Multi-objective Optimization of Scheduling Dataflows on Heterogeneous Cloud Resources

Ilia Pietri*, Yannis Chronis*‡,Yannis Ioannidis*†
{ipietri, i.chronis, yannis}@di.uoa.gr
* Department of Informatics and Telecommunications, University of Athens, Greece
† ATHENA Research and Innovation Center, Greece

***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.**

***Index Terms*—multi-objective optimization; dataflow scheduling; heterogeneous clouds**

I. 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 platforms 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 the number 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.

In 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 (MOQO) 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). 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 remainder of the paper is organized as follows. Related work is included in Section 2. The problem description follows in 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.

II. Related Work

Dataflow scheduling and resource provisioning on distributed computing environments like cluster, grid and cloud computing is the focus of many studies which address different optimization goals [1]–[5]. Among these, execution time and monetary cost minimization has received considerable attention. Several algorithms focus on single-objective optimization or constrained single-objective optimization [3], [6]. Heterogeneous Earliest Finish Time (HEFT) algorithm [6] is a well known heuristic that maps tasks to heterogeneous resources selecting the slot with the earliest finish time. Other studies

‡Yannis Chronis is currently with University of Wisconsin Madison (chronis@cs.wisc.edu).

[1], [7] propose multi-objective approaches that address the optimization problem as a weighted single-objective problem.

The work proposed in this paper is more related to pareto-based approaches for the scheduling of dataflows [2], [5], [8] which generate a set of trade-off solutions when no single schedule that optimizes all the objectives exists. MOHEFT [5] is a heuristic that extends HEFT by keeping a set of partial solutions at each step instead of a single allocation considering heterogeneous resources. The skyline is pruned based on the crowding distance metric which tries to measure the surrounding area of each skyline point where no other skyline point exists. Pruning schemes to select a number of appropriate (representative) pareto-efficient solutions have been the focus of several studies [9], [10]. The algorithm in [9] aims to efficiently compute the maximum coverage representative skyline and select points that maximize the area of dominant solutions (dominance area). Finally, MOQO approaches like [11], [12] are complementary to our work, focusing on a different level of optimization for query plans (the join orders of the operators).

In contrast to related work, the approach proposed in this work tries to explore an important part of the solution space, addressing the optimization problems of resource provisioning and task assignment in separate steps. Starting from a set of homogeneous configurations the algorithm updates the VM types of the solutions assessing the impact of the changes on the execution of the whole dataflow in order to identify heterogeneous configurations that further improve the skyline. The pruning of the solutions is based on a novel metric that considers the sharper areas of the skyline as more important.

III. 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. 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. The diversity of solutions may be more profound when heterogeneous resources are considered; although a single VM type can be used for the execution of a dataflow, heterogeneous configurations may often result in solutions with lower execution time, monetary costs and richer trade-offs. Following the principle of pareto dominance [13], more efficient solutions can be identified. The set of non-dominated solutions comprises a *pareto front* (*skyline*). The final 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.

1) Application and Performance Model: The paper considers dataflow queries modeled as a Directed Acyclic Graph (DAG). The nodes represent operators (computations) and the edges 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 (all the input data of an operator must be available before its execution) [14]. The runtime of operator op at vm type vm is given by: $runtime_{op} = \frac{size_{op}}{speed_{vm}}$, where the length of the operator, $size_{op}$, is given in millions instructions and the processing capability (computational speed) of the VM type used, $speed_{vm}$, is given in millions instructions per second. For simplicity, in the model considered it is assumed that operator runtime is inversely proportional to the VM's computational speed. Although this assumption may hold for CPU-bound jobs, performance may vary for jobs with different resource characteristics. However, the performance model can be easily extended to incorporate modeling of performance variation for operators with different resource characteristics.

2) 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 and stored locally at the assigned VM. Communication cost between operators running on the same VM is considered 0 [14]. 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 [15].

IV. 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.

The approach proposed in this paper is based on the idea that the mapping of 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 IV-A), assigning operators to idle slots. In the second stage (Section IV-B), the algorithm combines the pareto-efficient solutions identified for each VM type. Based on their unified skyline, heterogeneous configurations are explored to create

schedules that improve the skyline. The algorithm iteratively upgrades or degrades the type of VMs at each schedule, taking into account the *slack time* of operators (the time the execution of an operator can be delayed without extending its critical path), 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 and/or monetary cost.

