate the proposed algorithm. MOHEFT [6] is used as the baseline approach to evaluate the efficiency of the proposed algorithm. CloudHarmony [5] benchmarks are used for the performance and price parameters of the cloud model, selecting five Amazon EC2 instances with the same RAM per CPU ratio; these include the m1.small, m1.large, m2.xlarge, m2.2xlarge and m2.4xlarge VM instances. The CloudHarmony Compute Units (CCUs) are used to model the execution time of each operator at the different VM types. Also, experiments with different time units (per second or per hour pricing) are run. Finally, a network of 1Gbps is assumed to compute the communication cost between operators assigned to different VMs and parameter k, the number of solutions kept at each pruning step, is set to 30 unless stated otherwise.

Synthetic data of three families of dataflows, namely Montage [10], LIGO [13] and Lattice [11], are used to evaluate the algorithm based on realistic data. Montage and LIGO are real scientific applications; Montage is used to generate custom mosaics of the sky, while LIGO is used to analyze galactic binary systems for the detection of gravitational waves in the universe. Dataflows of 100 operators are generated using the workflow generator in [21]. In the case of the hourly pricing, multipliers are used for the operator execution times and data sizes to generate dataflows with longer execution times (order of hours). Lattice is a synthetic dataflow originally used in [11]. Two Lattice dataflows with height and branching factor pairs of 3-11 and 5-21, respectively (485 operators each) are used similarly to [11]. The structure of the dataflows is shown in Figure 3. As can be seen, several levels of Montage include bottleneck operators which collect or distribute data from a large number of parallel operators. The levels with bottlenecks and high parallelism in LIGO are more distributed. Finally, Lattice dataflows are used to model typical Map-Reduce dataflows.

Two similarity measures, the *Jaccard distance* [16] and the *skyline distance*, are used to evaluate the quality of the solutions obtained and compare the different skylines.



(a) Montage (b) LIGO

(c) Lattice113 (d) Lattice521

Figure 4: Comparison of the algorithms for the case of per second pricing.

The metrics try to capture the similarity between the skyline obtained from each algorithm and the common skyline S_c resulted by combining the individual skylines of the two compared algorithms. The Jaccard index is computed as:

$$jindex_a = \frac{|S_c \cap S_a|}{|S_c \cup S_a|}, \tag{11}$$

where $|S_c \cap S_a|$ is the number of plans (schedules) in the intersection between the common skyline and the skyline of the evaluated algorithm a, while $|S_c \cup S_a|$ is the number of plans in their union. The Jaccard distance is the subtraction of the Jaccard Index from 1:

$$jdist_a = 1 - jindex_a, \tag{12}$$

The skyline distance for each algorithm is computed as the sum of the euclidean distance of each plan in the skyline obtained from the algorithm to the common skyline:

$$dSet_a = \sum^{p \in S_a} dist_{p,S_a}. \tag{13}$$

We also compare the fastest and cheapest solutions identified in the different skylines.

5.1.1 Quality of Skyline

Per second pricing In this experiment, we compare the efficiency of the proposed algorithm, HHDS, with the baseline algorithm, MOHEFT, for heterogeneous resources priced by second. The results for Montage, LIGO and the two Lattice dataflows are presented in Figure 4. There are a few points where MOHEFT is slightly better than HHDS and the opposite happens in other cases. Apart from the case of Montage, where both algorithms identify solutions at the combined pareto front, the skyline obtained by HHDS is more disperse, identifying solutions with cheaper or faster configurations. Overall, HHDS results in solutions with a smaller number of VM types compared to MOHEFT. This is due to the fact that the algorithm starts with homogeneous configurations and updates the VM types assigned assessing the impact of the changes on the whole schedule.

