A Bi-objective Cost Model for Database Queries in a Multi-cloud Environment

Zisis Karampaglis
Aristotle University
of Thessaloniki, Greece
zkarampa@csd.auth.gr

Anastasios Gounaris
Aristotle University
of Thessaloniki, Greece
gounaria@csd.auth.gr

Yannis Manolopoulos
Aristotle University
of Thessaloniki, Greece
manolopo@csd.auth.gr

ABSTRACT

Cost models are broadly used in query processing to drive the query optimization process, accurately predict the query execution time, schedule database query tasks, apply admission control and derive resource requirements to name a few applications. The main role of cost models is to produce the time needed to run the query on a specific machine. In a multi-cloud environment, this is insufficient in two aspects: firstly, the machines employed are not defined *a-priori*, and secondly, time estimates need to be complemented with monetary cost information, because both the economic cost and the performance are of primary importance. This work addresses these two shortcomings and aims to serve as the first proposal for a bi-objective query cost model that is suitable for queries executed over resources provided by potentially multiple cloud providers. Moreover, our approach is applicable to more generic data flow graphs, the execution plans of which do not necessarily comprise relational operators.

1. INTRODUCTION

More and more companies and organizations consider moving their infrastructures and applications on the cloud, motivated by the promise of clouds to achieve economies of scale. One of the most attractive features of cloud computing is that it provides an alternative to the procurement and management of expensive computing resources, which are associated with high upfront investments and considerable human effort, respectively.

Cloud technology has evolved significantly and nowadays, is considered as robust and trustworthy. It leverages several traditional notions of distributed computing, such as the virtualization of resources and the provision of virtual machines (VMs), typically at a certain monetary cost. Cloud resources are not limited to emulations of raw physical machines; they can also cover provision of software middleware, databases and specialized tools. The proliferation of cloud options has raised the following problem faced by cloud users: *which cloud providers to choose to execute a specific task on the cloud?* This issue is not only important but also complex, especially when the requested resources can be offered by multiple providers. A key point to answer this question is to provide estimates of both the running time and the monetary cost; this is exactly the topic of our work.

We focus on database queries and more generic dataflow tasks that can be executed over remote resources provided by multiple providers [20, 18]. For example, assume a database query that joins data from cloud-enabled data stores, such as anonymized population census data and commercial data offered by a set of providers. Or, analyzing patient data using a series of specialized cloud-enabled services, as described in [25]. In such scenarios, to be in a position to take final allocation decisions, we need to be able to accurately estimate the running time of the tasks on cloud resources and the price to use such resources.

Estimating query execution time plays an important role in several applications and processes, including query optimization, scheduling, admission control and allocation of resources [23]. Typically, the query cost models are applied to physical execution plans and assume that the physical resources to be employed in query execution are predefined. These cost models either encapsulate a component to estimate the cardinalities of the data processed or accept such cardinality statistics as input; in the output, they produce an estimate of the query running time. Examples of such cost models in a distributed environment are provided in [11, 21]. In a multi-cloud environment, providing information only about the running time for specific processors is insufficient because (i) the machines employed are not defined *a-priori*, and (ii) time estimates need to be complemented with monetary cost information. This work addresses these two shortcomings.

The main contribution of this work is the proposal of a database query cost model that provides estimates of both the expected running time and the economic cost associated with running a specific query over VMs provided by one or more cloud providers. The cost model is modular and can be applied to arbitrary DAG data flows apart from simple query execution plans consisting of relational operators. It supports the main modes of fee charging to date, which leverage the pay-as-you-go approach. Nevertheless, the modularity of our proposal allows for easily plugging-in further models for running time and cost estimates, while the model is not tailored to any specific charging policy. In this work, we show how our proposal can be used to derive running time and monetary cost estimates through a

MEDES'14, September 15-17, 2014, Buraidah Al Qassim, Saudi Arabia

detailed example and a validation case study on a real cloud infrastructure.

The remainder of this paper is structured as follows. In the next section, we provide background material and discuss related work. In Section 3, we give the details of our cost model. Section 4 deals with a validation case study. We conclude in Section 5.

2. BACKGROUND

Before delving into cost model's details, we provide a brief overview of the main pricing policies currently adopted by cloud providers and are meant to be supported by our cost model. We also discuss the main factors involved in the cost of developing and maintaining a cloud infrastructure. In the last part of this section, we present the currently established cost models for distributed queries, which do not consider economic costs. Finally, we show how our proposal complements bi-objective optimizers that are suitable for multi-cloud database queries.

2.1 Cloud Pricing Policies and Costs

Cloud providers offer VMs at a specific price. The price depends on several factors including the computational characteristics of the VM, the reservation time and mechanism, and whether the VM comes with specific software installed (e.g., as typically occurs in PaaS/SaaS settings) or not. The price of VMs typically differs among providers, even when the offered VMs share the same characteristics.