A. Computation of Homogeneous Skyline Stage

The algorithm initially assigns weights (ranks) to the operators to determine the order in which they are scheduled, as described in Section IV-A1. 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 (Alg. 1). These include all the idle slots in the existing resources (VMs) that meet the data dependency constraints but also slots at newly created resources (lines 6-9). At the end of each step the pareto front of the partial schedules is computed (line 12). When the number of solutions is large only the k most representative solutions are kept using the pruning method described in Section IV-A2. Also, the algorithm assigns a parameter for each schedule (plan), $vmUpgrading$, depending on the VM type used, which indicates whether the type of each VM used at the plan can be later upgraded/degraded or both, in the next stage (line 6).

1) Operator Ordering: Dataflow operators can be divided into levels (stages) according to the data dependencies between them. Mean slack on different VM types, computed based on the structure of the DAG, is used to indicate the criticality of an operator. The operators are ranked based on their mean slack and level, so that data dependencies are met, and operators with smaller slack (more critical operators) are prioritized like in [16]. 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 have already been assigned.

Algorithm 1 Homogeneous Skyline Computation Algorithm

```
1: procedure GENERATEHOMOSKYLINE(dag, vmType, k)
2:     readyOps ← operators in dag with no dependencies
3:     while readyOps ≠ ∅ do
4:         op ← ready.pollFirst()        ▷ operators sorted based on ranking
5:         for p ∈ skylinePlans do
6:             candPlans ← allocateNewVM(vmType, vmUpgrading)
7:             for ∀vm ∈ p do
8:                 p' ← p + assign(op, vm)
9:                 candPlans.add(p')
10:            end for
11:        end for
12:        skylinePlans ← getPrunedSkyline(candPlans)
13:        Update readyOps
14:    end while
15:    return skylinePlans
16: end procedure
```

To do so, the operators are divided into levels based on their longest path from an entry node of the dataflow (dag). The level of each operator is computed as the maximum level of its predecessors, $preds_{op}$, increased by 1 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 [15], the longest path from an exit and entry node, respectively. The upward rank is computed recursively starting from an exit node as:

$$uRank_{op} = \overline{w}_{op} + max_{s \in succs_{op}}(uRank_s + comCost_{op \to s}), \quad (1)$$

where $comCost_{op \to s}$ is the communication cost between operators op and s. The upward rank of an exit node is equal to its mean execution time $\overline{w}_{op}$. The downward rank is computed recursively starting from an entry node as:

$$dRank_{op} = max_{p \in preds_{op}}(\overline{w}_p + dRank_p + comCost_{p \to op}). \quad (2)$$

The weight (rank) of operator op is then given by Eq. 3:

$$rank_{op} = uRank_{op} + dRank_{op}. \quad (3)$$

An operator op is considered to be on the critical path if $rank_{op} = \max_{\forall i \in dag} rank_i$ [15]. Hence, larger weights indicate more critical operators (smaller slack).

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 and its euclidean distance, $dist_{sp,kn}$, from its nearest knee kn of the pareto curve, hence the name *knee-based pruning*. The discrete second order derivative, θ''_{sp}, is approximated as the difference between the first order partial derivative of sp and its adjacent left and right skyline points. With the term *knee* we refer to skyline points which differ significantly (in terms of time or money) from solutions close to them offering more interesting trade-offs. 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 Eq. 4 (knee-based metric):

$$score_{sp} = \theta''_{sp} \cdot dist_{sp,kn}, \quad (4)$$

The idea is that skyline points at the knees of the pareto curve have a higher probability of being selected as representative

points. Such solutions are considered to be important as they can result in high savings compared with other solutions. For example, a significant decrease in execution time may be achieved with a small increase in cost. Additionally, points that are distant from the knees have a higher probability to be kept. In that way, the representative skyline covers a wider area of solutions. Parameters θ''_{sp} and $dist_{sp,kn}$ of Eq. 4 are normalized by their maximum value; the maximum second derivative between all the skyline points and the maximum distance between the two extreme skyline points, respectively. Distance $dist_{sp,kn}$ is set to 1 for skyline points that are knees.