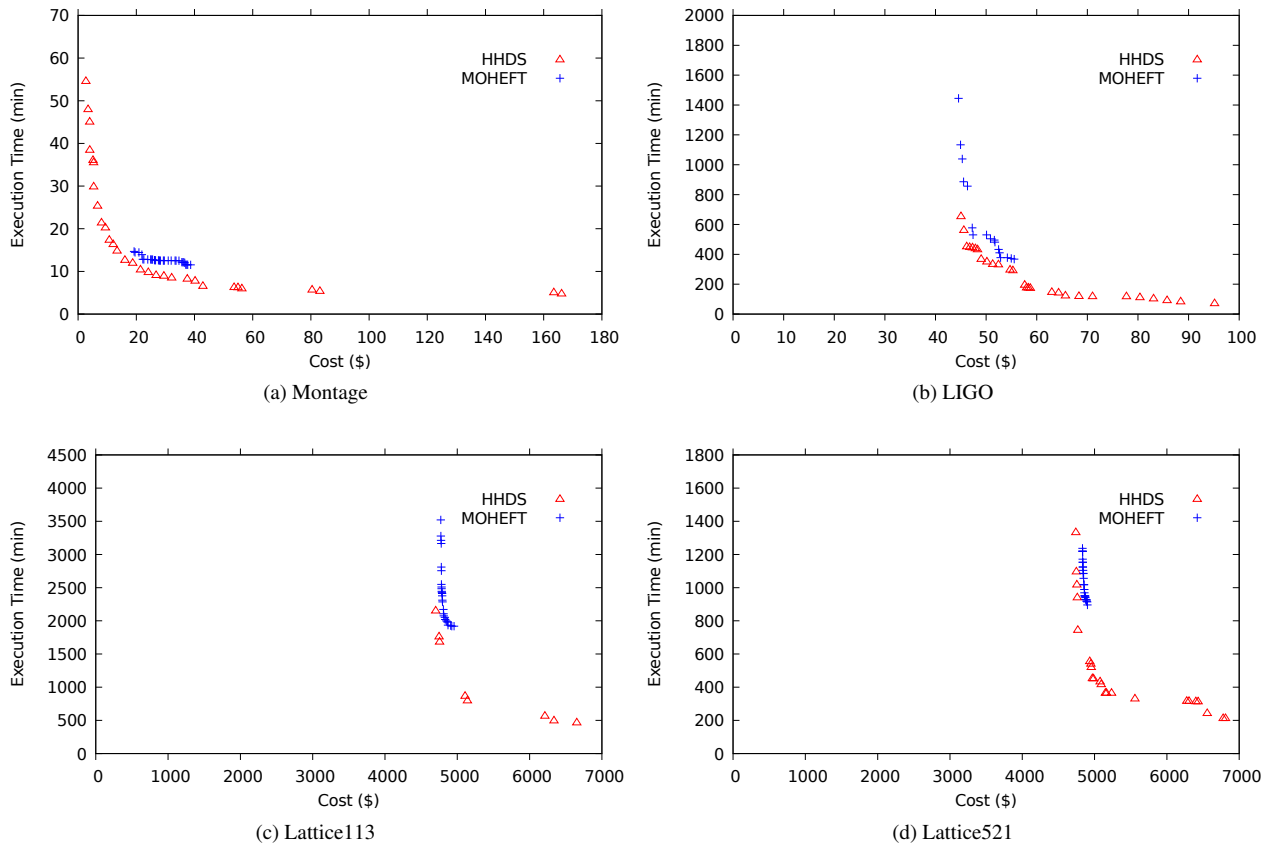
Table 1 shows the results for the metrics used. It can be seen that the Jaccard distance is significantly smaller for the proposed algorithm HHDS. The distance set metric is not significantly different as the distance between the skylines and the combined skyline is small. Also, the fastest

Table 1: Metrics for the comparison of the skyline quality for per second pricing.

