Cost models for geo-distributed massively parallel streaming analytics

Anna-Valentini Michailidou and Anastasios Gounaris
Aristotle University of Thessaloniki, Greece
{annavalen,gounaria}@csd.auth.gr

Konstantinos Tischlas
University of Patras, Greece
ktsichlas@ceid.upatras.gr

Abstract

This report is part of the DataflowOpt project on optimization of modern dataflows and aims to introduce a data quality-aware cost model that covers the following aspects in combination: (1) heterogeneity in compute nodes, (2) geo-distribution, (3) massive parallelism, (4) complex DAGs and (5) streaming applications. Such a cost model can be then leveraged to devise cost-based optimization solutions that deal with task placement and operator configuration.

1 Introduction

A fundamental research activity with a view to devising new methodologies regarding advanced dataflow (i.e., data-intensive workflow) optimization solutions is to develop and adopt appropriate cost models. Such cost models need to be capable of accurately reflecting the response time in parallel environments, taking into account issues such as partial overlapping, CPU core sharing and resource contention that are all common in modern dataflow executions. Furthermore, the cost models need to be extended to cover additional dataflow objectives, such as data quality. Finally, these cost models need to cover arbitrarily complex DAG (directed acyclic graph) execution plans. This report, apart from examining cost models that are directly amenable to optimizations, further discusses tractability issues regarding the optimization problems that typically accompany cost models.

More specifically, in Section 2 we present alternative approaches to develop cost models with a view to covering the landscape that is characterized by the combination of (1) heterogeneity in compute nodes, (2) geo-distribution, (3) massive parallelism, (4) complex DAGs and (5) streaming applications. We then highlight the need to develop our own cost model, which covers the afore-mentioned aspects, i.e., heterogeneity, massive
parallelism and geo-distribution, while being suitable for data-intensive streaming applications. This cost model is presented in detail in Section 3 and the relevant code is in https://github.com/annavalentina/Equality\textsuperscript{2} Section 4 concludes the report.

2 Cost Models for Dataflow Optimization

In this section, we discuss the main representatives regarding the cost models for data intensive analytics, along with the associated optimization problems, whenever such problems accompany a cost model presentation, and tractability analysis. The aspects that are of immediate interest include: massive parallelism, heterogeneity in terms of resources and (geo-)distribution, while we assume that the complete job is modelled by a (potentially complex) DAG. Please note that we use the terms DAG, topology, workflow (or dataflow) interchangeably in the text.

2.1 A Cost Model for a Multi-core Shared Memory Setting Tailored for Streaming Data Analytics

Zhang et al. \cite{37} propose a cost model that is used to maximize the processing throughput of data stream processing systems on shared-memory multi-core architectures. The throughput $R$ is equal to the sum of the output rates of all the sink nodes of the topology, that is $R = \sum_{\text{sink}} r_o$. The output rate $r_o$ for a time interval $t$ is estimated using the number of tuples that are processed, $num$, and the time needed to process them $t_p$. Hence $r_o = num/t_p$. $num$ is equal to the number of tuples generated from the previous nodes in the topology, $num = \sum_{\text{producers}} r_i \ast t$ where $r_i$ is the input rate. The $t_p$ variable is equal to $t_p = \sum_{\text{producers}} r_i \ast t \ast T(p)$ where $T(p)$ is the average time spent on handling each tuple and consists of the fetching ($T^f$) and execution ($T^e$) times. The fetching time $T^f$ is 0 in case the producer keeps its data locally; otherwise it is equal to $[N/S] \ast L(i,j)$ where $N$ is the average size of a tuple, $S$ the cache line size and $L(i,j)$ the worst case memory access latency between two CPU sockets $i$ and $j$ in a generic NUMA computing architecture. The variables $L(i,j)$ and $S$ are given as statistical input metadata, whereas $N$ and $T^f$ are found using profiling techniques. Finally, the input rate of the source operators is also given by the problem setting (denoted as $I$). The algorithms proposed in \cite{37} aim to maximize the throughput $R$ by deciding a placement of nodes to CPU sockets and a replication level for each operator. The proposed algorithms focus on the degree of parallelism of the operators taking into account the heterogeneity of the resources, without considering geo-distribution. Apart from this difference from our main focus, the heterogeneity in \cite{37} is limited and refers to the locality or not of data in a NUMA architecture, which impacts on the fetching time as described above.

In summary, this cost model emphasizes on the different fetching times based on the data memory address and the placement of the operator.

\textsuperscript{2}This code repository contains broader work.
2.1.1 Optimization Problem Details and Tractability

Zhang et al. [37] aim to maximize the application processing throughput, given an input stream rate by placing each operator on specific CPU sockets $S_i$, where $i = 1 \ldots n$, and deciding an optimal replication level in a multicore shared-memory architecture. The formulation of the problem they try to solve is the following:

$$\text{maximize} \sum_{\text{sink}} r_o \text{(i.e., expected } r_o)$$

s.t., $\forall i, j \in 1, ..., n,$

$$\sum_{\text{operators at } S_i} r_o \cdot T \leq C, \quad (1)$$

$$\sum_{\text{operators at } S_i} r_o \cdot M \leq B, \quad (2)$$

$$\sum_{\text{operators at } S_i} \sum_{\text{producers at } S_j} r_o(s) \cdot N \leq Q_{i,j}. \quad (3)$$

Equation (1) ensures that the aggregated CPU demand on each socket will not exceed its CPU capacity $C$, where $T$ is the average time spent on handling each tuple. Similarly, equation (2) ensures that the aggregated amount of bandwidth requested on a socket (calculated using the average memory bandwidth consumption per tuple, $M$) will not exceed its maximum attainable local DRAM bandwidth $B$. Finally, equation (3) enforces that the data transfer from one socket to another must not exceed the maximum remote channel bandwidth. It is also constrained that each operator is placed exactly once.

Each operator of the total $|o|$ operators can be replicated at most $k$ times, thus the total possible configurations are $k^{|o|}$. For each of these configurations, there exist $m^n$ different placements, where $m$ is the number of sockets and $n$ the number of replicas. Due to the exponential search space and the corresponding ILP that needs to be solved, a brute-force approach is prohibitive. The authors propose a branch and bound algorithm that iteratively places operators under a given replication level and then tries to increase the degree of replication for the bottleneck operator. In total, there exist $(n \cdot m)^n$ candidate solutions ($n \cdot m$ for $n$ iterations) and three heuristics are used in order to speed up the traversal of the solution space.

2.2 Cost Models for Data Flows Tailored to Parallel Executions

Kougka et al. [20] present three cost models for data flow response time in a massively parallel environment taking into account execution overlaps. Their approach can be extended to parallel distributed settings but heterogeneity of the resources is not considered.

The first two models refer to pipelined segments where the tasks in the data flow DAG form a chain and, also, no movement of data over the network occurs, since the execution takes place on a single machine (thus making communication cost time negligible). The idea behind the first cost model is that each task can be executed on a separate core
resulting to overlapping execution times. In such a case, the slowest execution time defines the total response time. Also the authors in [20] account for overheads coming from factors such as memory locks, contention and context switching between threads, where all these factors are captured by a single variable $\alpha$. The model proposed is the following:

$$\text{ResponseTime}(RT) = \alpha \times \max\{c_1, ..., c_n\},$$

where $c_i$ is the execution time of task $i$ in the DAG.