B. Computation of Heterogeneous Skyline Stage

After generating the unified skyline from homogeneous configurations (line 4 of Alg. 2), the algorithm tries to explore heterogeneous configurations with different trade-offs in the pareto front. For plans where the type of the VMs can be either upgraded or degraded (indicated using the value *ascending/descending*), 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 or smaller VMs accordingly) will be explored.

The procedure followed for each plan to be modified is described next. The slack time of each operator, which indicates the delay the execution time of an operator can be stretched without extending overall dataflow execution time, is computed as: $slackTime_{op} = lst_{op} - est_{op}$. In contrast to mean slack in Section IV-A1 which is independent of the assignments of the operators to VMs, the computation of the slack time at the plan is based on both the structure of the DAG and the assignments of the operators to the VMs. The earliest start time of each operator op is computed recursively as: $est_{op} = max_{p \in preds_{op}}(est_p + runtime_p + comCost_{p \to op})$. The predecessors $preds_{op}$ include all the predecessors in the DAG and the VM assigned. The latest start time of op is given by: $lst_{op} = min_{s \in succs_{op}}(lst_s - comCost_{op \to s} - runtime_{op})$. 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 it is computed and the VMs are sorted accordingly (line 8). The VM with the largest/smallest mean slack time is selected to be degraded/upgraded. The slots of the operators are updated and the new plan is generated. The monetary cost and the execution time for the new plan are computed and the plan is kept if the monetary cost and/or the execution time are reduced (line 16). Otherwise, the plan is rejected and the algorithm continues with the next plan in the list (line 14). After updating all the plans in the list, the algorithm computes (and prunes if needed) the new skyline rejecting dominated (non-representative) solutions (line 20). 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 final computed skyline when there are no new solutions kept at the skyline or the smallest/largest VM type is reached for every plan.

Algorithm 2 Heterogeneous Skyline Computation Algorithm

```
1: procedure GENERATEHETEROSKYLINE(dag, vmTypes, k)
2:     Rank operators at the dag                ▷ based on level and mean slack
3:     pareto.add(generateHomoSkyline(vmType)), ∀vmType ∈ vmTypes
4:     plansToUpgrade ← getPrunedSkyline(pareto)
                ▷ If vmUpgrading ≡ asc./desc. keep a copy for each value
5:     while plansToUpgrade ≠ ∅ do
6:         for plan ∈ plansToUpgrade do
7:             getSlackTime(op), ∀op ∈ p
8:             vmSorted ← sort VMs in plan based on their average slack
                                ▷ asc. or desc. order based on vmUpgrading
9:             while vmSorted ≠ ∅ do
10:                vmToUpgrade ← vmSorted.pollFirst()
11:                nextVMType ← getnextVMType(vmToUpgrade)
12:                newPlan ← update schedule of plan for nextVMType
13:                if newPlan.cost > plan.cost&&newPlan.time > plan.time
14:                    break
15:                else
16:                    modifiedPlans.add(newPlan)
17:                end if
18:            end while
19:        end for
20:        pareto ← getPrunedSkyline(pareto ∪ modifiedPlans, k)
21:        plansToUpgrade ← ∀plan ∈ modifiedPlans ∩ pareto
22:    end while
23:    Return pareto
24: end procedure
```