Metric	Montage		LIGO		LATTICE 11-3		LATTICE 5-21	
	HHDS	MOHEFT	HHDS	MOHEFT	HHDS	MOHEFT	HHDS	MOHEFT
jDist	0.36	0.829	0.0	1.0	0.0	1.0	0.0	1.0
dSet	1.76e-10	0.0	0.0	5.62e-09	0.0	0.0	0.0	2.24e-08
Fast(s)	2.9	7.9	44.2	273.4	8.0	30.5	3.4	15.8
Cheap($)	86.02	91.01	1548.12	1549.18	400.14	407.67	403.56	411.15

and cheapest solutions obtained by the baseline algorithm are generally less efficient than the proposed algorithm HHDS. More specifically, for Montage, the fastest solution identified by MOHEFT is 2.724 times the value of HHDS. The cost of the cheapest solution is slightly higher (1.058 times the cost obtained by HHDS). Similarly, for LIGO the fastest solution identified by MOHEFT is 5.083 times the value of HHDS while the cost of the cheapest solution is almost equal for the two algorithms (1.00068 times the value obtained by HHDS). For Lattice11-3, the fastest solution identified by MOHEFT is 3.83 times the value of HHDS, while the cost of the cheapest solution is 1.019 times the value of HHDS. For Lattice5-21, the fastest solution identified by MOHEFT is 4.639 times the value of HHDS, while the cost of the cheapest solution is 1.019 times the value of HHDS.

Hourly pricing In this experiment, we compare the efficiency of the proposed algorithm, HHDS, with the baseline algorithm, MOHEFT, for heterogeneous resources priced by hour. As mentioned earlier, the operator execution times and data sizes are scaled up by 100 and 100, respectively to generate longer dataflows. The obtained results are presented in Figure 5. It can be seen that HHDS



(a) Montage (b) LIGO

(c) Lattice113 (d) Lattice521

Figure 5: Comparison of the algorithms for the case of hourly pricing.

Table 2: Metrics for the comparison of the skyline quality for hourly pricing.

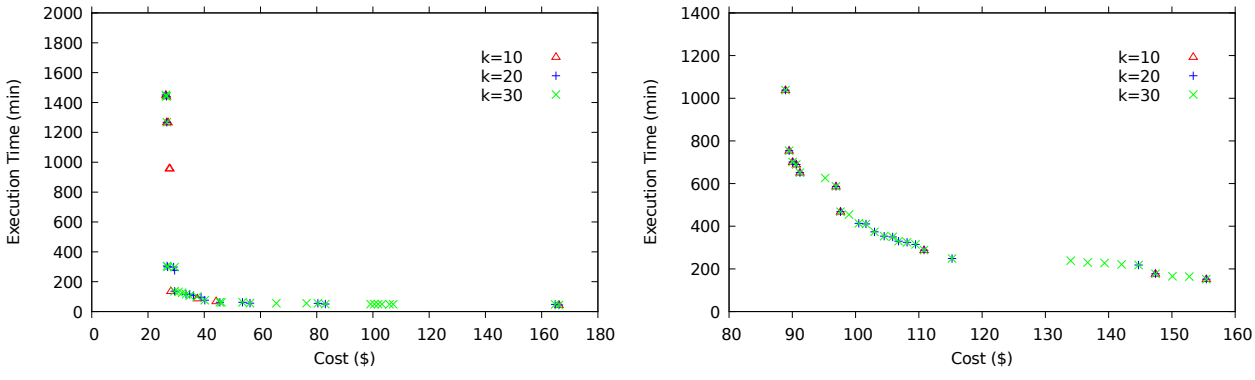
Metric	Montage		LIGO		LATTICE 11-3		LATTICE 5-21	
	HHDS	MOHEFT	HHDS	MOHEFT	HHDS	MOHEFT	HHDS	MOHEFT
jDist	0.0	1.0	0.067	0.956	0.0	1.0	0.0	1.0
dSet	0.0	1.493	0.0	0.022	0.0	14.0	0.0	9.0
Fast(s)	288.7	692.1	4344.2	22081.3	28091.3	115181.0	12830.2	53739.9
Cheap($)	2.68	19.24	45.03	44.56	4700.82	4771.35	4741.06	4832.82

outperforms MOHEFT in all cases. A richer and more diverse skyline is obtained for Montage and LIGO, while in the case of the Lattice dataflows the number of pareto solutions is limited.

