Fairness in Dataflow Scheduling in the Cloud

Ilia Pietri[a], Yannis Chronis[b], Yannis Ioannidis[a]

[a]*Department of Informatics and Telecommunications, University of Athens, Greece*
[b]*Computer Sciences Department, University of Wisconsin - Madison, United States*

Abstract

Expensive dataflow queries which may involve large-scale computations operating on significant volumes of data are typically executed on distributed platforms to improve application performance. Among these, cloud computing has emerged as an attractive option for users to execute dataflows allowing them to select proper configurations (e.g., number of machines) to achieve desired trade-offs between execution time and monetary cost. Discovering dataflow schedules that exhibit the best trade-offs within a plethora of potential solutions can be challenging, especially in a heterogeneous environment where resource characteristics like performance and price can be varied. To increase resource utilization, users may also submit multiple dataflows for execution concurrently. Traditionally, building fair schedules (schedules where the slowdown of all dataflows due to resource sharing is similar) while achieving good performance is a major concern. However, considering fairness in the cloud computing setting where monetary cost is part of the optimization objectives significantly increases the difficulty of the scheduling problem. This paper proposes an algorithm for the scheduling of multiple dataflows on heterogeneous clouds that identifies Pareto-optimal solutions (schedules) in the three-dimensional space formed from the different trade-offs between overall execution time, monetary cost and fairness. The results show that in most cases the proposed approach can provide solutions with fairer schedules without significantly impacting the quality of the execution time to monetary cost skyline compared to the state of the art where the fairness of a solution is not taken into account.

Keywords: Cloud computing, multiple dataflows, fairness, dataflow scheduling

1. Introduction

Big data applications may require the execution of expensive queries with the processing of large volumes of data. Such dataflow queries (or dataflows) can be modelled as a Directed Acyclic Graph (DAG) to describe operators (large-scale computations) and data flow dependencies between them. In a quest for performance optimization, distributed systems have been extensively used for the execution of dataflows allowing

*Corresponding author
Email address: ipietri@di.uoa.gr (Ilia Pietri)

users to take advantage of their degree of parallelism and run independent operators (operators without any data dependencies between them) simultaneously. On that account, optimization of application performance has been a major focus of research on dataflow scheduling.

With cloud computing gaining popularity for the execution of complex, scalable applications by offering a flexible environment to provision resources on demand on a pay-per-use basis, monetary cost has become an equally important optimization objective to consider when selecting from the number of resources on which to schedule a dataflow. Optimization objectives may often be conflicting resulting in a large space of solutions with diverse trade-offs. For example, using a large number of resources will usually lead to better performance at the expense of a higher monetary cost, compared to executing all operators in a single resource. Often, a slightly longer execution time may be tolerated when it comes with significant cost savings. Exploring configurations with good trade-offs between the conflicting objectives is the aim of multi-objective query optimization (MOQO). The complexity of the MOQO problem may further increase with cloud providers offering heterogeneous resources which may differ in their characteristics in terms of price and performance; different combinations of resource types, each with a different number of virtual machines (VMs), can be chosen. Although leading to a significantly larger space of alternative solutions compared to the homogeneous case, heterogeneous configurations may be preferred in several cases exhibiting better trade-offs.

In a distributed environment, a user may also submit for execution several independent dataflow queries (DAGs) at the same time to share resources between them. Such DAGs can be interleaved at a single execution schedule to better utilize the resources, exploiting idle slots for the execution of different DAGs. However, as these DAGs compete for the same resources, their performance may be affected compared to the scenario of isolated execution (each single DAG being executed alone), e.g. by assigning several operators to later slots or using less resources in a DAG. When scheduling multiple dataflows, providing fair schedules (schedules where all co-scheduled DAGs experience similar slowdown due to resource sharing) while achieving overall good performance are important optimization objectives [1, 2]. The structure and characteristics between different DAGs may significantly differ. For example, several DAGs may have a larger degree of parallelism, a smaller number of operators or much shorter operator runtimes. Such factors need to be considered when determining the order that operators from different DAGs are scheduled as they can significantly impact the fairness achieved in a schedule [3]. Achieving fairness in the Cloud when monetary cost is an additional concern in the optimization problem for the scheduling of multiple dataflows makes decision making even more difficult.

In this work, we present "Homogeneous to Heterogeneous Dataflow Scheduling with Fairness" (HHDS-F), a fairness-aware scheduling algorithm for the execution of multiple dataflows on heterogeneous clouds. The algorithm tries to identify Pareto-optimal trade-offs between overall execution time, monetary cost and fairness exploring the solution space in an efficient way. To the best of our knowledge, this is the first work that deals with the scheduling of multiple dataflows to achieve Pareto-efficient solutions in the three-dimensional space formed by the different trade-offs between overall execution time, monetary cost and fairness.

The main contributions of this work are the following:

- We develop a two-stage heuristic for the scheduling of multiple dataflows on heterogeneous cloud resources to investigate Pareto-efficient solutions with respect to overall execution time, monetary cost and fairness.
- We present a novel pruning method to select a number of respresentative solutions distributed along the Pareto curve by favoring points at sharper parts of the curve and leaving fewer points at flatter parts.
- We present a prioritization scheme to rank operators that come from DAGs with different characteristics and determine the order they are scheduled so that fairness can be promoted.
- We provide an experimental evaluation and comparison with the state of the art to show the effectiveness of the proposed approach to provide a better and more divergent skyline of solutions using realistic dataflows.