2.1.1 VM characteristics related to charging

A main characteristic that affects the charging fee is the exact type and volume of computational resources that each client requests. There is a significant deviation in the price depending on CPU speed[1], memory and storage space. Usually providers have some fixed combinations of the above components, so that they offer complete pre-specified VM options to users aiming to cover a broad range of needs. In addition, some providers allow their customers to build their own combination of resources, i.e., to customize their VMs. Some examples are Amazon Web Services [1] and CloudSigma [4], respectively. Finally, some providers, like Amazon [3] and Rackspace [14] follow a hybrid approach and offer additional storage space with extra cost on top of pre-specified VM instances.

Installed software is equally important. The software that is used (e.g., databases, operating systems etc) may be open-source or commercial. Generally, VMs with pre-installed open source software are less expensive than those with commercial software due to licensing fees. The built-in support for or the existence of programming frameworks (e.g. Apache Hadoop) increases the cost. The same holds for non-functional features, such as monitoring services, security features, ease of migration, and so on. Other price factors are the data transfer and geographical position of VMs. Some providers have additional charges for data transfer either from or to their servers and also apply different rates depending on the geographical location of the servers running the VMs (e.g., [2, 15]).

A cost model needs to be VM-independent to be usable in a multi-cloud environment. Our cost model is not tailored to any specific hardware characteristics. Rather, it provides generic formulas that can be calibrated according to the specific VM types at the disposal of a cloud client.

[1]CPU power is often abstracted through the use of the so-called ECU units.

2.1.2 Cloud Costs

The development and maintenance of cloud infrastructure requires the investment of a big amount of money that is amortized over a long time period. Except from the initial purchase costs, a cloud data center is expensive to maintain and run. The major development cost is the acquisition of the raw servers and network infrastructure. In addition, cloud data centers have high energy needs and require special power and cooling infrastructure, which is an extra cost [6]. Licenses for software, such as OS and virtualization software, are an additional expense. Finally, there is the cost for the real estate, where the data center will physically reside [10]. The most important economic cost related to maintenance cost is due to the significant power consumption. It is usually the 15-20% of the total budget [6, 7]. Other costs include the network expenses, which are the costs for communicating with the end users, and the salaries of the data center technicians and the rest of employees.

The cost of running a cloud infrastructure directly impacts on the charges requested by the end users for its usage. However, in our cost model, we do not deal with the issue of configuring the price of the VM options offered by the cloud providers.

2.1.3 Charging Models

The charging models are orthogonal to the VM characteristics and the costs for developing and running cloud infrastructures. Here, we review the most important charging models, which are all supported by our cost model.

The most common charging model is the *"Pay-as-you-go"* one, where the customer is charged for the actual period she uses the infrastructure. The usage periods are monitored in different granularities though; i.e., providers may have different minimum time unit for charging. For example, one provider's minimum time unit may be 1 hour and another's 5 minutes. So, if someone uses the former infrastructure for 1 hour and 23 minutes, she will be charged as if she used the infrastructure for 2 hours, whereas, in the latter case the price will be for 1 hour and 25 minutes. The *"Pay-as-you-go"* charging model is encountered in three main forms in Amazon EC2, but those forms are essentially generic to any cloud provider:

- *On-demand Instance*, where the payment is done after the use of infrastructure charging for as long as the customer used it, without any other commitment, as explained above.
- *Reserved Instance*, where the customer pays a small fee upfront for a specific time (i.e. month, year) and after that, she is charged like the on-demand policy for the time using the infrastructure, but with a great discount on the fee.
- *Spot Instance*, which is like an auction. The customer bids whatever price is willing to pay for the infrastructure and, if the bid is above the current spot price, she gets the VM and is charged for the actual usage period but with a very lower price. The drawback is that, if the spot price goes above the customers bid price, her VM will be shut down.

An additional charging model is the *"Committed VM"*. In this model, the client rents the infrastructure for a predefined time. This predefined time can be from one month to a year, or even more in some cases. During this time, the customer can use the infrastructure whenever she wants without any extra cost (except maybe network traffic). Usually, it is less expensive than *"Pay-as-you-go"* when the usage is high. An example of this model is GoGrid [5].

2.2 Existing Cost Models and Other Related Work

Cost models for distributed queries do not consider the economic cost associated and are limited to scenarios where the physical resources are predefined. A typical approach is presented in [11], where the query execution time is split into the time needed to execute CPU tasks, retrieve and store data to the disk and send data across hosts. [21] provides more detailed cost functions but follows the same approach as in [11]. More sophisticated approaches, such as [24, 16], perform more efficient and dynamic cost function calibration but still suffer from the main limitations mentioned above.

Finally, the work in [13] defines the tradeoff between performance and cost, when running an application over a different number of VMs of the same type in the same data center under volatile load. Our problem is different, since we consider cases where the applications consist of several subtasks that can run on different types of VMs, which are possibly provided by multiple providers.

2.3 How can the cost model be used?

Our setting is depicted in Figure 1. We assume that there exists a centralized optimizer that builds an execution plan in the form of a query tree, that is, vertices correspond to operators and data flows from bottom to top; the root node produces the final query results. At this stage, there is no locality information about which VM each operator is executed on. After that, the execution plan is decomposed into smaller sub-queries, where each subquery corresponds to an execution stage. Those stages will be referred to as *strides*. Before a stride begins its execution, all the lower strides must have been completed. In the figure, we provide an example of a query plan decomposed in 4 strides.