Table 2 shows the results for the metrics used. It can be seen that both the Jaccard distance and the distance set metrics are significantly smaller for the proposed algorithm HHDS. For Montage, the fastest solution identified by MOHEFT is 2.397 times the value of HHDS, while the cost of the cheapest solution is 7.179 times the value of HHDS. For LIGO, the fastest solution identified by MOHEFT is 5.083 times the value of HHDS, while the cost of the cheapest solution is slightly better compared to the solution of HHDS (0.99 times the value of HHDS). For Lattice11-3, the fastest solution identified by MOHEFT is 4.1 times the value of HHDS, while the cost of the cheapest solution is very close to the value of HHDS (1.015 times the value of HHDS). For Lattice5-21, the fastest solution identified by MOHEFT is 4.188 times the value of HHDS, while the cost of the cheapest solution is only 1.019 times the value of HHDS. Finally, the overhead imposed by our scheduler, the time required to generate the final skyline compared to the time required for the execution of the fastest plan is less than 6% (less than 1% in most cases).

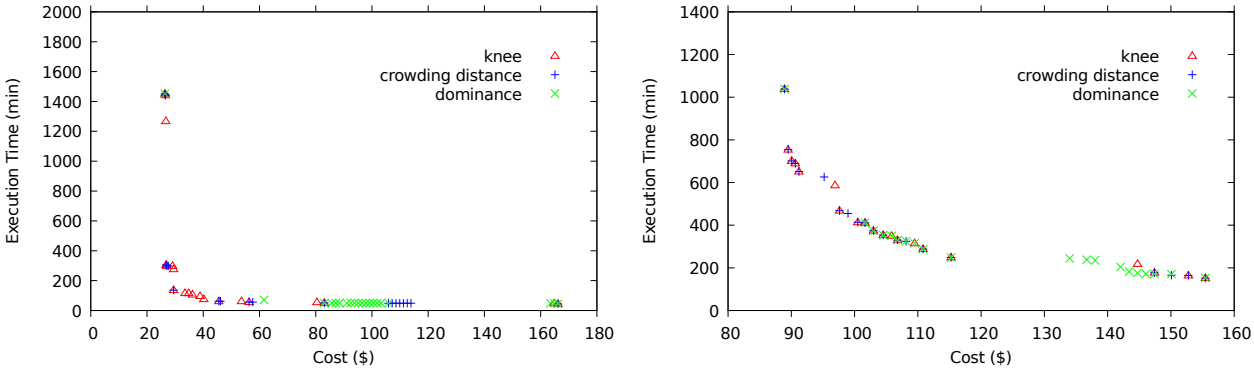
5.2 Varying the number of representative skyline points

In this experiment, we compare the efficiency of the proposed algorithm when varying the number of representative points at the pareto front; runs with k set equal to 10, 20 and 30, respectively, are used. Per hour pricing is applied. The results for the two real scientific dataflows, LIGO and Montage, where the number of solutions is quite large, are presented in Figure 6. In the case of the Lattices dataflows the skyline consists of a smaller number of points (8 and 23 solutions for Lattice 11-3 and 5-21, respectively). The execution time and data sizes of the dataflows are scaled up (by 200, 200 for LIGO and 1000, 300 for Montage) to generate longer data-intensive dataflows. In the case of LIGO all the solutions at the final skyline are kept for $k = 30$ as 28 pareto-efficient schedules are developed. It can be seen that by increasing k a larger number of solutions is kept at the resulted pareto front; however, the efficiency of the algorithm is not affected greatly for different values of k. Overall, Montage is more sensitive to parameter k. For example, there are two solutions for $k = 10$ which are not identified in the cases of $k = 20, 30$. Although it would be expected that keeping a larger number of solutions increases the quality of the skyline, this may not always be the case. Overall,



(a) Montage (b) LIGO

Figure 6: Different k values.