In the second model, the case where a core needs to take on more than one task is examined. The new model is the following: $\text{ResponseTime}(RT) = \alpha \times \max\{\max\{c_1, ..., c_n\}, \sum\{c_1, ..., c_n\}/m\}$ where $m$ is the total number of cores. The outcome of this model is the same as the first one in case there exists a single task with execution time higher than the sum of the execution times of all the tasks.

Finally, a generalized model is presented that takes into account multiple pipeline segments and multiple machines. The model presented is the following: $\text{ResponseTime}(RT) = \sum z_i \times w^e \times c_i + \sum z_{ij} \times w^{cc} \times cc_{i \rightarrow j}$. The first sum refers to execution time whereas the second one to communication cost time. The $z$ variables are binary ones that are equal to 1 when the task and the communication costs contribute to the final $RT$, capturing the overlapping of tasks. The $w$ variables generalize the $\alpha$ variable used in the previous models and capture the possible overheads of the execution. Finally, $cc_{i \rightarrow j}$ represents the communication cost of the edges between the tasks.

In summary, the cost models in [20] emphasize on capturing the effect of concurrent task executions on the response time in a parallel homogeneous setting.

In a similar setting, Ali et al. [4] propose a cost model for optimizing User Defined Functions (UDFs) in data-intensive workflows, such as ETL (Extract-Transform-Load) workflows. They try to optimize the execution of the UDFs in ETL workflows based on the execution time and monetary costs of task invocations. The optimization is divided into three stages. In the first one, the cost model determines if a UDF can be parallelized and in the second stage, it determines the degree at which it should be parallelized. Finally, in the third stage, the cost model generates an efficient machine configuration. The cost model is used to simulate the UDF execution and derive estimates regarding execution time as well as other metrics like the dataset size, the execution speed, CPU, I/O, memory characteristics etc.

In summary, the cost model in [4] focuses on the impact of increased degree of parallelism regarding certain expensive operators in the DAG.

### 2.2.1 Optimization Problem Details and Tractability

The cost model in [20] mentioned above is leveraged to find an optimal (partial) ordering of the operators (nodes) in a DAG, in the case that multiple orderings can produce the same result. The response time optimization in such a setting is not simpler than the general task ordering optimization problem aiming at the minimization of the sum of the costs, as reported in [21]. This problem, as proven in [8], is intractable and more importantly “it is unlikely that any polynomial time algorithm can approximate the optimal plan to within
a factor of $O(n^\theta)$”, where $\theta$ is some positive constant and $n$ is the number of the tasks in the data flow.

Ali et al. [4], as already explained, optimize the execution of UDFs in ETL workflows using also the sum cost metric. The workflows are divided into stages and each stage can be processed using multiple code/implementation variants. The objective is to find the optimal variant for each stage. In total, there exist $n^m$ possible variants ($m$ stages, $n$ code variants for each stage). This problem is NP-hard since the Multiple Choice Knapsack Problem (MCKP) is a special case of it. In the MCKP, given $m$ stages $N_1, ..., N_m$ of the workflow to pack in a knapsack of capacity $B$ (the budget constraint), where each code variant $j \in N_i, i = 1, 2, ..., m$, has execution time $T_{i,j}$ and cost $C_{i,j}$, the problem is to choose exactly one code variant from each stage such that the sum of execution times is minimized while the sum of costs does not exceed the budget $B$. The proposed formulation is the following (the decision variable is $x_{ij}$):

\[
\min \sum_{i=1}^{m} \sum_{j \in N_i} T_{i,j} * x_{i,j}
\]

Subject to:

\[
\sum_{i=1}^{m} \sum_{j \in N_i} C_{i,j} * x_{i,j} \leq B, \tag{1}
\]

\[
\sum_{j \in N_i} x_{i,j} = 1, i = 1, ..., m, \tag{2}
\]

\[
x_{i,j} \in \{0, 1\}, j \in N_i, i = 1, ..., m. \tag{3}
\]

### 2.3 Cost Models Tailored for Streaming Data Analytics in an Edge/Fog/Cloud Computing Setting

Hiessl et al. [15] aim to minimize response time, migration and enactment cost as well as to maximize availability in data stream processing systems by placing (and replacing) operators of stream topologies. However, the operator execution is not parallelized, i.e., each DAG node is placed on exactly one compute node. Their proposal targets fog environments, where heterogeneity and geo-distribution of the resources are inherent characteristics and they extend the system model and formulation presented in [9]. The adopted cost model represents both the resources and operators as graphs. In the former case, the links represent the network links where, in the latter case, the graphs refer to the flow of data streams between the operators. Each compute node has its CPU frequency, number of CPU cores, memory and storage capacity, whereas each operator has its own CPU frequency, number of CPU cores, memory and storage requirements. Also, for each operator, there exists a subset of the resources that are available to allocate its execution. The operators can be enacted and migrate from one compute node to another. The response time of a topology is equal to the maximum delay of all paths in the stream data flow, which depends on the
processing time of tuples and on the delay of sending the tuples. The modeling includes the $C_{op}(x)$ variable, which refers to the total enactment cost per second, which is the cost of running the topologies on the available resources. Another variable is $C_{mig}(x)$, which is the cost that comes from all the migrations of operators from one compute node to another and depends on the operator image size as well as the data rate of pulling this operator to a compute node.

In general, this model focuses on capturing the environment heterogeneity and then, in the subsequent optimization problem, the emphasis is placed on expressing several constraints.

Renart et al. [29] also place operators across edge and cloud devices in order to optimize metrics like end-to-end latency, WAN traffic and messaging cost. The setting they work on is geo-distributed with heterogeneous resources but the operators are not parallelized, as in the cost model above. Each resource is characterized by its CPU and memory capabilities ($cpu, mem$). Each network link $l_{k+l}$ has a certain bandwidth capability $bdw_{k+l}$ with other resources. Each operator has its own CPU and memory requirements ($cpu, mem$), selectivity, process rate $\mu_{<i,k>}$, input and output size $\varsigma_{i}^{in}, \varsigma_{i}^{out}$ and output production rate $\lambda_{i}^{out}$, which is based on its input rate $\lambda_{i}^{in}$. The resource and operator modeling is based on an M/M/1 queue model, which allows for calculating the computation and communication times. $stime_{<i,k>}$ gives the computation time of an operator $i$ placed on a resource $k$ and $ctime_{<i,k>,<j,l>}$ denotes the communication time for an operator $i$ placed on resource $k$ to send data to operator $j$ placed on resource $l$. The end-to-end latency $L_{pi}$ of a path $p_{i}$ is the sum of the computation times of all the operators in this path plus the communication times between them. The way the latency of each path is calculated in [29] is similar to that of [15] but the latter focuses on optimizing the maximum latency of all paths, whereas this work focuses on the sum of the latencies. The WAN traffic of a path $W_{pi}$ indicates the size of messages that cross it. The messaging cost of a path $C_{pi}$ shows the number of messages between edge and cloud. Finally the total aggregate cost model they use is the following: $AggregateCost_{pi} = w_{l} \cdot \frac{L_{pi}}{Par_{lat}} + w_{w} \cdot \frac{W_{pi}}{Par_{wan}} + w_{c} \cdot \frac{C_{pi}}{Par_{cost}}$, where $w_{l} + w_{w} + w_{c} = 1$ are weights and $Par_{lat}, Par_{wan}, Par_{cost}$ indicate the current latency, WAN traffic and messaging cost of the running application.