The remainder of the paper is organized as follows. Related work is presented in Section 2. Problem description follows in Section 3. The proposed approach is described in Section 4. The experimental evaluation and its results are discussed in Section 5. Finally, Section 6 concludes the paper.

2. Related Work

DAG scheduling [4] on distributed environments like grids and clouds has been extensively studied, traditionally aiming at optimizing execution time or monetary cost [5, 6, 7, 8, 9, 10, 11]. Several single-objective or constrained multi-objective optimization algorithms have been proposed [5, 8, 9], while others [6, 12] formulate the multi-objective optimization problem as a weighted single-objective problem incorporating user preference parameters into the objective function to balance the trade-off between the different objectives.

Our work is more related to Pareto-based approaches [7, 10, 13] that follow the concept of Pareto dominance to generate a set of non dominated solutions (skyline) and achieve different trade-offs between execution time and monetary cost when there is no single schedule that optimizes all the objectives. MOHEFT [10] extends HEFT [5], a well-known scheduling heuristic for the minimization of overall execution time on heterogeneous resources, by keeping at each step a set of partial solutions as opposed to a single schedule where the task is assigned to the slot with the earliest finish time. The selection of solutions to be kept is based on the crowding distance metric that tries to measure the surrounding area of each skyline point where no other skyline point exists. Different pruning schemes for the selection of a number of representative points in the skyline have been investigated [14, 15]. The maximum coverage representative skyline is computed in [14] selecting points that maximize the area of dominant solutions (dominance area), while the dominance power-inverse dominance power metric is introduced in [15] to favor solutions which dominate more genuine points, points that are dominated by fewer points.

Most of the aforementioned DAG scheduling approaches consider single dataflows. The problem of multiple DAG scheduling has also been studied [16, 2, 17, 1, 18]. Multiple DAGs with similar or different structures can be scheduled in different ways, such as one after the other, merged into a composite DAG with a common entry or in turns interleaving the tasks of different DAGs in the schedule [2, 1]. Fairness, defined by the slowdown of each DAG experienced due to resource sharing with other DAGs [1], is an important aspect in multiple DAG scheduling and several algorithms that account for fairness have been proposed and evaluated [2, 1, 19, 16, 18, 3]. The RANK_HYBD algorithm in [16] prioritizes jobs with larger upward ranks when they come from the same DAG (same as HEFT [5]) and smaller upward ranks when they come from different DAGs to favor jobs closer to the exit nodes or less complex jobs. The dynamic scheduler in [18] ranks the tasks of different DAGs based on the percentage of unscheduled tasks of the DAG and its critical path length with the aim to prioritize DAGs that are almost completed or only have few tasks to execute. Our work is more related to static algorithms that optimize both execution time and fairness like [1]. However, our work focuses on fairness in the Cloud when financial costs are part of the optimization problem. Thus, we also consider the monetary cost in decision making to explore different trade-offs for the scheduling of multiple DAGs of a single user that come at the same time. A considerable body of work considers multi-resource fairness to allocate proper resource shares (e.g., CPU and bandwidth) to users depending on the different job resource demands [20]. Our work focuses on the impact of operator assignments to different resource slots on the makespan of each dataflow in the solutions built.

Finally, multi-objective query optimization (MOQO) approaches like [21, 22] 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 [22] proposes approximation schemes to solve the optimization problem in cloud environments. Such MOQO schemes are complementary to our work, as they focus on a different level of query plan optimization.

3. Problem Description

Varying the assignments of dataflow operators may result in solutions with divergent trade-offs, among which several solutions may be better than others with respect to one or more objectives. A single optimal solution that outperforms all others may not exist, as the objectives may be conflicting. Following the principle of Pareto dominance [23], the set of non-dominated solutions comprises a *Pareto front* (*skyline*). This work focuses on the scheduling of multiple dataflows on the Cloud which are ready for execution at the same time with the aim to build fair schedules while optimizing overall execution time and monetary cost.

3.1. Application model

The dataflows under study can be modelled as a Directed Acyclic Graph (DAG), where the nodes represent operators (computations) and the edges flows of data transfers between them. We consider store-and-forward operators like in [24] where the execution of an operator can only start after all the input data from its predecessors are

available. The simplified performance model of Equation 1 is assumed with operator runtimes being inversely proportional to the computational speed of the VM assigned:

$$runtime_{op} = \frac{size_{op}}{speed_{vm}}, \tag{1}$$

where $size_{op}$ is the size of operator op in millions instructions and $speed_{vm}$ is the processing capability (computational speed) of the VM type given in millions instructions per second. In reality, performance may vary for jobs with different resource characteristics, such as CPU-intensive or I/O-bound jobs. The performance model can be easily extended to account for performance variation between operators with different resource characteristics and use more accurate runtime estimates.