(a) Montage (b) LIGO

Figure 7: Different pruning methods used.

different dataflows are not equally sensitive to the number of representative skyline solutions kept.

5.3 Comparison of Pruning Methods

In this experiment, we compare the efficiency of the pruning metric proposed in this paper, the knee-based metric, with two state of the art metrics, the crowding distance [6] and the dominance area [14]. Figure 7 shows the results for the Montage and LIGO dataflows used for the case of per hour pricing. The execution times and data sizes are scaled up (by 200, 200 for LIGO and 1000, 300 for Montage). The results for 20 solutions are presented since pareto solutions are 28 (less than 30) for LIGO and pruning can only be applied for smaller values of k.

Crowding distance based pruning [6] tries to evenly distribute the solutions across the skyline selecting points which have a larger surrounding area with no other pareto-efficient solutions. It can be seen that using the crowding distance may result on keeping fewer solutions at the knees of the skyline, where divergent trade-offs may occur. Also, a larger number of solutions is kept while moving to faster but more expensive plans in the case of Montage. This may be due to the fact that the values of the objectives are not normalized.

In the case of the dominance-based pruning, the aim is to select points which dominate a larger number of solutions but also their dominated areas differ so that the number of non skyline points dominated by the representative skyline is maximized. However, uneven distribution of solutions in the search space may lead to uneven distribution of skyline points. It can be seen that the skyline includes a large number of expensive plans. This may be due to the large concentration of solutions at that point.

On the other hand, knee-based pruning aims at identifying solutions close to the knees of the dataflow where more interesting trade-off solutions may appear. While moving to the extreme skyline points (minimum time or money), fewer solutions are selected as they may not differ significantly in terms of money or cost. Also, the pruning method finds an expensive solution with slightly longer execution time compared to the solution identified by the dominance-based pruning. This may be due to the fact that the dominance-based pruning keeps more solutions at the surrounding area and efficiency is improved.

5.4 Discussion

The pruning method used for pareto-based multi-objective approaches may affect the efficiency of the algorithm depending on the dataflow. Most pruning methods try to identify the most representative points at the skyline based on several (usually distance or dominance based) criteria. However, the number and space of partial solutions generated may differ between different iterations. For example, the number of solutions gets significantly larger for later iterations. Also, the selection of good solutions may not be equally important for the assignment of different operators. For example different assignments of more critical operators may lead to more diverse schedules. Hence, determining the number and quality of solutions required at each iteration based on the dataflow characteristics or the distribution of the solution space may be an interesting direction for future work. Additionally, in several scenarios taking into account dominated partial solutions may improve the skyline. The pruning methods examined in this work only consider skyline solutions. Identifying important non-dominated solutions which could lead to better execution schedules could be another challenging direction.

6 Conclusion

This work considered the problem of money-time tradeoff optimization for dataflow scheduling on the Cloud, exploiting resource heterogeneity to map the dataflow operators to proper VMs. The algorithm identifies a set of pareto-efficient execution schedules, while using a novel pruning method to select representative solutions when the number of schedules is very large. The experimental evaluation shows that, in several cases, the proposed approach can lead to a better and more diverse set of tradeoff solutions.

Future work could try to improve the algorithm from pruning partial execution schedules by identifying and including dominant yet important solutions that may improve the pareto front. Also, the algorithm can be extended for the execution of multiple dataflows (ensembles); preliminary results are promising and provide schedules with good utilization. Finally, the performance of the proposed algorithm could be evaluated on the Exareme platform on real scenarios.

References

[1] K. Bessai, S. Youcef, A. Oulamara, C. Godart, and S. Nurcan. Bi-criteria workflow tasks allocation and scheduling in cloud computing environments. In *Proceedings of the 5th IEEE International Conference on Cloud Computing (CLOUD)*, pages 638–645. IEEE, 2012.