This cost model is characterized by the usage of queue theory and the fact that it aims to capture the environment heterogeneity, which is a main goal for cost models in geo-distributed settings. As previously, the final cost is a weighted sum of multiple individual costs, which are computed by simple summation expressions, given that these cost models are typically leveraged by multi-objective placement optimization solutions.

2.3.1 Optimization Problem Details

Hiessl et al. [15] place stream operators in fog computing environments. Both the computing resources and the application are presented using topology graphs. The aim is to map the operators and edges from the application graph to the resources and net-
work links of the resource graph, respectively, in order to minimize the following objective function:  

\[ F'_{\text{cost}} = F'(x, y, r) + w_{\text{Cmax}} \frac{C_{\text{opmax}} - C_{\text{opmin}}}{C_{\text{migmax}} - C_{\text{migmin}}} \]

where \( F'(x, y, r) = w_r R_{\text{max}} + \log A(x,y) - \log A_{\text{min}} \). The \( w \) variables are weights used in the simple additive weighting method that applies to multi-objective problems. The \( F'(x, y, r) \) variable focuses on response time and availability (\( R \) and \( A \) respectively).

The formulation of the problem is the following:

\[
\text{minimize } F'_{\text{cost}}
\]

Subject to:

\[
C_{\text{op}}(x) \leq B_{\text{op}} \quad (1)
\]

\[
C_{\text{mig}}(x) \leq B_{\text{mig}} \quad (2)
\]

\[
\sum_{i \in \text{operators}} T_{(\text{actual},i)} S_i \leq T_{(\text{max},i)} \quad \forall i \in \text{operators} \quad (3)
\]

\[
\sum_{i \in \text{resources}} P_{(\text{CPU},i)} P_{(\text{Cores},i)} x_{i,u} \leq P_{(\text{CPU},u)} P_{(\text{Cores},u)} \quad \forall u \in \text{resources} \quad (4)
\]

\[
\sum_{i \in \text{resources}} P_{(\text{Mem},i)} x_{i,u} \leq P_{(\text{Mem},u)} \quad \forall u \in \text{resources} \quad (5)
\]

\[
\sum_{i \in \text{resources}} P_{(\text{HD},i)} x_{i,u} \leq P_{(\text{HD},u)} \quad \forall u \in \text{resources} \quad (6)
\]

\[
r \geq \sum_{k=1}^{n_p} \frac{ET_i}{S_u} x_{i,k} + \sum_{k=1}^{n_p-1} \sum_{(u,v) \in \text{available_resources}} d_{(u,v)} y(i_k,i_{k+1},(u,v)) \quad \forall p \in \text{paths} \quad (7)
\]

Equations (1) and (2) limit the enactment and migration cost using budget constraints \( B_{\text{op}} \) and \( B_{\text{mig}} \), respectively. Equation (3) ensures that the processing duration of an operator assigned to a resource should not exceed the maximum processing time limit; in the equation, \( S_i \) denotes the speed of processing of operator \( i \) in previous optimization circles and \( S_u \) the speed of processing of resource \( u \). Equations (4), (5) and (6) ensure that the CPU, Memory and storage requirements of the operators are fulfilled while not exceeding the capacities of the resources. Finally, equation (7) ensures that the total response time is greater than or equal to the response times of all the topology paths; \( d \) denotes the network delay and \( ET \) the execution time per tuple. To avoid quadratic or higher order equations, the authors skip the multiplication of the decision variables \( x_{i,u} \) and \( y(i,j),(u,v) \) which refer to the placement of operators and links, respectively.

Renart et al [29] place operators of IoT dataflows on either edge or cloud resources in order to minimize end-to-end latency, data transfer and messaging cost between the edge and the cloud. The aggregate cost formula for a single path has been presented above. The formulation of the problem they try to solve that is given below refers to the aggregation costs across all paths:
minimize $\sum_{p \in P} \text{AggregateCost}_p$

Subject to:

\[ \lambda^\text{in}_o < \mu_{<o,r>} \quad \forall o \in \text{operators}, \forall r \in \text{resources} | mo_{<o,r>} = 1 \quad (1) \]

\[ \lambda^\text{in}_o < \left( \frac{\text{bw}_{k \leftrightarrow l}}{s_{o'-1}} \right) \quad \forall o \in \text{operators}, \forall k \leftrightarrow n \in \text{links} | mo_{<o,k>} = 1 \quad (2) \]

\[ \sum_{o \in O} \text{mo}_{<o,r>} * \lambda^\text{in}_o \leq \text{cpu}_r \quad \forall r \in \text{resources} \quad (3) \]

\[ \sum_{o \in O} \text{mo}_{<o,r>} * \text{mem}_{<o,r>} \leq \text{mem}_r \quad \forall r \in \text{resources} \quad (4) \]

\[ \sum_{s_{i \rightarrow j} \in S} \text{ms}_{<i \rightarrow j,k \leftrightarrow l>} * s^\text{out}_i \leq \text{bw}_{k \leftrightarrow l} \quad \forall k \leftrightarrow l \in \text{links} \quad (5) \]

\[ \sum_{r \in R} \text{mo}_{<o,r>} = 1 \quad \forall o \in \text{operators} \quad (6) \]

\[ \sum_{k \leftrightarrow l \in L} \text{ms}_{<i \rightarrow j,k \leftrightarrow l>} = 1 \quad \forall s_{i \rightarrow j} \in \text{streams} \quad (7) \]

The variable mo_{<o,r>} is set to 1 if operator o is placed on resource r and, similarly, ms_{<i \rightarrow j,k \leftrightarrow l>} = 1 if the stream between operators i, j is assigned on the link between resources k, l. Equations (1) and (2) ensure that the input rate does not exceed the process rate of the resource and that the link is not saturated, respectively. Equations (3) and (4) ensure that the CPU and memory requirements of each operator are fulfilled and the resource capacities are not exceeded. Equation (5) ensures that the bandwidth limits of the links are not exceeded. Finally, equations (6) and (7) ensure that an operator is placed on only one resource and that each data stream is placed on only one link, respectively.