Assume now that for each stride, each cloud provider is capable of providing a bid as shown in the bottom right of the figure. Each cloud provider can offer multiple combinations of type and number of VMs at a different cost, and each

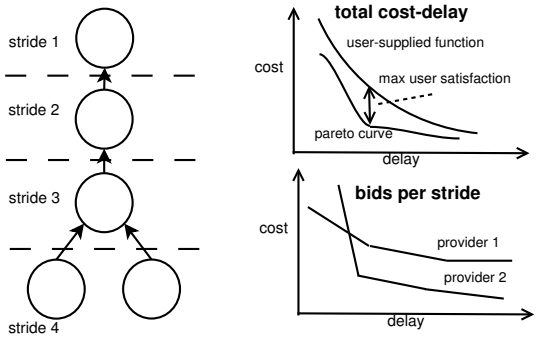

Figure 1: An analysis DAG (left) and example user and provider cost-delay functions (right) [22].

such combination may result in different expected execution time. In the generic case, the complete offer per provider per stride is described by a continuous function. In addition, we assume that each user specifies her own function that represents the worst acceptable trade-off. We further assume that the total monetary cost of the query plan is the sum of the cost of each stride; similarly, the total time delay is the sum of the delays in each stride. The aim of a multi-cloud optimizer is to derive an optimal assignment of strides to VMs. This is equivalent of determining exactly one bid point from each stride, so that the total monetary cost maximizes the difference from the user-supplied function. We are interested in this difference, which is termed as *user satisfaction*, because it captures the savings from the worst acceptable payment.

The problem above involves the computation of the pareto frontier and is NP-hard [12]. A simpler version of this problem has been investigated in the context of the Mariposa distributed query processing system [20, 12], where the bid of each provider for each stride is a single point rather than a continuous function. The proposal in [22] generalizes the initial solutions allowing arbitrary non-increasing cloud provider functions and guarantees optimal solutions with bounded relative error in pseudo-polynomial time.

The main problem with the above multi-cloud optimizers is that to date, there is no mechanism to provide the cloud provider bids, which serve as their input. This work aims to complement the proposals in [20, 12, 22] and provide such a mechanism. So, apart from the fact that a bi-objective cost model is significant in its own right, our proposal serves a secondary purpose, namely to assist in rendering the existing approaches to multi-cloud optimization applicable in practice.

3. OUR COST MODEL

The model we are presenting is used for estimating the time and the economic cost of a query plan executed on cloud-based VMs. To achieve this, we have built on top of the single-objective cost models described in [23] and [19]. However, our model can be applied to more generic data flows that are still expressed as DAGs. For simplicity, we start assuming that our process is a traditional query plan, and at the end of this section, we generalize.

3.1 Assumptions

Before we describe the cost model's rationale and functions, we need to state the assumptions we make:

- The shape of the query plan and the operator ordering have already been chosen by a centralized optimizer.
- There is a mechanism that decomposes the query plan into strides in place.
- Every operator can be executed only on one node. The number of VMs that will be used in every stride is equal to the number of the query plan vertices that the stride comprises. As such, there is no intra-operator parallelism, where an operator runs on several processors.

3.2 The Cost Model

Our cost model is modular and consists of components that model the charging policies, the computational and

the communication execution time, respectively. Based on those, the economic price is derived as explained below.

3.2.1 Modeling the charging policies and fees

In the first part, we model the charging policies described in Section 2 and we map them to specific VM offers. The notation is as follows:

- P_t^a: Denotes the charging policies of the providers, where:
  - a: it denotes the type of charging policy:
    - a=1: corresponds to *On demand Instance*;
    - a=2: corresponds to *Reserved Instance*;
    - a=3: corresponds to *Committed VM*;
    - a=4: corresponds to *Spot Instance*;
  - t: denotes the charging time unit of charging in minutes, $t \in 1, 5, 10, \ldots, 60$.
- VM_k: it denotes the specific VM instance, where $1 \leq k \leq |Available\ VMs|$.
- $F_{pr}(P_t^a, VM_k) = (f_a, f_u)$: it denotes the price offer from cloud provider pr for the VM instance VM_k according to the pricing policy P_t^a. It consists of two parts: f_a corresponds to the amortized price per time unit for the policies that involve an upfront fee payment (e.g., *Reserved Instance*), whereas it is normally 0 for the *On demand Instance* policy; f_u corresponds to the fee related to the actual VM usage, which is set to 0 for the *Committed VM* policy. In policies where the actual usage is not monitored, as may happen in the *Committed VM* one, t is set to 1, to allow for more detailed model estimates.