Cloud Model. Cloud providers offer resources in the form of VMs which may differ in their characteristics, such as performance and price. We consider a model similar to Amazon Elastic Compute Cloud (EC2) [25] with VMs being provisioned on demand for as much time as needed. The user is charged for the whole time quanta leased with partial time units 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. A VM is considered to be active during a time quantum when one or more operators run on it. Also, we only consider the computing service, assuming that the storage service is accessible from each VM at the same cost (network bandwidth). A shared cloud storage system, such as Amazon S3 [26], 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. Data transfer between operators that run on the same VM is 0 like in [24]. After the execution of an operator completes, 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, like in [27].

Multi-objective Model. As mentioned earlier, a user may submit multiple DAGs to be scheduled at a given time. The dataflows may differ in their characteristics such as the number of operators and DAG structure. An obvious concern for the user is achieving overall good performance, e.g. incur overall low cost and execution time. However, the performance of each single dataflow may also be of great importance. We quantify the quality of an execution schedule based on its overall makespan, monetary cost and fairness achieved. Overall makespan (or schedule length) is the time required to complete the execution of the set of the DAGs and can be computed as the maximum completion time between all the operators of the co-scheduled dataflows [18]. Similarly, overall monetary cost is the cost required to run all the DAGs in the schedule and is given by the summation of the number of time quanta each VM in the schedule is leased multiplied by its price. The quality of a schedule with respect to fairness is quantified using an unfairness metric that measures the difference between the slowdown of the DAGs that experience due to co-scheduling. The slowdown (or stretch) of a DAG is given by the level of performance (i.e., makespan) achieved when DAGs are co-scheduled

divided by the level of performance achieved when each DAG is scheduled alone [19]:

$$Slowdown_d = \frac{Makespan_{schedule_d}}{Makespan_{own_d}}, \tag{2}$$

where $Makespan_{schedule_d}$ is the makespan of DAG d in the schedule and $Makespan_{own_d}$ is the makespan when DAG d is scheduled alone. Lower values of Equation 2 indicate better performance in the presence of multiple dataflows (smaller slowdown). The inverse of the standard definition of the slowdown is also used in the literature [1]. Unfairness is defined as the average absolute value of the difference between the stretch (or slowdown) of each DAG d, $Slowdown_d$, and the average stretch (slowdown) per DAG, *AvgSlowdown* [19]:

$$Unfairness = \sum_{\forall d \in dags} |Slowdown_d - AvgSlowdown|. \tag{3}$$

Lower values of unfairness (close to 0) indicate fairer schedules where the slowdown of all DAGs is similar while larger values of unfairness indicate schedules where the slowdown of each DAG varies. In this work, the standard definition of slowdown (Equation 2) is used in decision making to build fair schedules where dataflow performance is affected as less as possible.

4. Algorithm Description

In this section a multi-objective scheduling algorithm, *Homogeneous to Heterogeneous Dataflow Scheduling with Fairness (HHDS-F)*, that iteratively builds a skyline of solutions for the execution of a set of dataflows on suitable VMs is presented. The algorithm extends HHDS in [28] to compute Pareto-efficient solutions in the three dimensional space formed by the different trade-offs between execution time, monetary cost and unfairness for multiple dataflows. HHDS-F uses a new ranking scheme to prioritize operators that come from different DAGs so that fairness is achieved and extends the knee-based pruning method to explore good trade-offs between execution time, monetary cost and unfairness. Similarly with HHDS [28], HHDS-F works in two stages; in the first stage, it tries to explore homogeneous solutions close to the optimal Pareto front for each available VM type, iteratively assigning the operators to potential idle slots. Based on their unified skyline, the algorithm explores heterogeneous configurations with lower execution time, monetary cost and/or unfairness in the second stage, gradually changing the type of VMs at each schedule when newly created solutions improve the obtained skyline.

4.1. Computation of Homogeneous Skyline Stage

The first stage of HHDS-F is shown in Alg. 1. The algorithm initially ranks the operators following the procedure described in Section 4.2 to determine the order in which they will be scheduled, providing a local ordering for operators within a DAG and a global ordering for operators from different DAGs. Next, for each VM type available, the algorithm iteratively builds the skyline of homogeneous solutions, assigning

the next ready operator (the operator with the highest priority) to all potential slots in each partial solution considered. These include both idle slots at already used VMs and slots at newly added VMs which meet the data dependency constraints (lines 6-9). The solutions built at the end of the step (i.e., the partial schedules built with the newly added operator) are then pruned to discard dominated solutions that do not belong to the skyline. The skyline is further pruned (line 12) when the number of non-dominated solutions is larger than the predefined parameter k (the desired number of solutions to keep) using the approach described in Section 4.2.3. Finally, a parameter ($vmUpgrading$) to indicate whether the type of the VM used at a schedule can be upgraded, degraded or both in the second stage (line 6) is assigned for each solution kept.

4.2. Operator Ordering

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. The selection of the operator to be scheduled next from the list of ready operators is based on their ranking. The following procedure is used to assign priorities: operators within each DAG are ranked based on their mean slack and level like in [29] to obtain an ordered list for each DAG (local ordering) and then the separate lists are interleaved to generate a global ordering and prioritize operators that may come from different DAGs: the first operator from the ordered list of each DAG is selected and among them the operator with the largest global rank is prioritized. In brief, the operator with the highest local priority in the DAG with the highest global rank is prioritized.