### 2.3.2 Tractability

In both [15] and [29], the problem is more complex than the problem of choosing among multiple execution engines, when the placement decision of a node affects the other placements. This problem is analyzed in [22], and is proven that it is NP-hard, and moreover, "cannot be approximated by a polynomial-time algorithm with approximation error bound less than 8/7." Also, evidence about the NP-hardness is provided in [9]. As shown from the problem formulations, such task placement problems are specific forms of constrained optimizations (mixed) integer linear programming problems.
2.4 A Cost Model Tailored to Stage-by-stage Analytics in a Cloud Computing Setting

Gounaris et al. [13] propose a bi-objective cost model in order to optimize queries in multi-cloud environments. The queries are in a form of DAGs and are divided into strides where each stride represents a step in the execution and consists either of one operator or a group of operators running in parallel. However each operator cannot be parallelized. The goal is to assign strides to cloud-based VMs. Each cloud provider offers VMs to users and charges based on one of the following policies: on-demand, where the payment is computed based on the exact usage time, reserved, where the user pays upfront for a specific time and then pays on-demand, spot instance, which is a bidding process or committed, where the customer rents the VM for a predefined time. The model they present estimates the execution time and monetary cost and models the charging policies and fees. The basic formula for modeling the execution time is the following:

\[ \text{TotalTime} = \sum_{s=1}^{n} \max(\sum_{i=1}^{m_s} S_{s,i \rightarrow j}), \ldots, \sum_{i=1}^{m_s} S_{s,m_s \rightarrow j}) \]

where \( n \) is the number of strides, \( m_s \) the number of operators in the \( s \)th stride and \( S_{s,i \rightarrow j} \) is the time it takes to transfer data produced from operator \( i \) which runs on \( VM_k \) to operator \( j \) on \( VM_{k'} \).

In case there exist network bottlenecks, the total time is not the max of the sum of execution and transmission time of each operator but rather the sum of them as follows:

\[ \text{TotalTime} = \sum_{s=1}^{n} \sum_{i=1}^{m_s} S_{s,i \rightarrow j} \]

In case pipelining is supported, an operator can start transmitting data before its execution is complete. Then the next formulation holds:

\[ \text{TotalTime} = \sum_{s=1}^{n} \sum_{i=1}^{m_s} S_{s,i \rightarrow j} \]

2.5 A Cost Model for (Incremental) MapReduce Jobs

Li et al. [23] propose a system built on top of incremental Hadoop that allocates resources and uses runtime scheduling techniques in order to meet user latency requirements while maximizing throughput. The queries are represented using DAGs where each node is either a data distributor or a Map Reduce pair. They model the mean and the variance of latency both per-tuple and per-window. Regarding the per-tuple latency, G/G/1 queues are used to represent streams. The total latency and variance are a sum of latencies and variances due to 12 causes including batching, queuing, heartbeats awaiting, CPU, network
and Disk I/O. The mean CPU latency per tuple is formulated as follows:

\[ E(\tilde{L}_{\text{cpu}}) = \frac{u}{2} \times \min\{1 - \frac{p}{n}, 1/n\} \times C, \]

where \( u \) is the total resource required by a batch and depends on the average number of tuples and the average resource required per tuple. Also, \( p \) is the fraction of the resource required by the other threads on the same node since resources receive multiple map-reduce tasks, \( n \) is the number of cores of a CPU and \( C \) is the number of processing cycles of a CPU per unit time. Network and Disk I/O are modeled in a similar way. The queuing delay can be calculated based on the theory of the G/G/1 queues. The per-window latency is modeled in cases where the results of a window are computed only when a reducer has received all the tuples and is equal to \( E(\tilde{L}) = E(U) + E(F) \) where \( U \) is the maximum of all the tuples’ latencies in this window and \( F \) represents the execution of a particular partitioned window.

In summary, this cost model focuses on splitting the latency factors of a tuple processed by a MapReduce round into 12 parts, and takes into account resource sharing.

2.5.1 Optimization Problem Details and Tractability

The work in Li et al. \[23\] decides the amount of resources per Map/Reduce solving a non-linear constrained optimization problem (using MinConNLP). In the generic case, these problems are intractable. The main focus is on scheduling the tuples through re-ordering their execution on-the-fly.

2.6 Summary of Techniques

We summarize the techniques examined thus far in Table 1. We also provide an overview of their main limitations, which call for the development of a new cost model, tailored to massively parallel complex streaming analytics jobs over heterogeneous geo-distributed nodes; this cost model is presented in Section 3. The cost models in \[37, 20, 4\] do not cover heterogeneity in compute node characteristics and/or locations. The models in \[13, 29, 13\] do not account for massive parallelism. Therefore, they are inadequate and unsuitable to be adopted for our case, which focuses on complex DAGs that are executed over geo-distributed heterogeneous resources while benefiting from partitioned parallelism.

2.7 Additional Issues

We have already identified that the existing cost models fall short in supporting the use cases targeted hereby. For completeness, in this section, we examine more cost modelling approaches and we also consider scheduling problems.

2.7.1 Other Modeling Approaches

Karnagel et al. \[19\] estimate the performance of tasks in order to achieve an optimal adaptive placement of OLAP queries on heterogeneous hardware. To estimate the query runtime, they use an automated black-box online-learning approach. They use monitoring for execution times as well as input data cardinalities. To estimate the transfer costs, they
| Work          | Setting                                                                 | Objective                      | Resources’ metadata required                                                                 | Operators’ metadata required                                                                 |
|--------------|-------------------------------------------------------------------------|--------------------------------|------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| Zhang et al. | Multicore shared-memory architectures: decide placement of operators to sockets and degree of parallelism | Processing Throughput          | CPU cycles per socket, DRAM bandwidth, bandwidth between sockets, memory access latency, cache line size | Memory bandwidth consumption, execution time, average size of tuples, input stream rate       |
| Kougka et al. | Distributed multicore machines: decide operator ordering                | Response Time                  | Communication cost, number of cores                                                           | Execution cost, selectivity                                                                    |
| Hiessl et al. | Edge/Fog environment: decide placement of operators on compute nodes   | Response Time                  | Availability, delay of network links, speedup, CPU frequency, number of CPU cores, memory and storage capacity per compute node | Speedup from previous run, image size, requirements for CPU frequency, number of CPU cores, memory and storage per operator |
| Renart et al. | Edge/cloud environment: decide placement of operators on compute nodes | End-to-end latency, WAN traffic, messaging cost | CPU and memory capacities, bandwidth costs                                                    | CPU and memory requirements, selectivity, input/output event rate, input/output event size    |
| Gounaris et al. | Multi-cloud environments: decide placement of operators on compute nodes | Execution time, monetary cost | Charging policy, CPU, I/O access speed                                                        | Size of data                                                                                 |
| Ali et al.    | Multicore machines: decide degree of parallelism                        | Execution time, monetary costs | Configuration, row and byte process rate                                                      | Size of data, memory, I/O and CPU usage                                                       |
| Li et al.     | Multicore machines running MR: decide degree of parallelism (amount of resources) and scheduling | Latency, throughput            | CPU, disk I/O and network characteristics                                                     | Estimated data rate, resource required and input-to-output ration                            |

Table 1: Summary of techniques examined.

rely on statistics collected in the calibration phase rather than performing monitoring. In general, learning task costs using (machine)-learning approaches is quite common and is orthogonal to how individual costs are combined to estimate the cost of the complete DAG.