3.2.2 Estimation of Execution Time

To estimate the time that a query takes to be executed on a specific VM, we use the following formula:

$$TotalTime = \sum_{s=1}^{n} \max(S_{s,1\rightarrow i}^{VM_{k\rightarrow k'}}, ..., S_{s,m^s\rightarrow j}^{VM_{k\rightarrow k'}})$$

where:

- n = the number of strides.
- m^s = the number of operators in the s-th stride.
- $S_{s,i\rightarrow j}^{VM_{k\rightarrow k'}} = O_{s,i}^{VM_k} + T_{s,i\rightarrow j}^{VM_{k\rightarrow k'}}$, where:
  - $O_{s,i}^{VM_k}$ = time to execute i-th operator of s stride on VM_k.
  - $T_{s,i\rightarrow j}^{VM_{k\rightarrow k'}}$ = time to transfer the data produced by operator i of the s-th stride, which runs on VM_k, to node j, which runs on $VM_{k'}$. Typically, j belongs to the $(s+1)$-th stride, and the following conditions hold:
    - $1 \leq s \leq n$,
    - $1 \leq i \leq m^s$,
    - $1 \leq j \leq m^{s+1}$,
    - $1 \leq k \leq |Available\ VMs|$

The rationale behind the *TotalTime* formula is that: (i) all strides are executed sequentially, and (ii) all operators in the same stride are executed in parallel.

To calculate $O_{s,i}^{VM_k}$ we can employ the technique described in [23] although our approach is orthogonal to the way $O_{s,i}^{VM_k}$ is estimated. According to [23], the equation for calculating the cost of an operator given a specific VM_k is:

$$\begin{aligned} O_{s,i}^{VM_k} &= \mathbf{n^T c_{VM_k}} \\ &= n_s c_s^{VM_k} + n_r c_r^{VM_k} + n_t c_t^{VM_k} + n_i c_i^{VM_k} + n_o c_o^{VM_k} \end{aligned} \quad (1)$$

$\mathbf{c_{VM_k}}$ is a vector of statistical metadata for the instance VM_k. The values of $\mathbf{c}$ depend only on the underlying hardware and can be found through a simple calibration procedure. This also means that the calibration of $\mathbf{c}$ has to be done only once for every system that we want to test, since they are independent of specific queries.

$\mathbf{n}$ is a vector of statistics of the data processed by the operator in the form of cardinalities. For estimating the cardinalities $\mathbf{n}$, Wu et al. in [23] propose a sampling-based approach. These values depend only on the query plan and not on the hardware that will be used to execute the query. Since the query plan is only one and is known, it is possible to calculate the cardinalities with this method without incurring big overhead.

To calculate $T_{s,i\rightarrow j}^{VM_{k\rightarrow k'}}$ we use, as our basis, the model described in [19], along with the results of cardinality estimation used for estimate the execution time of the operator on the i-th node. With the cardinality estimation results, we can calculate the size of produced data X, that will be transferred over the network. We can employ different cost estimation formulas depending on the physical position of the nodes. The nodes i and j of the stride s and $s+1$ respectively, can belong either to the same cluster or to a different one. In the first case we have *intra-cluster* communication, while in the second we have *inter-cluster* communication. These cases are examined as follows.

- Intra-cluster communication: we have two subcases:
  - Same VM instance ($k = k'$). The j-th operator will be executed on the same node as the i-th operator executed. This means that the produced data are already in the same VM. So we have:
    $$T_{s,i\rightarrow j}^{VM_{k\rightarrow k}} = 0$$
  - Different VM instance ($k \neq k'$). The j-th operator will be executed on a different VM instance than the i-th operator, but in the same cluster u. This means that data (X) has to be transferred from node i to node j. In this case:
    $$T_{s,i\rightarrow j}^{VM_{k\rightarrow k'}} = C_{k\rightarrow k'}^{u}(X) = \alpha_{k,k'}X + \beta_k$$
    where $\alpha_{k,k'}$ is the communication cost to transfer one unit of data from node k to node k' and β_k is the communication startup cost.
- Inter-cluster communication. In this case, operator i will be executed on a node in cluster u, while operator j will be executed on a node in cluster v. So the transfer time is estimated by the following formula.
  $$T_{s,i\rightarrow j}^{VM_{k\rightarrow k'}} = C_{k\rightarrow g_u}^{u}(X) + C_{g_u,g_v}(X) + C_{g_v\rightarrow k'}^{v}(X)$$

where:

- $C_{g_u,g_v}(X)$ is the communication cost for X amount of data between cluster u and v through their gateways g_u and g_v respectively.
- $C^u_{i\to g_u}(X)$ is the transmission cost between node k and the gateway node (g_u) in cluster u.
- $C^v_{g_v\to j}(X)$ is the transmission cost between gateway node (g_v) and node k' in cluster v.

3.2.3 Monetary Cost Estimation

The third part of our model is the estimation of the cost of a query in monetary units. To estimate this, we need to combine the pricing offers of the different providers with the time estimation of our model. The total price depends on the execution time of each operator and the associated fee:

$$\sum_{s=1}^{n}\sum_{i=1}^{m^s} Price(S^{VM_{k\to k'}}_{s,i\to j}, F_{pr}(P^a_t, VM_k), F_{pr}(P^a_{t'}, VM_{k'})),$$

where *Price*, computes the fee for using a VM_k for S time based on the P^a_t charging policy and transferring data to $VM_{k'}$.