4.2.1. Local ordering of operators from a single DAG

Operators within a DAG are initially divided into levels (stages) based on the data dependencies between them. The level of each operator is given by its longest path

Algorithm 1 Homogeneous Skyline Computation Algorithm

```
1: procedure GENERATEHOMOSKYLINE(dags, vmType, k)
2:     readyOps ← operators from all dags with no dependencies
3:     while readyOps ≠ ∅ do
4:         op ← ready.pollFirst() ▷ operator with maximum local priority from the DAG with the highest global rank
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
```

from an entry node of the dataflow (*dag*), computed as the maximum level of its predecessors, $preds_{op}$, increased by 1:

$$level_{op} = max_{p \in preds_{op}} level_p + 1, \tag{4}$$

with the level of each entry node (operators without any predecessors) being 0. Operators in lower levels are assigned a higher priority. Between operators of the same level more critical operators are prioritized. The criticality of an operator is indicated by its mean slack on the different VM types, computed based on the summation of their upward and downward ranking (the longest path from an exit and entry node, respectively) like in [27]. A larger summation indicates more critical operators (a smaller slack). The upward rank is computed recursively starting from an exit node as:

$$uRank_{op} = \overline{w}_{op} + max_{s \in succs_{op}}(uRank_s + comCost_{op \to s}), \tag{5}$$

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:

$$dRank_{op} = max_{p \in preds_{op}}(\overline{w}_p + dRank_p + comCost_{p \to op}). \tag{6}$$

The weight (rank) of the operator op is then given by Equation 7:

$$rank_{op} = uRank_{op} + dRank_{op}. \tag{7}$$

An operator op is considered to be on the critical path if $rank_{op} = \max_{\forall i \in dag} rank_i$ [27]. Hence, operators with larger weights are assigned a higher priority.

4.2.2. Global ordering of operators from different DAGs

The idea is to alternate operators from different DAGs so that the scheduling of operators from DAGs that have a larger percentage of remaining operators or a larger percentage of the critical path to complete are prioritized. This is opposed to the goal of dynamic scheduling in [18] to prioritize DAGs that are almost completed or have a few tasks to run assigning ranks based on the inverse of the percentage of remaining operators and the percentage of the critical path. Thus, we use Equation 8 to assign priorities (global rank) between operators that belong to different DAGs so that fairness is achieved:

$$gRank_{op_d} = remOpsPerc_d \cdot remPathPerc_d, \tag{8}$$

where $remOpsPerc_d$ is the percentage of unordered operators in DAG d of operator op and $remPathPerc_{op_d}$ is the longest path from the operator to an exit node (upward rank) divided by the critical path of the DAG computed using Equation 9.

$$remPathPerc_{dag} = \frac{uRank_i}{\max_{\forall i \in dag} rank_i}, \tag{9}$$

with $uRank_i$ and $rank_i$ computed using Equations 5 and 7, respectively. Note that communication cost is considered in the notion of the path (included in the computation of the ranks) as the impact of data transfers on data-intensive flows may be significant.

4.2.3. Skyline Pruning

The number of different execution schedules built at each step may be large and exhaustive search (keeping all the potential solutions) may not be feasible. To reduce the search space, the skyline of partial solutions is computed and pruned at each step to keep only a number of representative solutions that outperform in terms of execution time, monetary cost or unfairness.

The *knee-based pruning* method proposed in [28] is extended to select representative points in the three-dimensional space of trade-offs between execution time, monetary cost and unfairness when the number of skyline solutions is large (more than the predefined parameter k).

As solutions are only partially built and overall makespan is not known until all operators from the DAGs are scheduled, the notion of partial unfairness is introduced to select partial solutions that can lead to schedules where fairness can be promoted. We compute partial unfairness based on Equation 3 by setting the slowdown of each DAG to:

$$Slowdown_d = \frac{Makespan_{partial_d}}{Makespan_{partialOwn_d}}, \tag{10}$$

where $Makespan_{partial_d}$ is computed as the maximum completion time between the already scheduled operators of DAG d based on the operator assignments in the partial solution built and $Makespan_{partialOwn_d}$ is the maximum completion time between the scheduled operators in the case of isolated execution of the DAG which is computed as the maximum downward rank of Equation 6 between the already scheduled operators based on the structure of the DAG [1]. The idea is to compare the makespan achieved in the partial solution built with a partial critical path, as the schedule is not fully built.

Knee-based pruning aims to identify the *knees* of the Pareto curve, i.e. skyline points which differ significantly in terms of time, money or unfairness from solutions close to them offering more interesting trade-offs. Such solutions may be important as they can result in high savings compared with other solutions. For example, significant savings in monetary cost may be achieved with a small increase in execution time and unfairness. Thus, skyline points at the knees of the Pareto curve have a higher probability of being selected. To do so, we compute the average value θ''_{sp} of the discrete second partial derivatives $\theta tc''_{sp}$ and $\theta uc''_{sp}$ of each skyline point sp of execution time and unfairness with respect to monetary cost, respectively. The discrete second partial derivative $\theta tc''_{sp}$ is approximated as the difference between the first order partial derivative of execution time with respect to cost of sp and its adjacent left and right skyline points, and $\theta uc''_{sp}$ is similarly computed for unfairness. We consider a skyline point to be a knee if θ''_{sp} is more than or equal to the mean of all the skyline points. Finally, points that are distant from the knees also have a higher probability to be kept so that the representative skyline can cover a wider area of solutions. The procedure followed to prune the skyline of solutions in each step is described next. The two extreme points which correspond to the cheapest and most expensive plans in the sorted skyline