Learning methods can also apply to the whole DAG. More specifically, Witt et al. [36] survey predictive performance modeling (PPM) techniques that estimate metrics like
execution time, required memory and predict future wait times of tasks and jobs in data-parallel and distributing systems. They compare the works by the performance factors they consider; that is workload (resource usage patterns), heterogeneity (of resources), scale (amount of resources and problem size) and contention (caused by resource sharing). They study both task performance models and job performance ones. The representative works we present from this survey here consider the performance factors of heterogeneity and scalability. In the first category (namely, task performance models), a single task runs on a single node. Marin et al. [24] predict the performance of tasks, the cost of the computation and the execution duration using memory access patterns that are monitored using histograms, the cache size and input size. Matsunaga et al. [25] predict execution duration, output file size and resident set size of a task using regression trees and characteristics like CPU speed, cache size, available memory and disk capabilities, number of threads as well as location and size of input. Iverson et al. [17] predict the execution duration of a task using the input size and hardware characteristics and a local learning approach. The second category refers to approaches that deal with multiple nodes that comprise a job. Delimitrou et al. [11] predict the performance by observing other applications with similar resource usage patterns and make use of Singular Value Decomposition to classify incoming jobs. Shanjay et al. [30] use regression to model the computation duration and the non-overlapped communication duration, whereas the total execution duration is equal to their sum. Cunha et al. [10] aim to predict performance in order to choose between cloud and a local cluster. To achieve this, they propose a linear model and an empirical one that favors local execution when the prediction is unclear. Last, Alipourfard et al. [5] propose an off-line search method that uses a Gaussian model to select cloud configurations like number of virtual machines, number of cores, CPU speed, RAM per core etc in order to optimize the performance of a job.

Finally, the work of Agrawal et al. [2] selects the task ordering and the assignment of tasks to compute nodes, while it targets both the bottleneck cost metric (called period) and the critical path cost (which is equivalent to the latency); this work considers precedence constraints between the tasks in the DAG but the exact ordering is flexible. Each task has a cost and a selectivity (selectivities are independent). Also, there is a set of servers acting as the compute nodes. The focus is on both homogeneous and heterogeneous platforms (heterogeneity refers to network speeds) but not on geo-distributed settings. The communication speed is considered in some flavors. If there are no communication costs, for the cases of homogeneous machines, Agrawal et al. present algorithms that perform optimal ordering in polynomial time. They also prove that if the machines are heterogeneous, the problems are NP-hard even when no precedence constraints exist. On the other hand, if there are communication costs, they consider the models of overlapping (communication fully overlaps with computation), and sequential (no overlap) processing. The problem is always NP-complete, apart from the case of period minimization and overlap. In general, in this work the cost models are simple and the value lies in the algorithmic analysis and solutions, which however refer to a more restricted setting than the one assumed in this report.
2.7.2 Cost Models in Scheduling Works

We organize the discussion in this section around five recent surveys in the area of scheduling.

First, Hong et al. \cite{16} review algorithms, infrastructures and architectures for resource management on edge/fog computing. Infrastructure refers to hardware, software and middleware. The architectures are divided into data flow, control and tenancy ones and they present data flow architectures that provide techniques which take into consideration bandwidth, latency, energy efficiency, quality, security and heterogeneity in order to reduce the communication cost. Regarding the algorithms, they divide them into four categories; discovery, benchmarking, load-balancing and placement. Discovery algorithms aim to identify edge resources to use for deployment of applications. Benchmarking focuses on capturing the actual performance (CPU, memory, storage) using stress tests and tools. Load-balancing aims to distribute tasks using several approaches, such as optimization techniques, graph-based balancing, breadth-first search or cooperative load balancing. Finally, placement algorithms assign tasks onto resources and can be divided into dynamic, condition-aware and iterative ones.

The placement techniques in \cite{16} are summarized as follows. Wang et al. \cite{35} propose off-line and on-line techniques for finding an optimal placement of task (service) instances on Mobile Micro Clouds in order to minimize the average cost while predicting future cost parameters with known accuracy. The off-line algorithm finds the optimal placement of services within a predefined time-window whereas the on-line approximation one makes placement decisions upon arrival of the services. Taneja et al. \cite{33} map IoT application jobs on fog resources by ordering both based on their CPU, RAM and bandwidth requirements and capacities, respectively. In case, it is not feasible to map jobs to fog nodes, these are delegated to cloud resources. Souza et al. \cite{32} propose three algorithms, namely First-Fit, Best-Fit and Best-Fit with Queue, for placing services, which are divided into smaller atomic-services that form DAGs, on fog or cloud nodes. First-fit assigns services to the first available fog node, Best-fit first orders the services based on their requirements and then assigns them. Both these algorithms assign services to the cloud, in case no available fog node exists, whereas Best-fit with Queue puts these services in queues while waiting for fog nodes to become available by predicting if that will be more beneficial. Fernando et al. \cite{12} propose Honeybee, a model for load balancing in a group of heterogeneous mobile nodes by using the work-stealing method. The independent jobs are pulled by the mobile sources instead of being pushed to them and, additionally, a node can “steal” jobs from another node. Also, this work supports the mobility of the devices, in the sense that they can join or leave the group at any time. Skarlat et al. \cite{31} place IoT services on fog nodes taking into account their QoS requirements (e.g. execution deadlines). They organize the fog devices, called fog cells, into fog colonies, which act as data centers. Each colony has a control node used for management and communication with the other colonies and the cloud. The placement problem is formulated as an Integer Linear Programming one. Ni et al. \cite{27} propose a dynamic resource allocation strategy for fog computing based on Priced
Time Petri Nets (PTPN) in order to improve the resource utilization while satisfying the users’ QoS requirements. The authors also propose algorithms to predict the time cost and price cost of the tasks as well as compute the credibility evaluation of resources. The users can select the resources from a set of provided resources based on these characteristics. All these proposals do not account for multiple compute nodes co-operating for processing the same task in complex DAGs.

In a second survey, Beaumont et al. [7] present and compare algorithms that focus on scheduling tasks on CPU and GPU resources. The scheduling algorithms decide which task to execute first, when to start the execution and where to allocate it. They provide performance guarantees in the form of lower makespan bounds. They categorize the algorithms as on-line and off-line and the tasks as independent or with precedence constraints. In the on-line scenario they prove lower bound conditions, while, in the off-line algorithms, a linear programming problem is solved to keep the workload on CPU and GPU processors under a defined threshold and also to satisfy any precedence relations between tasks. Nevertheless, these solutions cannot be transferred to massively parallel (streaming) settings, while also heterogeneity of inter-node communication speeds are not considered. On the other hand, online scheduling techniques can inspire adaptive resource placement policies in the context of our project.

The third survey considered is authored by Barika et al. [6], who review the challenges in orchestrating big data analysis workflows. First, they present the different programming models that exist, namely MapReduce, Stream, NoSQL, SQL, Message Ingestion and Hybrid. They also provide a taxonomy for the challenges, which can be divided into three categories: cloud-based, data-based and workflow-based ones as depicted in Figure 1.