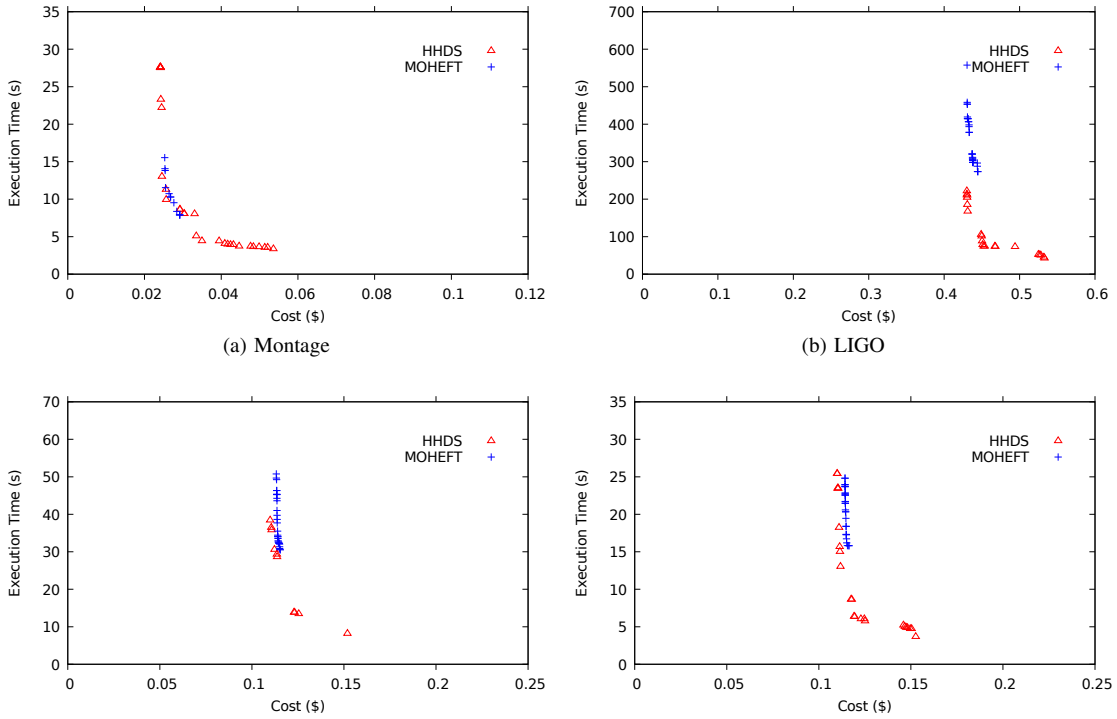
V. Experimental Evaluation

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

A. Methodology

The ADP simulator [14] of Exareme [17], a distributed dataflow processing system, was modified and used to implement and validate the proposed algorithm. MOHEFT [5] is used as the baseline approach to evaluate the efficiency of the proposed algorithm. Five Amazon EC2 instance types (m1.small, m1.large, m2.xlarge, m2.2xlarge and m2.4xlarge) that cover different instance families are used. The execution time of each operator at the different VM types is modeled based on the CloudHarmony Compute Units[1]. 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 (operator runtimes and data sizes) of three families of dataflows, namely Montage [18], LIGO [19] and Lattice [14], are used. Montage and LIGO are real scientific applications; Montage is used to generate image 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 [20]. Lattice is a synthetic

[1] https://cloudharmony.com/



(a) Montage (b) LIGO

(c) Lattice11-3 (d) Lattice5-21

Fig. 1: Comparison of the algorithms for the case of per second pricing.

dataflow originally used in [14] to model typical Map-Reduce applications. Two Lattice dataflows with height and branching factor pairs of 3-11 and 5-21, respectively (485 operators each) are used similarly to [14].