[1] Note that we consider the mean execution time $\overline{w}_{op}$ of each operator op in the DAG and the average communication cost in the computation of the partial critical path as the set of critical operators may vary in different configurations (VM types and number of VMs) in the heterogeneous environment.

are kept and $k-2$ solutions are additionally selected based on the scoring function of Equation 11 (knee-based metric):

$$score_{sp} = \theta''_{sp} \cdot dist_{sp,kn}, \tag{11}$$

where θ''_{sp} is the average value of the discrete second partial derivatives of skyline point sp of execution time and unfairness with respect to monetary cost and $dist_{sp,kn}$ is its euclidean distance from its nearest knee kn of the Pareto curve. Skyline points with larger scores are preferred. Parameters θ''_{sp} and $dist_{sp,kn}$ of Equation 11 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.

4.3. Computation of Heterogeneous Skyline Stage

After computing the unified skyline from the homogeneous configurations obtained for the available VM types (line 4 of Alg. 2), the algorithm tries to explore heterogeneous configurations that improve the skyline with better and more diverse trade-offs. This is done by gradually upgrading or degrading the VM types used at the obtained configurations according to the value of the parameter *vmUpgrading* assigned to each solution. For homogeneous solutions with the parameter set to *ascending/descending* indicating that both directions (upgrading the VMs or degrading the VMs) can be followed, the algorithm keeps two copies of the plan in the skyline and changes the parameter of the copies to *ascending* and *descending*, respectively, to explore configurations

Algorithm 2 Heterogeneous Skyline Computation Algorithm

```
1:  procedure GENERATEHETEROSKYLINE(dags, vmTypes, k)
2:      Order the operators from all dags globally
3:      pareto.add(generateHomoSkyline(dags, vmType, k)), ∀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 is dominated by plan    ▷ no improvement on time, cost or partialUnfairness
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
```

with larger VM types (first copy) and smaller VM types (second copy). The procedure that follows is repeated to modify the current plans and improve the skyline. For each plan, the slack time of each operator, the time the execution of the operator can be delayed without extending overall dataflow execution time, is computed as:

$$slackTime_{op} = lst_{op} - est_{op}. \quad (12)$$

In contrast to mean slack defined in Section 4.2 which is independent of the operator assignments to VMs, the computation of the slack time in Equation 12 accounts for both the structure of the DAG and the specific assignments in the plan. The earliest start time of each operator op at the plan is computed recursively as: $est_{op} = max_{p \in preds_{op}}(est_p + runtime_p + comCost_{p \to op})$, where $preds_{op}$ includes both the predecessors of the operator in the DAG and the VM assigned. The latest start time of op is given by: $lst_{op} = min_{s \in succs_{op}}(lst_s - comCost_{op \to s} - runtime_{op})$, with $succs_{op}$ including both the successors in the DAG and the VM assigned. 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 (based on the plan's $vmUpgrading$ value) and the slots assigned to operators are updated. The monetary cost and the execution time of the newly generated plan are computed and the plan is kept when savings in monetary cost and/or execution time can be achieved (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 and non-representative solutions (line 20). The algorithm continues to the next iteration following the same procedure for the newly created plans in the representative skyline to upgrade/degrade the VMs to the next available VM type and generate new solutions. The algorithm terminates when no new solutions are kept in the computed representative skyline or the smallest/largest VM type is reached for every plan returning the obtained skyline.

5. Experimental Evaluation

In this section, the skyline obtained using the proposed algorithm HHDS-F is evaluated and compared with the state of the art based on simulations for two different dataflow families.

5.1. Methodology

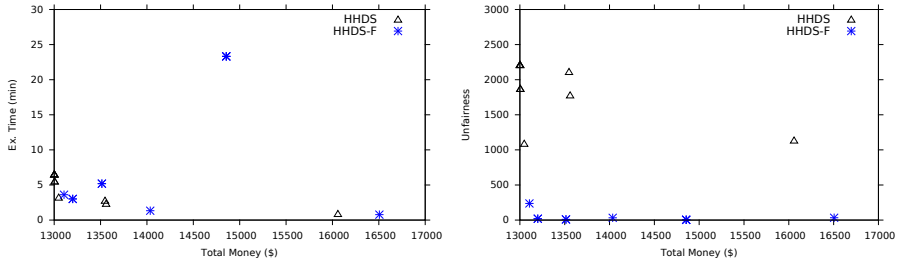
The simulator of the Exareme distributed dataflow processing engine [24, 30] was modified and used to implement and evaluate the proposed approach [31]. Different VM types are assumed using five Amazon EC2 instance types, namely, m1.small, m1.large, m2.xlarge, m2.2xlarge and m2.4xlarge, to cover different instance families. The time required for the execution of each operator at the different VM types is modelled based on the CloudHarmony Compute Units[2]. The communication cost between