![Workflow orchestration challenges taxonomy proposed in [6].](image)
Next, they focus on some of the current approaches and techniques for the workflow-based challenges. In a nutshell, according to their discussion, the workflow specification language can be generic or custom with the latter limiting the ease to use multiple execution environments. The initialization of workflow execution aims to partition the workflow into fragments. This can be done by taking into consideration constraints like security, privacy, storage and compute capacities, data transfer constraints, or a combination of them or taking into consideration only the task and data dependencies within the workflow. Parallelization can be coarse-grained focusing on workflow level (when dealing with multiple workflows) or fine-grained focusing on activity/task level. Scheduling can be push-based, which aims to schedule tasks to available workers or pull-based, where the workers request for the tasks they will process. The former can be further categorized into static, dynamic or hybrid. Workflow fault-tolerance can be achieved either through reactive or proactive techniques. Finally the security challenge can be tackled by using multi-cloud architectures or replication-level techniques. No cost models are explicitly discussed, but the significance to cover issues such as resource, storage and data transmission characteristics is mentioned.

Adhikari et al. [1] survey scheduling strategies for workflows in cloud environments. They offer equations and definitions for multiple optimization objectives. The execution time of a task is equal to \( ET_{ki} = \frac{SZ_k}{CP_i} \) where \( SZ_k \) is the size of the task and \( CP_i \) the capacity of the cloud VM instance which is the product of the number of cores and the size of each core (in MIPS or MFLOPS). The completion time of a task is either equal to its execution time or equal to the sum of its execution and waiting times. The waiting time for a task is 0 or equal to the maximum completion time of its predecessor tasks. The execution cost of a task refers to the cost of running on a VM instance and is equal to \( \frac{ET_{ki}}{\tau} \) where \( \tau \) is the time period the task used the VM and \( CO_i \) represents the cost of the VM. The energy consumption of a task refers to the amount of energy required to run on a VM instance and is equal to \( E_{ik} = DP_{ik} + SP_{ik} \), that is the sum of the dynamic and static power consumption. The makespan (total execution time) of a workflow is the maximum completion time of its tasks, the total execution cost is the sum of the execution costs of all the tasks and similarly the total energy consumption is the sum of the energy consumption of all the tasks. Some of the most common scheduling objectives are total execution time (makespan), total execution cost, schedule length ratio (SLR-the ratio of the completion time of a task to the total execution time of the workflow), availability, reliability, energy consumption, resource utilization, communication overhead and security. The authors also discuss some of the emerging trends in workflow scheduling which include containers (e.g. Docker, Kubernetes), server-less computing (DynamoDB, Amazon SNS, Google cloud pub/sub) and fog computing. The latter environment also raises the need to consider the latency objective between two compute nodes \( i \) and \( j \), which is equal to \( LT_{i,j} = PD_{i,j} + SD_{i,j} \). \( PD_{i,j} = \frac{DS_{i,j}}{BW_{i,j}} \) is the propagation delay and depends on the distance between the nodes \( DS_{i,j} \) and the bandwidth of the network \( BW_{i,j} \). \( SD_{i,j} = \frac{SZ_{i,j}}{TR_{i,j}} \) is the serialization delay and depends on the size of the workflow in bits and the transmission rate of the network. However, the content of this survey is not suitable for our setting,
because it does not account for placement for complex streaming applications, where all DAG nodes are active simultaneously and partitioned parallelism is employed.

Finally, Salaht et al. survey techniques for the service placement problem in Fog Computing environments. They characterize the works as being centralized or decentralized, off-line or on-line, static or dynamic and as to whether they support mobility of the devices or not. Furthermore, they focus on the optimization problems (multi- vs. mono-objective), the metrics considered (latency, resource utilization, cost, energy consumption, congestion ratio, etc.), the formulation of the problem (integer linear programming, integer non-linear programming, mixed integer linear programming, mixed integer quadratic programming) and finally the resolution strategies (exact solution, heuristics, meta-heuristics, approximations). In terms of modeling, the constraints to be covered include resource, network and applications (e.g., locality of a task placement) ones. The main limitation of the techniques examined that account for complex DAGs to be applicable to our setting is that they do not consider massive parallelism.
3 The Proposed Cost Model Proposal

The cost model we propose is tailored to streaming data scenarios focusing on geo-distributed, heterogeneous resources. The heterogeneity apart from network speeds also refers to hardware characteristics, like CPU and RAM. Also we consider the massive parallelism of the operators. Since it refers to a streaming case, in terms of performance, the focus is on latency. The cost model extends the one presented in [14] which however, apart from smaller differences, does not cover massive parallelism on top of arbitrary resources.

Table 2 presents the main notation used. Specifically, we represent a streaming analytics job as a Directed Acyclic Graph (DAG) $G_{op} = (V_{op}, E_{op})$ where each node in $V_{op}$ represents an operator and the edges in $E_{op}$ represent the data flow between operators. An operator represents a set of analysis job steps at a fine level of granularity that can run on the same device (typically in a pipelined manner). Each operator can be physically partitioned into multiple instances, where each instance is responsible for a disjoint data partition. Edges represent data re-distribution among the partitioned operator instances. For example, according to the above description, in Apache Spark, operators correspond to Spark job stages while edges are placed when data shuffling takes place. The data sources are the operators in the $G_{op}$ without incoming edges, and we assume that they produce data in batches periodically.

Each operator takes as input multiple tuples (i.e., records) in batches that need to be analyzed and also has a certain selectivity $s_i$ according to its functionality. For example, a transformation operator with selectivity $s_i = 1$ outputs a tuple for each tuple it takes as input, while a filtering operator with $s_i = 0.5$ outputs a tuple for every two input tuples on average. If operator $i$ is a sink node in the graph, then, there is no selectivity, whereas, if it is a source node, its selectivity is equal to 1. Each pair of edge devices has a different communication cost, $comCost_{u,v}$ ($u, v \in ED$), which relates to the connected devices network speed; the latter is also dependent on the physical distance and link capacity.
between the devices.

Each operator is assigned to multiple edge devices that run in parallel. That means that each device \( u \) is assigned a fraction \( x_{i,u} \) of tuples of an operator \( i \) to analyze. However, due to privacy, security and other reasons, there exists a subset of edge devices where an operator can be assigned to (denoted by the flag \( \text{available}_{i,u} \) for operator \( i \) regarding device \( u \)). We define as \( ED_i \subset ED \) the subset of edge devices operator \( i \) can be assigned to.

As already stated, the QoS metric we try to minimize is the average latency. This latency is equal to the latency of the critical path (i.e., the slowest path) with regards to a single input data batch and consists of the average communication latency between the operators in the critical path. In order for the critical path’s latency to be representative of the whole graph’s latency, we assume there is no waiting time between the tasks of the operators. This is achieved through setting the operator requirements at such levels, so that they sustain the input data production rate. A natural consequence of this is that each operator can start its execution as soon as it receives its input batch of tuples from its parent nodes in the graph. As our focus is on geo-distributed realms, we make the realistic assumption that the execution latency of each operator is negligible and the communication cost dominates. The total latency of a tuple is the time to flow from its source node downstream to a sink node.