Assume that $F_{pr}(P^a_t, VM_k) = (f_a, f_u)$ and $F_{pr}(P^a_{t'}, VM_{k'}) = (f'_a, f'_u)$. Then *Price* is estimated as follows:

$$Price = (f_a + f_u)\lceil\frac{S^{VM_{k\to k'}}_{s,i\to j}}{t}\rceil + (f'_a + f'_u)\lceil\frac{T^{VM_{k\to k'}}_{s,i\to j}}{t'}\rceil$$

capturing the fact that data transfer results in concurrent usage of both the sender and the receiver VMs.

It is worth noting that there are several alternatives when defining f_a, depending on the expected usage of the model. For example, if the user is not interested in the pre-paid amount, then f_a can be set to zero for any charging policy. Also, if the cost estimation is used to decide whether to request further VMs in addition to those already reserved, then f_a for the *on-demand* instances may include the amortized cost of the reserved instances instead of being 0. The formulas presented above are generic enough to support such scenarios. Finally, it is straightforward to extend them to support charges based on the volume of the data transferred across the network.

3.3 An Example

To give a better view of our model, we will present a simple example. Suppose we have query q that is executed in two strides. It consists of two operators, one in each stride. For simplicity we will assume that:

- The time to transfer the intermediate data from *Stride 2* to *Stride 1* is 1 hour (60 mins) if the VMs of each stride are different, or 0 if it is the same VM.
- There is no charge for data transfer between the nodes that execute the query. Usually this only happens when the nodes are of the same provider.
- All the data that the query needs, including the initial data and the intermediate data, fit completely in the storage space provided by the specific instance. This implies that we do not have any extra cost for storage space.

We have two IaaS providers, A and B, with their provided VMs presented in Table 1. The charging policies are in Table 2. The pricing offers based on those charging policies can be seen in Table 3 along with their respective f_a and f_u. Finally, the estimated time to execute each operator on the provided VMs ($O^{VM_k}_{s,i}$) can be seen in Table 4.

VM	Providers	ECU	Memory	Storage
VM_1	A, B	3	3.75 GiB	1 x 4 SSD
VM_2	A	6.5	7.5 GiB	1 x 32 SSD
VM_3	B	6.5	15 GiB	1 x 32 SSD
VM_4	B	13	15 GiB	2 x 40 SSD
VM_5	A	14	7.5 GiB	2 x 40 SSD

Table 1: Example of VM instances taken from AWS

Pricing Policy	Charging Time Unit
P^1_{60}	Hour
P^1_5	5 Min
P^2_{60}	Hour
P^3_1	Month
P^3_1	3 Month

Table 2: Pricing policies

Pricing Policy	f_a	f_u	Description
On-demand			
$F_A(P^1_{60}, VM_1)$	0	5	5$/Hour
$F_A(P^1_{60}, VM_2)$	0	17	17$/Hour
$F_B(P^1_5, VM_3)$	0	1.70	1.70$/5Min
$F_B(P^1_5, VM_4)$	0	4.80	4.80$/5Min
$F_A(P^1_{60}, VM_5)$	0	60	60$/Hour
Reserved Instance			
$F_A(P^2_{60}, VM_2)$	0.685	15	15$/Hour+500$/Month
$F_A(P^2_5, VM_3)$	0.057	1.5	1.50$/5min+500$/Month
$F_A(P^2_5, VM_4)$	0.057	4.3	4.30$/5min+500$/Month
$F_A(P^2_{60}, VM_5)$	0.685	53	53$/Hour+500$/Month
Committed VM			
$F_A(P^3_1, VM_1)$	0.1667	0	7300$/Month
$F_B(P^3_1, VM_1)$	0.1826	0	8000$/Month

Table 3: Pricing offers in the example

	VM_1	VM_2	VM_3	VM_4	VM_5
$O^{VM_k}_{2,1}$	2000 sec	1200 sec	1200 sec	620 sec	600 sec
$O^{VM_k}_{1,1}$	1500 sec	620 sec	620 sec	240 sec	230 sec

Table 4: Example of estimated execution times

In Figure 2, we present a diagram of the estimated execution times along with the estimated monetary costs for the query q for every combination of VMs with *On demand* and *Reserved* charging policies in Table 3. Black bullets represent the *On-demand* model, while the white ones correspond to the *Reserved Instance* model. In general, the monetary cost is inversely proportional to the execution time and *Reserved Instance* pricing is less expensive than *On demand*. In Figure 2, the rightmost combination (A) is to allocate both operators to VM_1, which is the less expensive albeit the slowest VM, while combination C corresponds to the mapping of the two query strides to $VM_5 \to VM_4$, which

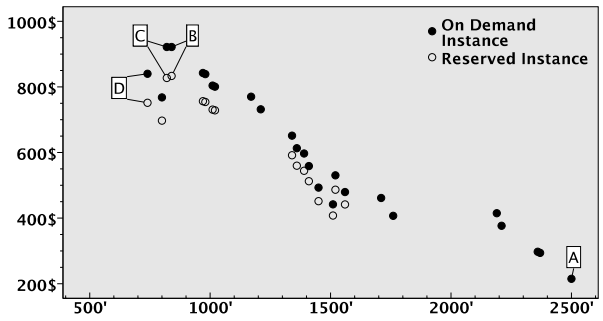

Figure 2: Time and monetary cost combinations for the example query

are the faster and most expensive VMs.