[2] D. Bozdag, U. Catalyurek, and F. Ozguner. A task duplication based bottom-up scheduling algorithm for heterogeneous environments. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 12–pp. IEEE, 2006.

[3] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng. Cost optimized provisioning of elastic resources for application workflows. *Future Generation Computer Systems*, 27(8):1011–1026, 2011.

[4] L.-C. Canon et al. Mo-greedy: an extended beam-search approach for solving a multi-criteria scheduling problem on heterogeneous machines. In *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, pages 57–69. IEEE, 2011.

[5] CloudHarmony. `https://cloudharmony.com/`.

[6] J. J. Durillo, H. M. Fard, and R. Prodan. Moheft: A multi-objective list-based method for workflow scheduling. In *Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 185–192. IEEE, 2012.

[7] Exareme, MADGiK group, University of Athens. `"http://madgik.github.io/exareme/`.

[8] S. Ganguly, W. Hasan, and R. Krishnamurthy. *Query optimization for parallel execution*, volume 21. ACM, 1992.

[9] T. T. Huu and J. Montagnat. Virtual resources allocation for workflow-based applications distribution on a cloud infrastructure. In *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pages 612–617. IEEE, 2010.

[10] D. S. Katz, J. C. Jacob, E. Deelman, C. Kesselman, G. Singh, M.-H. Su, G. Berriman, J. Good, A. Laity, and T. A. Prince. A comparison of two methods for building astronomical image mosaics on a grid. In *Proceedings of the IEEE International Conference on Parallel Processing Workshops (ICPPW)*, pages 85–94. IEEE, 2005.

[11] H. Kllapi, E. Sitaridi, M. M. Tsangaris, and Y. Ioannidis. Schedule optimization for data processing flows on the cloud. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 289–300, 2011.

[12] J. Li, S. Su, X. Cheng, Q. Huang, and Z. Zhang. Cost-conscious scheduling for large graph processing in the cloud. In *Proceedings of the 13th IEEE International Conference on High Performance Computing and Communications (HPCC)*, pages 808–813. IEEE, 2011.

[13] LIGO project, LIGO laser interferometer gravitational wave observatory. `http://www.ligo.caltech.edu/`.

[14] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE)*, pages 86–95. IEEE, 2007.

[15] R. Prodan and M. Wieczorek. Bi-criteria scheduling of scientific grid workflows. *IEEE Transactions on Automation Science and Engineering*, 7(2):364–376, 2010.

[16] P. H. Sneath, R. R. Sokal, et al. *Numerical taxonomy. The principles and practice of numerical classification.* 1973.

[17] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, and J. Wang. Cost-efficient task scheduling for executing large programs in the cloud. *Parallel Computing*, 39(4):177–188, 2013.

[18] H. Topcuoglu, S. Hariri, and M.-y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

[19] I. Trummer and C. Koch. A fast randomized algorithm for multi-objective query optimization. In *Proceedings of the International Conference on Management of Data*, pages 1737–1752. ACM, 2016.

[20] G. Valkanas, A. N. Papadopoulos, and D. Gunopulos. Skyline ranking à la ir. In *Proceedings of the EDBT/ICDT Workshops*, pages 182–187, 2014.

[21] Workflow Generator. "`https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator`.

[22] H. Wu, X. Hua, Z. Li, and S. Ren. Resource and instance hour minimization for deadline constrained dag applications using computer clouds. *IEEE Transactions on Parallel and Distributed Systems*, 27(3):885–899, 2016.

[23] J. Yu, R. Buyya, and C. K. Tham. Cost-based scheduling of scientific workflow applications on utility grids. In *Proceedings of the 1st International Conference on e-Science and Grid Computing*, pages 8–pp. IEEE, 2005.

[24] F. Zhang, J. Cao, K. Li, S. U. Khan, and K. Hwang. Multi-objective scheduling of many tasks in cloud platforms. *Future Generation Computer Systems*, 37:309–320, 2014.