The communication latency between two nodes is expressed by

\[
\max \{x_{i,u} \cdot s_i \cdot \sum_{v \in ED_j} \text{comCost}_{u,v} \cdot x_{j,v}\}, \quad u \in ED_i \text{ across all instances of operator } i.
\]

This is equal to the slowest data transfer that comes from a single device and refers to the cost of a batch of input data. However, this modeling is over-simplistic because it does not take into account the overhead of an operator instance maintaining multiple remote connections. In order to take into account such an overhead, we introduce a parameter, notated as \( \alpha \) multiplied by the number of enabled links. A link between device \( u \) and device \( v \) for two operators \( i, j \) is enabled when \( x_{i,u} \neq 0, x_{j,v} \neq 0 \) and \( u \neq v \). Thus, \( \text{enabledLinks}_{i,j} \) denotes the number of devices that exchange data between two operators over the network. The total latency of the topology is equal to the latency of the critical path, that is slowest path of the graph that leads from a source node to a sink one (not including). We denote as \( \text{path} \) any path from a source to an operator just upstream a sink operator (\( j \) is the operator just after \( i \)).

\[
\text{Latency} = \max_{\text{path} \in G_{op}} \left\{ \sum_{i \in V_{op} \in \text{path}} \max \left\{ x_{i,u} \cdot s_i \cdot \sum_{v \in ED_j} \left( \text{comCost}_{u,v} \cdot x_{j,v} \right) + \alpha \cdot \text{enabledLinks}_{i,j} \right\} \right\}
\]

### 3.1 Accounting for Additional Objectives

The above cost model targets a single objective. It is trivial, through simple sum functions to consider the movement of data over the network, as in [26], or the total time resources are occupied for processing a specific time period.

An important novelty, however, in modern dataflows if their focus on additional objectives, such as quality optimization. The quality of the data is an important aspect in IoT scenarios. Low quality can lead to less useful results. The quality of data can be catego-
| Device | 0  | 1  | 2 |
|--------|----|----|---|
| 0      | 0  | 1.5| 2 |
| 1      | 1.5| 0  | 1 |
| 2      | 2  | 1  | 0 |

Table 3: Communication cost between edge devices in GBps.

| operator/device | 0  | 1  | 2 |
|-----------------|----|----|---|
| 0               | 0.8| 0.2| 0.0|
| 1               | 0.7| 0.0| 0.3|
| 2               | 0.3| 0.4| 0.3|

Table 4: Fraction of operator assigned to each edge device.

...
graph depends on the selectivity of that operator, the communication cost of the devices and the assigned fractions. Each device sends data to every other device based on the fraction of the current and the succeeding operator. For example, for operator 0, device 0 will take \(0.8 \times 1 \times 0 \times 0.7 \text{ sec/unit}\) to send data to device 0 (itself), \(0.8 \times 1 \times 1.5 \times 0 \text{ sec/unit}\) to device 1 and \(0.8 \times 1 \times 2 \times 0.3 \text{ sec/unit}\) to device 2. We sum these cost elements to derive the total cost of communication between operator 0 and operator 1 for the first device with id 0; this sum is 0.48. For device 1, we have \(0.2 \times 1 \times (1.5 \times 0.7 + 0 \times 0 + 1 \times 0.3) = 0.27 \text{ sec/unit}\). For device 2, the communication cost is 0. Therefore, the latency due to the link 0 \(\rightarrow\) 1 is \(\max\{0.48, 0.27, 0\} = 0.48\). Similarly, we can derive that the cost between 1 \(\rightarrow\) 2 is \(\max\{1.26, 0, 0.45\} = 1.26 \text{ sec/unit}\). Therefore, the overall latency is 1.74 sec/unit.

If we assume that the \(DQ_{fraction}\) is 0.5 then with \(\beta = 1\), the objective function \(F\) becomes 1.16. Further assume that we examine another scenario, where \(DQ_{fraction}\) is 1 at the expense of moving the complete fraction \(x_{2,0}\) to device 2, i.e., the last operator runs 40\% on device 1 and 60\% on device 2. Then, the latency cost of 1 \(\rightarrow\) 2 becomes \(\max\{1.89, 0.18\} = 1.89\) and the complete latency is 2.37. The new value of \(F\) is 1.185, i.e., despite the important increase in latency, the new plan is not better in terms of \(F\). However, if we give even more weight to the data quality through increasing the \(\beta\) value, e.g., setting \(\beta = 2\), then the initial and modified allocation yield \(F\) values of 0.87 and 0.79, respectively, i.e., the trade-off in the modification has become beneficial.
4 Summary

In this report, we first describe existing cost modeling approaches highlighting their inadequacy to cover all aspects relevant to a modern setting including heterogeneity in compute node characteristics and network connections, massive parallelism, and complex DAG-shaped streaming application execution plans. In addition, we briefly discuss tractability issues related to the optimization problems associated with cost model proposals. To cover the aforementioned gap, in Section 3, we propose a new cost model, which also accounts for data quality.

Acknowledgment. The research work was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the “First Call for H.F.R.I. Research Projects to support Faculty members and Researchers and the procurement of high-cost research equipment grant” (Project Number:1052, Project Name: DataflowOpt).
References

[1] Mainak Adhikari, Tarachand Amgoth, and Satish Narayana Srirama. A survey on scheduling strategies for workflows in cloud environment and emerging trends. *ACM Comput. Surv.*, 52(4):68:1–68:36, 2019.

[2] Kunal Agrawal, Anne Benoit, Fanny Dufossé, and Yves Robert. Mapping filtering streaming applications. *Algorithmica*, 62(1-2):258–308, 2012.

[3] Farah Aït-Salaht, Frédéric Desprez, and Adrien Lebre. An overview of service placement problem in fog and edge computing. *ACM Comput. Surv.*, 53(3):65:1–65:35, 2020.

[4] Syed Muhammad Fawad Ali and Robert Wrembel. Towards a cost model to optimize user-defined functions in an ETL workflow based on user-defined performance metrics. In Tatjana Welzer, Johann Eder, Vili Podgorelec, and Aida Kamisalic Latific, editors, *Advances in Databases and Information Systems - 23rd European Conference, ADBIS 2019, Bled, Slovenia, September 8-11, 2019, Proceedings*, volume 11695 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2019.

[5] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In Aditya Akella and Jon Howell, editors, *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 469–482. USENIX Association, 2017.

[6] Mutaz Barika, Saurabh Kumar Garg, Albert Y. Zomaya, Lizhe Wang, Aad van Moorsel, and Rajiv Ranjan. Orchestrating big data analysis workflows in the cloud: Research challenges, survey, and future directions. *ACM Comput. Surv.*, 52(5):95:1–95:41, 2019.

[7] Olivier Beaumont, Louis-Claude Canon, Lionel Eyraud-Dubois, Giorgio Lucarelli, Loris Marchal, Clément Mommessin, Bertrand Simon, and Denis Trystram. Scheduling on two types of resources: A survey. *ACM Comput. Surv.*, 53(3):56:1–56:36, 2020.

[8] Jen Burge, Kamesh Munagala, and Utkarsh Srivastava. Ordering pipelined query operators with precedence constraints. Technical Report 2005-40, Stanford InfoLab, 2005.