It can be seen that some VM combinations dominate some others, i.e., they are both more efficient in terms of execution time and less expensive. This is attributed to the different charging policies (e.g., charges for each hour or for each 5 min period of usage). One other factor is the fact that, in some combinations, both operators are executed on the same VM, and as such, there is no data transfer across the network, which leads to reduced execution time and monetary cost. For instance, for combination D on the diagram, VM_5 is used for both operators. Due to the absence of intermediate data transfer, this combination has both lower cost (by 60$) and running time (by 60 mins) under the *On demand* policy than combination B for example, where data has to be transferred from VM_4 to VM_5.

In this example, we do not have any extra cost for the transfer of intermediate data between strides. If we had such a cost, we could use the cardinality estimation to determine the size of data transferred. Since we know the data that to be transferred and the charging policies of the IaaS providers, we are able to determine the cost for transferring data between strides. Also, in the example, we have assumed that all the data, including initial and intermediate data, fits completely in the storage space. If this is not the case, we can estimate the data volume that needs to be stored based on the cardinality estimates, and then estimate the monetary cost for using extra storage space.

3.4 Generalizations

The cost model described in this section can be generalized in two main ways: to support arbitrary data analysis flows and intra-operator parallelism.

Data analysis flows are typically represented as directed acyclic graphs (DAGs), which can be naturally split in multiple stages, exactly as the query plans we consider do. The main difference between arbitrary data flows and query plans is that query execution plans consist of operators from the extended relational algebra, whereas data flows also encompass data and text analytics, machine learning operations, and so on [18]. The implication in our cost model is in the way $O_{s,i}^{VM_k}$ is estimated. The approach in [23] cannot apply because it is specific to atomic query operators; so we need to resort to micro-benchmarking solutions, e.g., as described in [8, 17]. The rest of the cost model details remain the same.

Regarding intra-operator parallelism, the extensions are more straightforward. We can assume that, if we fix the degree of intra-operator parallelism, then we modify the query execution plan, so that each instance of a partitioned operator appears as a separate query plan vertex. Then, we can apply the cost model without any modifications.

Finally, a shortcoming of the monetary cost estimation in Section 3.2.3 is that a VM that receives data from a remote host is activated and, if it does not start processing its results immediately, it may not be de-activated. The formula presented does not capture this, but it is straightforward to devise more sophisticated formulas that keep track of the first time a VM is activated until it finishes the execution of all the tasks allocated to it.

	$\alpha_{k,k'}$(ms/bytes)			β_k (ms)
	$k'=1$	$k'=2$	$k'=3$	
$k=1$	-	6.29e-05	5.54e-05	8.61e02
$k=2$	6.60e-05	-	6.39e-05	9.05e02
$k=3$	5.67e-05	6.09e-05	-	1.08e03

Table 5: Network α and β parameters

4. VALIDATION CASE STUDY

In this section, we demonstrate how exactly the cost model is used in a real multi-cloud environment. The first part shows how the cost model is calibrated, while the second part shows how we derive time estimates. Monetary cost estimates are covered by the example in Section 3.3.

4.1 Experimental Setting and Model Calibration

For our experiments we used *okeanos*. *okeanos* is a IaaS platform for the Greek Academic and Research Community [9]. More specifically, we used 3 VMs with the following hardware configurations:

- 2 VMs (VM_1 and VM_3) : 60GB Disk, 4GB Ram, 1-core x 2.1GHz
- 1 VM (VM_2): 40GB Disk, 2GB Ram, 1-core x 2.1GHz

Our software setup includes the installation of PostgreSQL 9.1.11 on Linux Kernel 3.2.0-58-generic. The data we used come from the TPC-H decision support benchmark and the database size is 26GB.

To use the cost model, we need to parameterize the time estimates for data transfer, writing (resp. reading) raw data to (resp. from) disk and for the relational operators. For the calculation of the network speed, our case is that the VMs belong to the the same cluster. So the network formula is given by:

$$T_{s,i\to j}^{VM_{k\to k'}} = C_{k\to k'}^{u}(X) = \alpha_{k,k'}X + \beta_k$$

where $\alpha_{k,k'}$ is the communication cost to transfer one unit of data from node k to node k' and β_k is the startup cost.

To find α and β, we conducted two experiments with different X for every combination of VMs. We used the command *dd* of unix to produce two files and the command *scp* to transfer these files between servers and measure the network performance. We repeated this experiment 10 times for each X value and then calculated the mean time. Since we have a linear function, we can calculate α and β with simple maths. The results are presented in the Table 5.