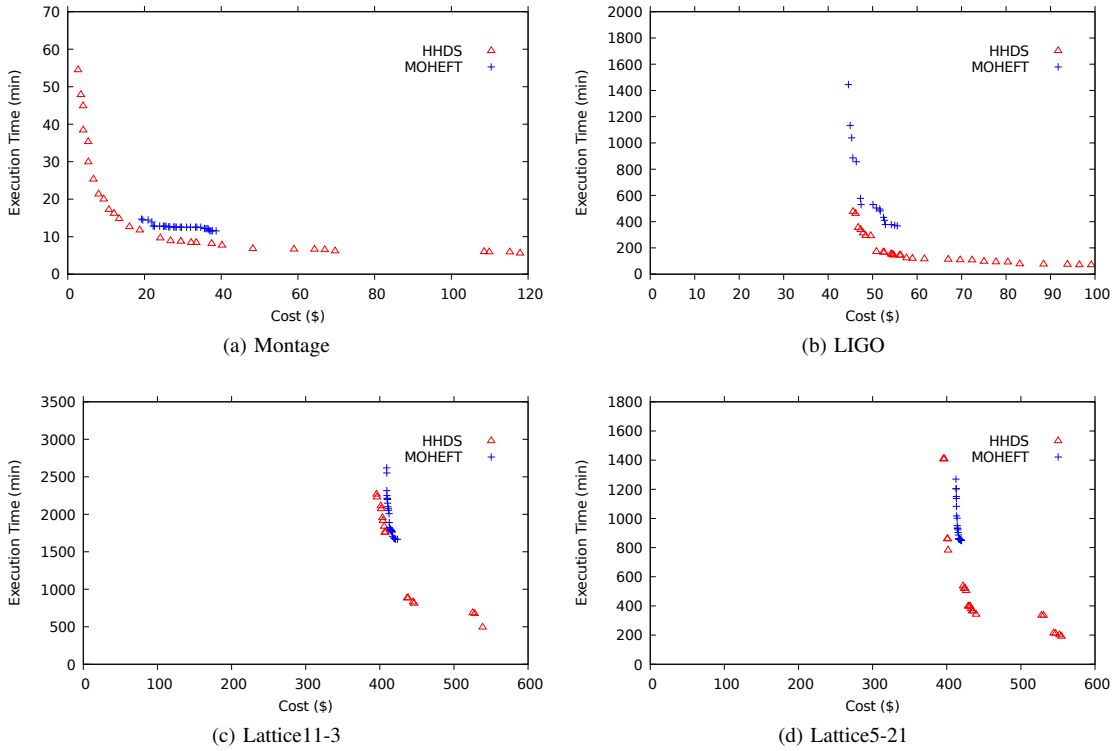
B. Quality of Skyline

1) Per second pricing: Initially, we compare the efficiency of the proposed algorithm, HHDS, with the baseline algorithm, MOHEFT, for heterogeneous resources priced by second. 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 (Fig. 1). Overall, HHDS results in plans with a smaller number of VM types compared to MOHEFT. This may be due to the fact that HHDS starts with homogeneous configurations and updates the VM types assessing the impact of the changes on the whole schedule.

The *Jaccard distance* [21] is used to quantitatively evaluate the quality of the solutions obtained and compare the different skylines. The metric tries to capture the similarity between the skyline obtained from each algorithm and the common skyline resulted by combining the individual skylines of the two compared algorithms. The Jaccard distance for each algorithm a, $jDist_a$, is the subtraction of the Jaccard index from 1, taking values between 0 (when the common skyline, S_c, and the skyline of the algorithm, S_a, are identical) and 1 (when algorithm a does not explore any solutions in S_c). The Jaccard index is computed as: $jIndex_a = \frac{|S_c \cap S_a|}{|S_c \cup S_a|}$, 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. We also compare the fastest and cheapest solutions identified in the different skylines. Table I summarizes the results for the metrics used. It can be seen that the Jaccard distance is significantly smaller for the proposed algorithm HHDS. Also, the fastest 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 2.32 times slower than HHDS. Similarly, for LIGO the fastest solution identified by MOHEFT is 6.3 times the value of HHDS. For Lattice11-

TABLE I: 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	0.30	0.84	0.0	1.0	0.0	1.0	0.0	1.0
Fast(s)	3.4	7.9	43.5	273.4	8.0	30.5	3.4	15.8
Cheap($)	86.7	91.01	1548.12	1549.18	395.58	407.67	395.58	411.15



(a) Montage (b) LIGO

(c) Lattice11-3 (d) Lattice5-21

Fig. 2: Comparison of the algorithms for the case of hourly pricing.

3, the fastest solution identified by MOHEFT is 3.8 times slower than the solution of HHDS. For Lattice5-21, the fastest solution identified by MOHEFT is 4.64 times the value of HHDS, In all cases, the cost of the cheapest solution identified by MOHEFT is slightly higher than the cost of the cheapest solution obtained by HHDS. Overall, the plans identified by HHDS cover a more divergent set of solutions in all scenarios examined.

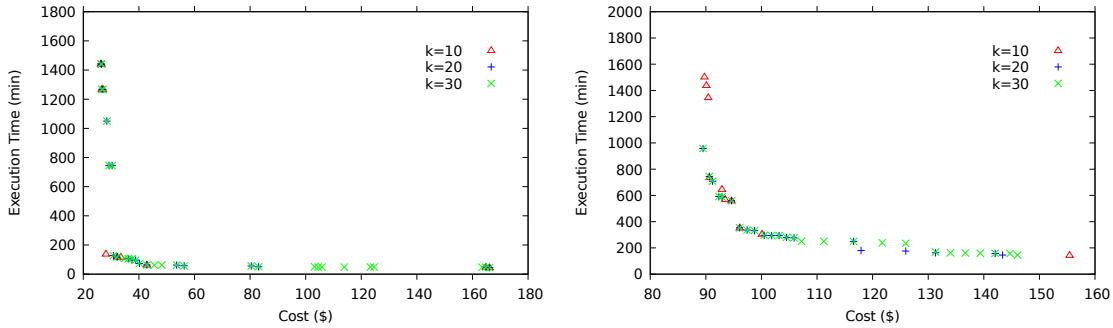
2) Hourly pricing: In this section, we compare the efficiency of the proposed algorithm, HHDS, with the baseline algorithm, MOHEFT, for heterogeneous resources priced by the hour. Multipliers are used for the operator runtimes and data sizes (100, 100, respectively) to generate dataflows with longer execution times (order of hours). The obtained results are presented in Fig. 2. It can be seen that HHDS 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 smaller.