[9] Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti, and Matteo Nardelli. Optimal operator placement for distributed stream processing applications. In Avigdor Gal, Matthias Weidlich, Vana Kalogeraki, and Nalini Venkatasubramanian, editors, *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, DEBS ’16, Irvine, CA, USA, June 20 - 24, 2016*, pages 69–80. ACM, 2016.
[10] Renato Luiz de Freitas Cunha, Eduardo R. Rodrigues, Leonardo P. Tizzei, and Marco A. S. Netto. Job placement advisor based on turnaround predictions for HPC hybrid clouds. Future Gener. Comput. Syst., 67:35–46, 2017.

[11] Christina Delimitrou and Christos Kozyrakis. Quasar: resource-efficient and qos-aware cluster management. In Rajeev Balasubramonian, Al Davis, and Sarita V. Adve, editors, Architectural Support for Programming Languages and Operating Systems, ASPLOS ’14, Salt Lake City, UT, USA, March 1-5, 2014, pages 127–144. ACM, 2014.

[12] Niroshinie Fernando, Seng W. Loke, and Wenny Rahayu. Computing with nearby mobile devices: A work sharing algorithm for mobile edge-clouds. IEEE Trans. Cloud Comput., 7(2):329–343, 2019.

[13] Anastasios Gounaris, Zisis Karampaglis, Athanasios Naskos, and Yannis Manolopoulos. A bi-objective cost model for optimizing database queries in a multi-cloud environment. J. Innov. Digit. Ecosyst., 1(1-2):12–25, 2014.

[14] Lin Gu, Deze Zeng, Song Guo, Yong Xiang, and Jiankun Hu. A general communication cost optimization framework for big data stream processing in geo-distributed data centers. IEEE Trans. Computers, 65(1):19–29, 2016.

[15] Thomas Hiessl, Vassileios Karagiannis, Christoph Hochreiner, Stefan Schulte, and Matteo Nardelli. Optimal placement of stream processing operators in the fog. In 3rd IEEE International Conference on Fog and Edge Computing, ICFEC 2019, Larnaca, Cyprus, May 14-17, 2019, pages 1–10. IEEE, 2019.

[16] Cheol-Ho Hong and Blesson Varghese. Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms. ACM Comput. Surv., 52(5):97:1–97:37, 2019.

[17] Michael A. Iverson, Füsun Özgüner, and Lee C. Potter. Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. IEEE Trans. Computers, 48(12):1374–1379, 1999.

[18] Aimad Karkouch, Hajar Mousannif, Hassan Al Moattasime, and Thomas Noel. Data quality in internet of things: A state-of-the-art survey. Journal of Network and Computer Applications, 73:57 – 81, 2016.

[19] Tomas Karnagel, Dirk Habich, and Wolfgang Lehner. Adaptive work placement for query processing on heterogeneous computing resources. Proc. VLDB Endow., 10(7):733–744, 2017.

[20] Georgia Kougka and Anastasios Gounaris. Optimization of data flow execution in a parallel environment. Distributed Parallel Databases, 37(3):385–410, 2019.
[21] Georgia Kougka, Anastasios Goumaris, and Alkis Simitsis. The many faces of data-centric workflow optimization: a survey. *Int. J. Data Sci. Anal.*, 6(2):81–107, 2018.

[22] Georgia Kougka, Anastasios Goumaris, and Kostas Tischlas. Practical algorithms for execution engine selection in data flows. *Future Gener. Comput. Syst.*, 45:133–148, 2015.

[23] Boduo Li, Yanlei Diao, and Prashant J. Shenoy. Supporting scalable analytics with latency constraints. *Proc. VLDB Endow.*, 8(11):1166–1177, 2015.

[24] Gabriel Marin and John M. Mellor-Crummey. Cross-architecture performance predictions for scientific applications using parameterized models. In Edward G. Coffman Jr., Zhen Liu, and Arif Merchant, editors, *Proceedings of the International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS 2004, June 10-14, 2004, New York, NY, USA*, pages 2–13. ACM, 2004.

[25] Andréa M. Matsunaga and José A. B. Fortes. On the use of machine learning to predict the time and resources consumed by applications. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGrid 2010, 17-20 May 2010, Melbourne, Victoria, Australia*, pages 495–504. IEEE Computer Society, 2010.

[26] Anna-Valentini Michailidou and Anastasios Goumaris. Bi-objective traffic optimization in geo-distributed data flows. *Big Data Res.*, 16:36–48, 2019.

[27] Lina Ni, Jinquan Zhang, Changjun Jiang, Chungang Yan, and Kan Yu. Resource allocation strategy in fog computing based on priced timed petri nets. *IEEE Internet Things J.*, 4(5):1216–1228, 2017.

[28] Christos H. Papadimitriou and Mihalis Yannakakis. Multiobjective query optimization. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 21-23, 2001, Santa Barbara, California, USA*. ACM, 2001.

[29] Eduard Gibert Renart, Alexandre Da Silva Veith, Daniel Balouek-Thomert, Marcos Dias de Assunção, Laurent Lefèvre, and Manish Parashar. Distributed operator placement for iot data analytics across edge and cloud resources. In *19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2019, Larnaca, Cyprus, May 14-17, 2019*, pages 459–468. IEEE, 2019.

[30] H. A. Sanjay and Sathish S. Vadhiyar. Performance modeling of parallel applications for grid scheduling. *J. Parallel Distributed Comput.*, 68(8):1135–1145, 2008.

[31] Olena Skarlat, Matteo Nardelli, Stefan Schulte, and Schahram Dustdar. Towards qos-aware fog service placement. In *1st IEEE International Conference on Fog and Edge Computing, ICFEC 2017, Madrid, Spain, May 14-15, 2017*, pages 89–96. IEEE Computer Society, 2017.
[32] Vitor Barbosa C. Souza, Xavier Masip-Bruin, Eva Marín-Tordera, Sergio Sánchez-López, Jordi Garcia, Guang-Jie Ren, Admela Jukan, and Ana Juan Ferrer. Towards a proper service placement in combined fog-to-cloud (F2C) architectures. *Future Gener. Comput. Syst.*, 87:1–15, 2018.

[33] Mohit Taneja and Alan Davy. Resource aware placement of iot application modules in fog-cloud computing paradigm. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, Portugal, May 8-12, 2017*, pages 1222–1228. IEEE, 2017.

[34] Efthymia Tsamoura, Anastasios Gounaris, and Kostas Tsichas. Multi-objective optimization of data flows in a multi-cloud environment. In *Proceedings of the Second Workshop on Data Analytics in the cloud*, pages 6–10, 2013.

[35] Shiqiang Wang, Rahul Urgaonkar, Ting He, Kevin Chan, Murtaza Zafer, and Kin K. Leung. Dynamic service placement for mobile micro-clouds with predicted future costs. *IEEE Trans. Parallel Distributed Syst.*, 28(4):1002–1016, 2017.

[36] Carl Witt, Marc Bux, Wladislaw Gusew, and Ulf Leser. Predictive performance modeling for distributed computing using black-box monitoring and machine learning. *CoRR*, abs/1805.11877, 2018.

[37] Shuhao Zhang, Jiong He, Amelie Chi Zhou, and Bingsheng He. Briskstream: Scaling data stream processing on shared-memory multicore architectures. In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 705–722. ACM, 2019.