Next, to estimate the query time correctly, we need to know the read and write speed of the disk of every VM. We

	VM_1	VM_2	VM_3
Read Speed (MB/s)	291.60	255.30	43.88
Write Speed (MB/s)	100.83	106.20	96.39

Table 6: Disk Performance in MB/s

Optimizer Parameter (ms)	VM_1	VM_2
seq_page_cost	2.16e-02	3.21e-02
cpu_tuple_cost	3.76e-04	3.48e-04
cpu_operator_cost	2.17e-04	2.48e-04

Table 7: Cost units of our VMs

measure the sequential speed using the *dd* unix tool while we set the block size to 4096 bytes. Again, we repeated the measurements 10 times and we calculated the mean values. The results are presented in Table 6.

Finally, to estimate the execution time $O_{s,i}^{VM_k}$ of relational operators executed by postgreSQL, we need both cost units **c** and cardinalities **n**. For cardinalities **n**, we assume that they can be calculated with the method described in [23] and thus are accurately known. To calculate **c**, we have used the calibration queries from [23]. We have calculated only the cost units that are involved in the experiments in the next part; the results are in Table 7. The queries are:

- `SELECT * FROM R`
  R is memory resident. This query is used to find *cpu_tuple_cost*.
- `SELECT COUNT(*) FROM R`
  R is memory resident. This query along with the previous are used to find *cpu_operator_cost*.
- `SELECT * FROM R`
  R does not fit in memory. With the help of the first query, this query is used to find *seq_page_cost*.

where *seq_page_cost* gives the time cost to sequentially perform an I/O operation accessing a disk page. *cpu_tuple_cost* is the time to retrieve and process a tuple. *cpu_operator_cost* captures the extra cost of applying a hash or an aggregate function to a tuple (that cost is not covered by *cpu_tuple_cost*).

4.2 Running time estimates

In our experiments, we tried to validate whether our cost model can predict with adequate precision the execution time of a query. We used postgreSQL database only for the operators of the bottom strides. All the other operators are implemented with unix scripts. The cost units **c** were used only to predict the running time of the postgreSQL operators. To calculate the time of operators implemented with scripts, we treated them as black boxes, we executed them with different inputs and measured their performance.

Experiment 1

The query of the first experiment is:

```
SELECT c_nationkey, count(*)
FROM customer
GROUP BY c_nationkey
```

The corresponding plan is presented in Figure 3 on the left.

For the execution of the above query, we use two VMs (VM_1 and VM_2). VM_1 is used for stride 1, while VM_2 runs stride 2. The execution is split in three steps as follows:

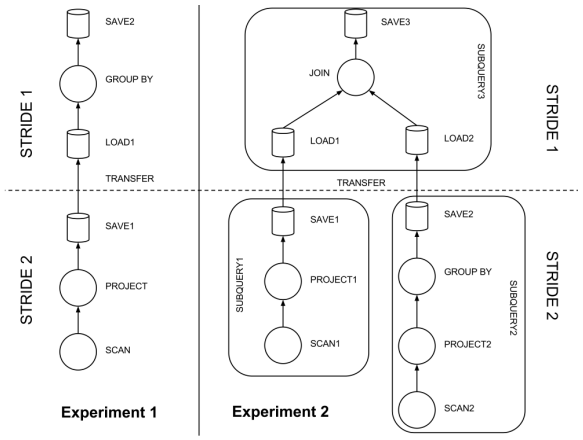

Figure 3: Plans of the experiments.

	Estimated Values (ms)	Actual Values (ms)
Exp. 1	15531	16172
Exp. 2	3131	3285

Table 8: Results of actual execution time and estimated time

1. SCAN, PROJECT and SAVE1 are executed on VM_1 with as single SQL sub-query submitted to the postgeSQL database.
2. TRANSFER is performed using the unix *scp* command.
3. LOAD1, GROUP BY and SAVE2 is implemented as a single unix script running on VM_2.

We sum the execution time of each of the above steps to calculate the total actual execution time of the query. Our experiments were repeated 10 times, and their mean value is in the *Actual Values* column of Table 8.

To find the estimated execution time, we used the actual cardinality. For the calibration of the execution time of the *GroupBy* operator, we first applied this operator on a set of 1 million random numbers for 10 different times. Again, the total time is given by the sum of the times of the three steps and the results are in Table 8 (see the *Estimated Times* column). From the table, we can observe that the deviation between the actual and the estimated times in that experiment is less than 4%.

Experiment 2

The query of the second experiment is:

```
SELECT nation.n_name, count(*)
FROM customer, nation
WHERE customer.c_nationkey = nation.n_nationkey
GROUP BY nation.n_nationkey
```

The detailed execution plan can be seen in Figure 3 on the right. We can observe that the database optimizer has performed an optimization, which pushes the group-by under the join yielding significantly lower execution times.

For the execution of this query, we use three VMs. VM_1 and VM_2, which have both postgreSQL database installed, are used for the bottom stride, while VM_3 executes the first stride. The execution consists of three sequential steps:

1. SubQuery1 and SubQuery2 are executed in parallel on VM_1 and VM_2, respectively.

2. The intermediate data is transferred with the help of *scp* as previously.

3. SubQuery3 is implemented as unix script (using *join* command) that runs on VM_3.

The total time is estimated by adding the maximum estimated time of subqueries 1 and 2 in Stride 2 along with the data transfer from the first stride to the second one. Then we add the estimated time of subquery3. The average results of the 10 runs are in Table 8. Again, the deviation of the estimates from the actual running times is low (4.69%). It is important to clarify that the VMs used were not installed on dedicated machines. On the contrary, the cloud infrastructure in our experiments is heavily used by a big community that shares the physical resources provided.