Table II shows the results for the metrics used. It can be seen that the Jaccard distance is significantly smaller for the proposed algorithm HHDS. For Montage, the fastest solution identified by MOHEFT is 2.03 times the value of HHDS, while the cost of the cheapest solution is 7.18 times the value of HHDS. For LIGO, the fastest solution identified by MOHEFT is 5.05 times the value of HHDS, while the cost of the cheapest solution is slightly better compared to the solution of HHDS (0.98 times the value of HHDS). For Lattice11-3, the fastest solution identified by MOHEFT is 3.35 times the value of HHDS, while the cost of the cheapest solution is close to the value of HHDS (1.03 times the value of HHDS). For Lattice5-21, the fastest solution identified by MOHEFT is 4.38 times the value of HHDS, while the cost of the cheapest solution is only 1.04 times the value of HHDS.

Finally, the overhead imposed by our scheduler, the time required to generate the skyline compared to the time required to run the fastest plan, is less than 6% (and less than 1% in most cases), while there is room for optimization by parallelizing HHDS. Overall, both algorithms required a few seconds (smaller dataflows) to tens of minutes (more computationally demanding runs).

TABLE II: 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	0.11	0.91	0.23	0.88	0.0	1.0
Fast(s)	340.0	692.1	4373.2	22081.3	29889.2	100020.0	11611.2	50901
Cheap($)	2.68	19.24	45.6	44.56	395.6	409.23	395.58	412.4



(a) Montage (b) LIGO

Fig. 3: Different k values.

C. 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 equal to 10, 20 and 30 for per hour pricing are used. The results for the two real scientific dataflows, LIGO and Montage, where the number of solutions is quite large, are presented in Fig. 3. In the case of the Lattice dataflows the skyline consists of a smaller number of points (16 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 26 pareto-efficient schedules are computed. Overall, the efficiency of the algorithm is not greatly affected for different values of k. Although by decreasing k a smaller number of solutions is kept at the resulted pareto front, the quality of the skyline does not change significantly. Different dataflows are not equally sensitive to the number of representative skyline solutions kept. For example, LIGO is more sensitive to parameter k.

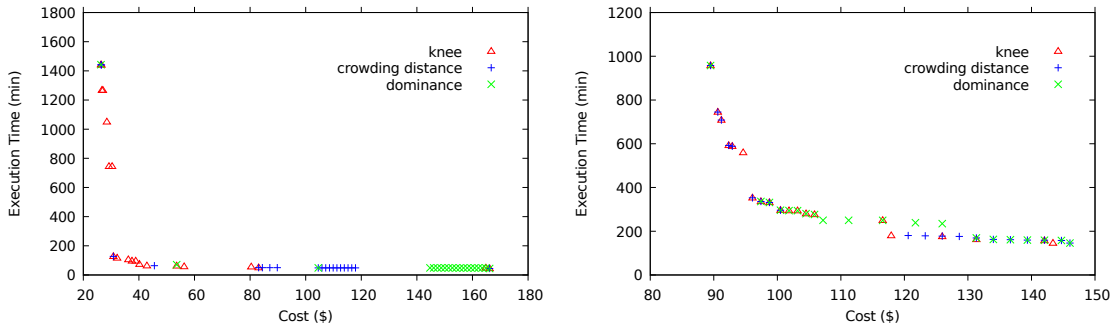
D. Comparison of Pruning Methods

In this experiment, we compare the efficiency of the proposed knee-based pruning metric with two state of the art metrics, the crowding distance [5] and the dominance area [9]. Fig. 4 shows the results for the Montage and LIGO dataflows 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 $k = 20$ are presented so that pruning is applied (since there are only 26 pareto plans for LIGO).

Crowding distance based pruning [5] 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.