[2]https://cloudharmony.com/

operators assigned to different VMs is computed assuming a network of 1Gbps, while it is considered to be 0 between operators that run on the same VM. HHDS [28] is used as a baseline to evaluate the performance of the proposed approach HHDS_F modelling the set of dataflows to be scheduled as a DAG of independent subdags that can start at the same time (common entry). Pruning parameter k, which indicates the number of solutions to be kept in the representative skyline at each step, is set to 10. Synthetic data (operator runtimes and data sizes) of two real scientific applications, Montage [32] and LIGO [33], are used. Montage is an astronomy dataflow application used to generate image mosaics of the sky. LIGO is used to analyze galactic binary systems for the detection of gravitational waves in the universe. A set of 20 dataflows with mixed sizes (50 and 100 operators) is used with dataflows obtained by the workflow generator in [34]; an equal number of dataflows for each application and size is used. Finally, the labels "multiObj" and "biObj" refer to the pruning schemes where fairness is considered or not in decision making (HHDS-F and HHDS, respectively), while the labels "dagMerge" and "commonEntry" refer to the ranking schemes where operators from different subDAGs are prioritized or not (HHDS-F and HHDS, respectively).

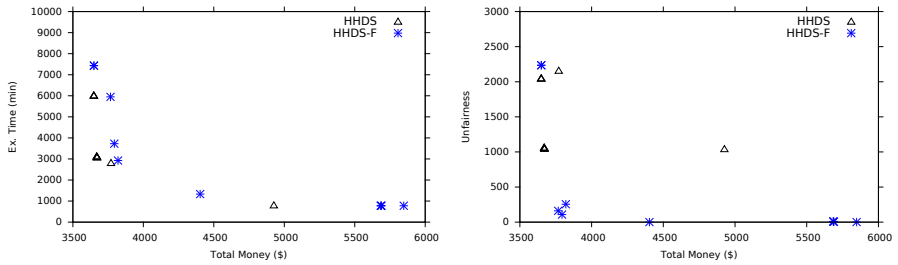
Comparison of skylines for different pricing schemes. Initially, the skylines obtained using the baseline algorithm HHDS and the proposed variant HHDS_F are compared. Figures 1a-1b and Figures 1c-1d show the results for per second and per hour pricing, respectively. As HHDS does not account for multiple dataflows, the dataflow ensemble is considered as a single DAG (superDAG) of subDAGs with different entry nodes. Also, in the case of hourly pricing, multipliers are used for the operator runtimes and data sizes (100, 100, respectively) to generate dataflows with longer execution times (order of hours), similarly to [28]. Overall, HHDS-F keeps schedules with lower values of unfairness sometimes at the expense of monetary cost or execution time, while there is a single solution obtained by HHDS-F which is dominated by a solution obtained using HHDS in the case of hourly pricing. This may be due to the fact that HHDS takes into account resource utilization to select between plans with similar execution time and cost while HHDS-F favors partial plans with lower values of partial unfairness.

Impact of pruning scheme. As mentioned earlier, the main differences between the algorithm HHDS and its variant HHDS-F are the pruning and prioritization schemes used. In this section, we investigate the impact of the pruning scheme in the quality of the obtained skyline (Figure 2). Pruning used by HHDS-F (labelled as multiObj) takes into account unfairness in decision making to also keep fair schedules where the delay in the execution of different subDAGs depends on their characteristics. For example, the makespan for subDAGs with shorter critical paths will be smaller (resulting in a similar slowdown). In contrast, HHDS may select solutions with low monetary cost or execution time where all the subDAGs have similar makespan although they may significantly differ in their characteristics. For example, the execution of operators from longer subDAGs which belong to lower levels may be favored delaying the execution of operators that belong to shorter subDAGs but a higher level. As a result, incorporating fairness in decision making leads to keeping solutions which may be dominated by other solutions in terms of execution time and cost (Figure 2a) but lead to fairer schedules (Figure 2b).



(a) Cost-time trade-off for per second pricing

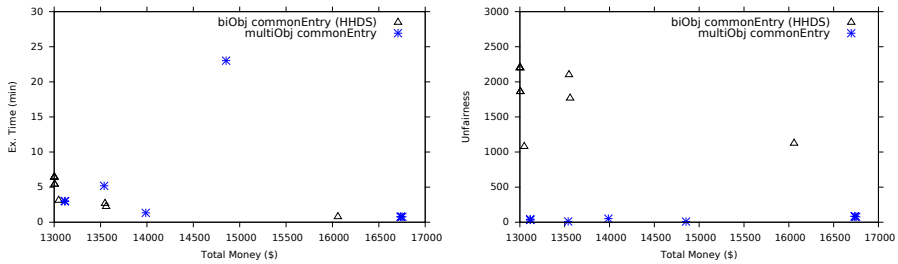
(b) Unfairness-cost trade-off for per second pricing



(c) Cost-time trade-off for per hour pricing

(d) Unfairness-cost trade-off for per hour pricing

Figure 1: Comparison of skylines obtained for different pricing schemes.