5. CONCLUSIONS

In this work, we presented a bi-objective cost model that provides time and monetary cost estimates of query plans running on VMs from multiple cloud providers. Our cost model extends existing approaches that focus on time estimates when tasks run on predefined machines and is tailored to a multi-cloud environment. The cost model is also applicable to generic data flow tasks, and through a detailed example and a validation case study, we show how it can be employed in practice. There are several avenues to extend our work, since providing time and cost estimates is a complex issue. Two of the most important directions are to devise cost models that map tasks to the amortized cost of using the infrastructure (rather than the price charged) and to perform more thorough validation.

Acknowledgements

This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

6. REFERENCES

[1] Amazon.com, Inc. Amazon EC2 Instances. `http://aws.amazon.com/ec2/instance-types/`.

[2] Amazon.com, Inc. Amazon EC2 Pricing. `http://aws.amazon.com/ec2/pricing/`.

[3] Amazon.com, Inc. Amazon Elastic Block Store.

[4] Cloudsigma AG. Features-True Flexibility. `http://www.cloudsigma.com/#features`.

[5] GoGrid, LLC. Go Grid Pricing. `http://www.gogrid.com/products/cloud-servers#pricing`.

[6] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: Research problems in data center networks. *ACM SIGCOMM Computer Communication Review*, 39(1):68–73, 2008.

[7] J. Hamilton. Cooperative expendable micro-slice servers (CEMS): Low cost, low power servers for internet-scale services. Technical report, 2009.

[8] B. Huang, S. Babu, and J. Yang. Cumulon: optimizing statistical data analysis in the cloud. In *Proc. SIGMOD Conference*, pages 1–12, 2013.

[9] V. Koukis, C. Venetsanopoulos, and N. Koziris. ∼okeanos: Building a cloud, cluster by cluster. *IEEE Internet Computing*, 17(3):67–71, 2013.

[10] X. Li, Y. Li, T. Liu, J. Qiu, and F. Wang. The method and tool of cost analysis for cloud computing. In *Proc. CLOUD Conference*, pages 93–100, 2009.

[11] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems, 3rd Edition*. Springer, 2011.

[12] C. H. Papadimitriou and M. Yannakakis. Multiobjective query optimization. In *Proc. PODS Conference*, pages 52–59, 2001.

[13] D. Perez-Palacin, R. Calinescu, and J. Merseguer. log2cloud: log-based prediction of cost-performance trade-offs for cloud deployments. In *Proc. SAC Conference*, pages 397–404, 2013.

[14] Rackspace US, Inc. Rackspace Cloud Block Storage. `http://www.rackspace.com/cloud/block-storage/`.

[15] Rackspace US, Inc. Rackspace Cloud Pricing. `http://www.rackspace.com/cloud/servers/pricing/`.

[16] A. Rahal, Q. Zhu, and P.-Å. Larson. Evolutionary techniques for updating query cost models in a dynamic multidatabase environment. *The VLDB Journal*, 13(2):162–176, 2004.

[17] P. Shivam, S. Babu, and J. S. Chase. Active and accelerated learning of cost models for optimizing scientific applications. In *Proc. VLDB Conference*, pages 535–546, 2006.

[18] A. Simitsis, K. Wilkinson, M. Castellanos, and U. Dayal. Optimizing analytic data flows for multiple execution engines. In *Proc. SIGMOD Conference*, pages 829–840, 2012.

[19] Y. Slimani, F. Najjar, and N. Mami. An adaptive cost model for distributed query optimization on the grid. In *Proc. OTM Conference*, pages 79–87, 2004.

[20] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A wide-area distributed database system. *The VLDB Journal*, 5(1):48–63, 1996.

[21] D. Taniar, C. H. C. Leung, J. W. Rahayu, and S. Goel. *High Performance Parallel Database Processing and Grid Databases*. John Wiley & Sons, 2008.

[22] E. Tsamoura, A. Gounaris, and K. Tsichlas. Multi-objective optimization of data flows in a multi-cloud environment. In *Proc. DanaC Workshop*, pages 6–10, 2013.

[23] W. Wu, Y. Chi, S. Zhu, J. Tatemura, H. Hacigumus, and J. F. Naughton. Predicting query execution time: Are optimizer cost models really unusable? In *Proc. ICDE Conference*, pages 1081–1092, 2013.

[24] V. Zadorozhny, L. Raschid, T. Zhan, and L. Bright. Validating an access cost model for wide area applications. In *Proc. CoopIS Conference*, pages 371–385, 2001.

[25] W. Zeng, C. Mu, M. Koutny, and P. Watson. A flow sensitive security model for cloud computing systems. Technical report, 2014.