Dominance-based pruning aims to select points which dominate a larger number of solutions and whose dominated areas differ so that the number of points dominated by the representative skyline is maximized. However, uneven distribution of solutions in the solution 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 keeps more solutions



(a) Montage (b) LIGO

Fig. 4: Different pruning methods used.

at the knee of the curve where more interesting trade-offs appear. While moving to the extreme cases (minimum time or money), fewer points are chosen as they may not differ greatly in terms of money or cost, especially for expensive plans.

E. Discussion

The pruning method used in 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 (e.g. distance or dominance based) criteria. However, the number and space of partial solutions generated may differ between 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. Different assignments of more critical operators may lead to more diverse schedules. 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. The algorithm does not change the number of VMs at each homogeneous solution considered in the second stage. Hence, the efficiency of the algorithm depends on the quality and diversity of the initial skyline of homogeneous configurations. Future work could investigate how its efficiency could be improved in such cases to explore heterogeneous configurations with varied number of VMs. Finally, communication costs and startup overhead may vary for heterogeneous VM types. A possible research direction could study their impact on the heterogeneity of skyline solutions for applications with a large amount of input data.

VI. Conclusion

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

Future work could try to improve the algorithm from pruning partial schedules by identifying and including dominant yet important solutions that may improve the skyline. Also, the algorithm can be extended for multiple dataflows; 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.

Acknowledgment

The authors would like to thank Herald Kllapi and George Valkanas for their help. This work has received funding from the European Union's Horizon 2020 research and innovation program GA No 720270 and Seventh Framework Program (FP7) GA No 318338.

References

[1] 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*, vol. 39, no. 4, pp. 177–188, 2013.

[2] 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 CLOUD*. IEEE, 2012, pp. 638–645.

[3] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng, "Cost optimized provisioning of elastic resources for application workflows," *FGCS*, vol. 27, no. 8, pp. 1011–1026, 2011.

[4] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," in *Proceedings of the e-Science*. IEEE, 2005, pp. 8–pp.

[5] 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 CloudCom*. IEEE, 2012, pp. 185–192.

[6] H. Topcuoglu, S. Hariri, and M. Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE TPDS*, vol. 13, no. 3, pp. 260–274, 2002.

[7] 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 HPCC*. IEEE, 2011, pp. 808–813.

[8] L.-C. Canon *et al.*, "MO-Greedy: an extended beam-search approach for solving a multi-criteria scheduling problem on heterogeneous machines," in *Proceedings of the IEEE IPDPSW*. IEEE, 2011, pp. 57–69.

[9] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, "Selecting stars: The k most representative skyline operators," in *Proceedings of the 23rd IEEE ICDE*. IEEE, 2007, pp. 86–95.

[10] G. Valkanas, A. N. Papadopoulos, and D. Gunopulos, "Skyline ranking à la IR." in *Proceedings of the EDBT/ICDT Workshops*, 2014, pp. 182–187.

[11] S. Ganguly, W. Hasan, and R. Krishnamurthy, *Query optimization for parallel execution*. ACM, 1992, vol. 21, no. 2.

[12] I. Trummer and C. Koch, "A fast randomized algorithm for multi-objective query optimization," in *Proceedings of the ACM SIGMOD*. ACM, 2016, pp. 1737–1752.

[13] V. T'Kindt and J.-C. Billaut, *Multicriteria scheduling: theory, models and algorithms*. Springer Science & Business Media, 2006.

[14] 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*, 2011, pp. 289–300.

[15] D. Bozdag, U. Catalyurek, and F. Ozguner, "A task duplication based bottom-up scheduling algorithm for heterogeneous environments," in *Proceedings of the 20th IEEE IPDPS*. IEEE, 2006, pp. 12–pp.

[16] H. Wu, X. Hua, Z. Li, and S. Ren, "Resource and instance hour minimization for deadline constrained DAG applications using computer clouds," *IEEE TPDS*, vol. 27, no. 3, pp. 885–899, 2016.

[17] Y. Chronis *et al.*, "A relational approach to complex dataflowss," in *EDBT/ICDT Workshops*, 2016.

[18] D. S. Katz *et al.*, "A comparison of two methods for building astronomical image mosaics on a grid," in *Proceedings of the IEEE ICPPW*. IEEE, 2005, pp. 85–94.

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

[20] Workflow Generator, "https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.

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