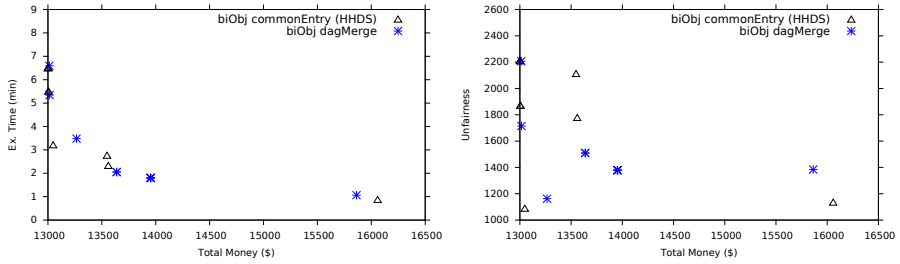


(a) Cost-time trade-offs

(b) Unfairness-cost trade-offs

Figure 2: Impact of pruning on the obtained skyline (per-second pricing).

Impact of prioritization scheme. We also investigate the impact of the prioritization scheme used on the quality of the obtained skyline (Figure 3). The skyline solutions obtained using the prioritization scheme of HHDS-F (labelled as dagMerge) are compared with the solutions obtained using HHDS to indicate the importance of global ranking for fairness. Although unfairness is not considered in decision making, prioritizing between operators from different DAGs may greatly affect the obtained skyline identifying in several cases more expensive but fairer execution schedules. There is a single solution obtained by prioritizing between operators from different DAGs which is dominated by a solution obtained using HHDS.



(a) Cost-time trade-offs (b) Unfairness-cost trade-offs

Figure 3: Impact of ranking on the obtained skyline (per-second pricing).

Selecting solutions based on user-defined constraints. The set of skyline solutions can be presented to users to manually select the configuration that better fits to their requirements. Alternatively, users may specify their preferences in advance to refine the solution space constraining the scheduling problem, such as setting the objective to maximize fairness while meeting specific deadline and budget constraints. Although by adding the constraints the solution space to be explored may be reduced, the quality of the solutions obtained may also be affected. In this section, we investigate the effectiveness of the algorithm to handle such scenarios in order to generate solutions that

Algorithm 3 Minimize unfairness under budget and deadline constraints

procedure GENERATESOLUTION($dags, vmTypes, k, budget, deadline$)
  Order the operators from all $dags$ globally
  $pareto.add(generateHomoSkyline(dags, vmType, k)), \forall vmType \in vmTypes$
  $plansToUpgrade \leftarrow getPrunedSkyline(pareto)$
    ▷ If $vmUpgrading \equiv asc./desc.$ keep a copy for each value
  while $plansToUpgrade \neq \emptyset$ **do**
    for $plan \in plansToUpgrade$ **do**
      $getSlackTime(op), \forall op \in p$
      $vmSorted \leftarrow$ sort VMs in plan based on their average slack
        ▷ asc. or desc. order based on $vmUpgrading$
      while $vmSorted \neq \emptyset$ **do**
        $vmToUpgrade \leftarrow vmSorted.pollFirst()$
        $nextVMType \leftarrow getnextVMType(vmToUpgrade)$
        $newPlan \leftarrow$ update schedule of $plan$ for $nextVMType$
        if $newplan$ is dominated by $plan$ ▷ no improvement on time, cost or partialUnfairness
          break
        else
          $modifiedPlans.add(newPlan)$
        end if
      end while
    end for
    $pareto \leftarrow getPrunedSkylineWithinConstraints(pareto \cup modifiedPlans, k, budget, deadline)$
    $plansToUpgrade \leftarrow \forall plan \in modifiedPlans \cap pareto$
  end while
  Return $pareto.getMinUnfairnessPlan$
end procedure

meet the user-defined objectives. To do so, we modified Algorithm 2 so that solutions that exceed the makespan and cost constraints are not considered in the computation of the pruned skyline (line 20), while the plan that minimizes unfairness is selected from the solutions obtained in the final skyline (line 23), as shown in Algorithm 3. Table 1 shows the results for different combinations of time and cost constraints covering scenarios with shorter/longer deadlines and lower/higher budgets. In most cases, the variant of the algorithm (Algorithm 3) returns solutions within the constraints. All of the solutions belong to the skyline obtained in Figure 1 of Algorithm 2 where no constraints are considered. In the case of the short deadline (2 mins) and low budget ($13.5x10^3$\$), there is no solution found that meets the constraints.

Table 1: Solutions for fairness under different budget and deadline constraints (per-second pricing).

Constraints		Solution with min unfairness			Difference from skyline		
Budget (x10^3\$)	Deadline (min)	Cost (x10^3\$)	Makespan (min)	Unfair-ness	Cost (%)	Makespan (%)	Unfair-ness (%)
13.5	2.0	no solution found			0	0	0
13.5	5.0	13.2	3.0	19.2	0	0	0
13.5	8.0	13.2	3.0	19.2	0	0	0
15.0	2.0	14.0	1.3	32.5	0	0	0
15.0	5.0	13.2	3.0	19.2	0	0	0
15.0	8.0	13.5	5.2	8.0	0	0	0
16.5	2.0	14.0	1.3	32.5	0	0	0
16.5	5.0	13.2	3.0	19.2	0	0	0
16.5	8.0	13.5	5.2	8.0	0	0	0

6. Conclusion

The work in this paper addressed the problem of scheduling multiple dataflows on heterogeneous clouds to identify Pareto-optimal schedules in the solution space formed by the different trade-offs between overall execution time, monetary cost and fairness. The proposed algorithm extends previous work by incorporating a ranking scheme to prioritize between operators that belong to different dataflows and a pruning method to account for fairness in addition to execution time and monetary cost introducing the notion of partial unfairness. Our proposed approach is static considering the scheduling of multiple DAGs submitted by a single user. Future work could deal with multiple DAGs that come at different times. Finally, improving the knee-based metric to better approximate the knees of the Pareto curve using all partial derivatives and investigating its impact on the quality of the skyline could be another direction of future work.

References

[1] H. Zhao, R. Sakellariou, Scheduling multiple DAGs onto heterogeneous systems, in: Proceedings of the 20th IEEE International Parallel Distributed Processing Symposium, 2006, pp. 14 pp.–.

[2] L. F. Bittencourt, E. R. M. Madeira, Towards the scheduling of multiple workflows on computational grids, Journal of Grid Computing 8 (3) (2010) 419–441.

[3] T. N'Takpe, F. Suter, Concurrent scheduling of parallel task graphs on multi-clusters using constrained resource allocations, in: Proceedings of the IEEE International Symposium on Parallel Distributed Processing, 2009, pp. 1–8.

[4] L.-C. Canon, E. Jeannot, R. Sakellariou, W. Zheng, Comparative evaluation of the robustness of DAG scheduling heuristics, in: Grid Computing: Achievements and Prospects, Springer, 2008.

[5] H. Topcuoglu, S. Hariri, M. Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE TPDS 13 (3) (2002) 260–274.

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

[7] K. Bessai, S. Youcef, A. Oulamara, C. Godart, 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.

[8] E.-K. Byun, Y.-S. Kee, J.-S. Kim, S. Maeng, Cost optimized provisioning of elastic resources for application workflows, FGCS 27 (8) (2011) 1011–1026.

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

[10] J. J. Durillo, H. M. Fard, R. Prodan, MOHEFT: A multi-objective list-based method for workflow scheduling, in: Proceedings of the 4th IEEE CloudCom, IEEE, 2012, pp. 185–192.

[11] R. Prodan, M. Wieczorek, Bi-criteria scheduling of scientific grid workflows, IEEE T-ASE 7 (2) (2010) 364–376.

[12] J. Li, S. Su, X. Cheng, Q. Huang, Z. Zhang, Cost-conscious scheduling for large graph processing in the cloud, in: Proceedings of the 13th IEEE HPCC, IEEE, 2011, pp. 808–813.

[13] 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.

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

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

[16] Z. Yu, W. Shi, A planner-guided scheduling strategy for multiple workflow applications, in: Proceedings of the International Conference on Parallel Processing - Workshops, IEEE, 2008, pp. 1–8.

[17] M. Xu, L. Cui, H. Wang, Y. Bi, A multiple qos constrained scheduling strategy of multiple workflows for cloud computing, in: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing with Applications, IEEE, 2009, pp. 629–634.

[18] H. Arabnejad, J. Barbosa, Fairness resource sharing for dynamic workflow scheduling on heterogeneous systems, in: Proceedings of the 10th International Symposium on Parallel and Distributed Processing with Applications, IEEE, 2012, pp. 633–639.

[19] H. Casanova, F. Desprez, F. Suter, On cluster resource allocation for multiple parallel task graphs, Journal of Parallel and Distributed Computing 70 (12) (2010) 1193 – 1203.

[20] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, I. Stoica, Dominant resource fairness: Fair allocation of multiple resource types, in: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, NSDI'11, USENIX, 2011, pp. 323–336.

[21] S. Ganguly, W. Hasan, R. Krishnamurthy, Query optimization for parallel execution, Vol. 21, ACM, 1992.

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

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

[24] H. Kllapi, E. Sitaridi, M. M. Tsangaris, Y. Ioannidis, Schedule optimization for data processing flows on the cloud, in: Proceedings of the ACM SIGMOD, 2011, pp. 289–300.

[25] AWS, Amazon Elastic Compute Cloud (EC2) (2018).
URL `http://aws.amazon.com/ec2`

[26] AWS, Amazon simple storage service (2018).
URL `https://aws.amazon.com/s3/`

[27] D. Bozdag, U. Catalyurek, 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.

[28] I. Pietri, Y. Chronis, Y. Ioannidis, Multi-objective optimization of scheduling dataflows on heterogeneous cloud resources, in: Proceedings of the IEEE International Conference on Big Data, IEEE, 2017, pp. 361–368.

[29] H. Wu, X. Hua, Z. Li, S. Ren, Resource and instance hour minimization for deadline constrained DAG applications using computer clouds, IEEE TPDS 27 (3) (2016) 885–899.

[30] Y. Chronis, et al., A relational approach to complex dataflows, in: EDBT/ICDT Workshops, 2016.

[31] MADGiK group, University of Athens. [link].
URL https://github.com/madgik/DataflowScheduler

[32] 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.

[33] LIGO, Laser interferometer gravitational wave observatory.

[34] Workflow Generator (2012). [link].
URL https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator