An Integer Linear-programming based Resource Allocation Method for SQL-like Queries in the Cloud

Mohamed Mehdi Kandi, Shaoyi Yin, Abdelkader Hameurlain

An Integer Linear-programming based Resource Allocation Method for SQL-like Queries in the Cloud

Mohamed Mehdi Kandi
IRIT Laboratory, Paul Sabatier University
Toulouse, France
mkandi@irit.fr

Shaoyi Yin
IRIT Laboratory, Paul Sabatier University
Toulouse, France
yin@irit.fr

Abdelkader Hameurlain
IRIT Laboratory, Paul Sabatier University
Toulouse, France
hameurlain@irit.fr

ABSTRACT

Cloud computing has emerged as a paradigm for delivering Information Technology services over Internet. Services are provided according to a pricing model and meet requirements that are specified in Service Level Agreements (SLA). Recently, most of cloud providers include services for DataBase (DB) querying dedicated to run on MapReduce platform and a virtualized architecture. Classical resource allocation methods for query optimization need to be revised to handle the pricing models in cloud environnements. In this work, we propose a resource allocation method for the query optimization in the cloud based on Integer Linear-Programming (ILP). The proposed linear models can be implemented in any fast solver for ILP. The method is compared with some existing greedy algorithms. Experimental evaluation shows that the solution offers a good trade-off between the allocation quality and allocation cost.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**; • **Computing methodologies → MapReduce algorithms**;

KEYWORDS

Cloud Computing, PaaS, MapReduce, Query Optimization, Resource Allocation, Integer Linear-Programming

1 INTRODUCTION

Cloud computing became a common way to provide on-demand Information Technology services. Cloud services are offered by a provider who owns a hardware architecture and a set of software tools that meet client needs. In the cloud, resources can be reserved and released in an elastic way, which means that it is possible to change the allocated amount at any time. The services are provided according to a pricing model and meet a set of performance requirements that are specified in Service Level Agreements (SLA). If requirements are not met, the provider pays penalties to the client.

We are interested in cloud services for database querying (Platform-as-a-Service database, PaaS), particularly the problem of resource allocation. Most of the current cloud providers include services for database querying with languages similar to SQL in which queries run on MapReduce [2] clusters (Hive [7]). Among these services, we mention Amazon Elastic MapReduce[1], Microsoft Azure HDInsight[2], Oracle BigData Cloud service[3]. The proposed query languages are usually called SQL-like[4]. With the above services, a SQL-like query is transformed into a set of dependent MapReduce jobs. Each job contains a set of parallel tasks. These tasks are submitted to an allocator that places them within the available resources and defines the execution scheduling over time that respects the precedence constraints and resource availability.

Several solutions have been proposed for resource allocation in MapReduce paradigm [4][9][11]. The aim of most of this work is to ensure fairness (i.e. assign resources so that all jobs get an equal share of resources over time) and data locality (i.e. assign the task to the node that contains its data). These methods are better suited to classical parallel environments and do not handle cloud constraints. In classical parallel environments, resource allocation is efficient when it minimizes execution time and maximizes throughput. However, in the cloud, the aim is to maximize the monetary gain of the provider and meet the client requirements established in SLAs. Existing methods that take into account these aspects are generally based on greedy methods [3] that have the advantage of quick decision-making and simplicity of their design. However, greedy methods do not give a theoretical guarantee on the quality of the solution in terms of monetary gain, which generates a negative effect on the provider's gain.

Motivated by the limitations of the above methods, we propose a resource allocation method for the execution of SQL-like queries in the cloud. The solution consists of two phases : (1) place tasks in available resources and (2) choose the time windows allocated to each task. Each phase is modeled by an ILP model, so the resolution can be done with any exact ILP optimization algorithm. We compare in the experimental section our method with a one-phase ILP method, and some existing greedy methods [3]. We show that

https://doi.org/10.1145/3167132.3167148

[1] https://aws.amazon.com/fr/emr/
[2] https://azure.microsoft.com/fr-fr/services/hdinsight/
[3] https://cloud.oracle.com/bigdata
[4] https://docs.treasuredata.com/articles/hive

our method offers a good trade-off between the allocation cost and the monetary cost generated by the execution of queries.

The rest of this paper is organized as follows. In section 2, we present the considered database cloud services. Then we detail our resource allocation method in section 3. Section 4 reports the experimentation results, while section 5 reviews some related work on resource allocation for MapReduce applications. Finally, we conclude in section 6.

2 CLOUD DB SERVICE DESCRIPTION

2.1 Query Compilation and Execution

Figure 1 shows the considered architecture. SQL-like queries are submitted through a client interface. The lexical and syntactic analyzer checks if the query is correct and generates a graph of operators (joins, projections...). Logical optimization consists in reducing the volume of manipulated data by applying classical transformation rules of algebraic trees. Physical optimization determines the join algorithms and join order, then generate a graph of the execution plan. The nodes of this graph are MapReduce jobs and edges represent dependencies between them. A query is transformed into a job graph in different ways : (1) associate a job to each join operator [7], (2) associate one job to all operators [1] or (3) decompose the join operators into several groups and associate a MapReduce job to each group [10]. Intra-job parallelization consists in defining the number of Map and Reduce tasks of each job of the graph.

The provider's cloud infrastructure consists of a set of physical machines. A hypervisor, whose role is to manage the Virtual Machines (VMs), is installed on each physical machine. Each VM represents a MapReduce node. It contains a set of logical resources and a local resource allocation manager that receives allocation decisions from the global manager and returns the state of resources. Each logical resource can contain only one task at a given time. A logical resource is an abstract representation of a certain reserved CPU, memory and storage capacity. The global resource allocation manager receives the graphs of execution plans and performs task placement and scheduling given the available resources. Section 3 is devoted to a new resource allocation method that takes into account economic aspects.

2.2 Economic Model

We propose an economic model for a PaaS database provider. The profit is obviously defined by:

$$Profit = Income - Expenditure \tag{1}$$

We assume that $\mathbb{O}$ is the set of client classes, $\mathbb{C}(o)$ is the set of clients belonging to the class $o \in \mathbb{O}$ and $\mathbb{Q}(c)$ is the set of queries issued by the client c. We assume a query-based pricing model, i.e: the client pays an amount of money for each submitted query. The price of the query depends on its nature (number of operators, manipulated data sizes...) and the client class. The income of the PaaS provider is equal to the price of all submitted queries:

$$Income = \sum_{o \in \mathbb{O}} \sum_{c \in \mathbb{C}(o)} \sum_{q \in \mathbb{Q}(c)} QueryPrice(q) \tag{2}$$

Expenditures consist of resource costs and penalties.

$$Expenditure = Resources + Penalties \tag{3}$$

The resources are made available to the PaaS provider by an Infrastructure-as-a-Service (IaaS) provider in the form of VMs. VMs are rented by the PaaS depending on the duration of use. In Equation (4), $\mathbb{T}$ is the set of VM types, $\mathbb{V}(t)$ is the set of VMs of type t, $Price_t$ is the price of using a VM of type t for one time unit, D_v is the duration of use of the VM $v \in \mathbb{V}(t)$.

$$Resources = \sum_{t \in \mathbb{T}} \sum_{v \in \mathbb{V}(t)} Price_t * D_v + NetworkAccess \tag{4}$$

The price of penalties depends on the duration of the deadline violation ($ViolationDur$), the price of the query ($Query- Price$) and a percentage that depends on the class of the client ($PercentageSLA$):

$$Penalties = \sum_{o \in \mathbb{O}} \sum_{c \in \mathbb{C}(o)} \sum_{q \in \mathbb{Q}(c)} ViolationDur(q) * W(q) \tag{5}$$

$W(q) = QueryPrice(q) * PercentageSLA(o)$. The resource allocation models that we propose in the following are intended to minimize expenditure (resources+penalties).

3 RESOURCE ALLOCATION METHOD

We propose a method based on Integer Linear-Programming (ILP) for the problem of resource allocation. Given a set of logical resources, the aim of our allocation method is to find a placement and a scheduling over time that minimize monetary costs for the PaaS cloud provider. The proposed solution adopts a two-phase approach. First, the placement involves choosing a pool of resources on which each task group will be executed. Then, scheduling consists of choosing the time windows allocated to each task group. A resource pool is a set of Map (or Reduce) resources with the same characteristics and physically close to each other. A task group is a set of Map (or Reduce) tasks that belong to the same job. We assume that the cardinality of resource pools is equal to the cardinality of task groups. We present in the following the ILP placement model (section 3.1) then the ILP scheduling model (section 3.2).

3.1 Placement Model (1^{st} phase)

The placement consists of choosing the resource pool on which each task group is executed. We introduce the following variables:

- $x_{i,m,a} = 1$ if the Map task group m of the job i is placed on the Map resource pool a; = 0 if not.
- $y_{i,r,b} = 1$ if the Reduce task group r of the job i is placed on the Reduce resource pool b; = 0 if not.
- $z_{a,b}$ = the maximum amount of data transferred between the task groups placed in the pool a and the task groups placed in the pool b.

$\mathbb{J}$ is the set of jobs for all submitted queries, $\mathbb{M}_i$ is the set of Map task groups of the job i, $\mathbb{R}_j$ is the set of reduce task groups of the job j, $\mathbb{A}$ is the set of Map resource pools, $\mathbb{B}$ is the set of Reduce resource pools:

$$x_{i,m,a} \in \{0,1\},\ \forall i \in \mathbb{J}, m \in \mathbb{M}_i, a \in \mathbb{A} \tag{6}$$

$$y_{j,r,b} \in \{0,1\},\ \forall j \in \mathbb{J}, r \in \mathbb{R}_j, b \in \mathbb{B} \tag{7}$$

$$z_{a,b} \in \{0,1....,UpperBound(z)\},\ \forall a \in \mathbb{A}, b \in \mathbb{B} \tag{8}$$

Multiple tasks can be assigned to the same resource. The exclusivity of execution is then ensured in time with the scheduling model which will be presented in the section 3.2.

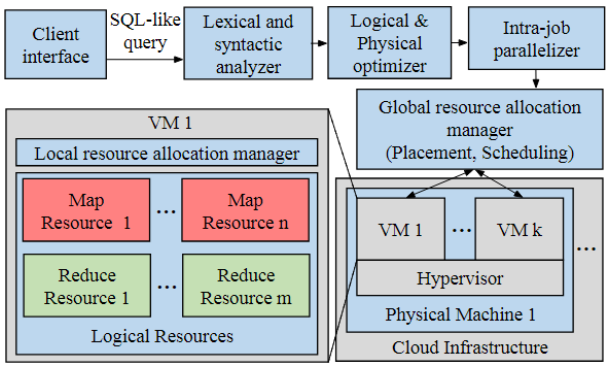


Figure 1: SQL-like query compilation and execution in the cloud

The processor capacity C_p^M (resp. memory C_m^M and storage C_s^M) needed by a Map task group can not exceed the available capacity C_p (resp. C_m and C_s) in the resources of the chosen pool - equations (9), (10), (11) -. Likewise, for Reduce tasks - equations (12), (13), (14) -:

$$C_p^M(i) * x_{i,m,a} \leq C_p(a),\ \forall i \in \mathbb{J}, m \in \mathbb{M}_i, a \in \mathbb{A} \quad (9)$$

$$C_m^M(i) * x_{i,m,a} \leq C_m(a),\ \forall i \in \mathbb{J}, m \in \mathbb{M}_i, a \in \mathbb{A} \quad (10)$$

$$C_s^M(i) * x_{i,m,a} \leq C_s(a),\ \forall i \in \mathbb{J}, m \in \mathbb{M}_i, a \in \mathbb{A} \quad (11)$$

$$C_p^R(i) * y_{i,r,b} \leq C_p(b),\ \forall i \in \mathbb{J}, r \in \mathbb{R}_i, b \in \mathbb{B} \quad (12)$$

$$C_m^R(i) * y_{i,r,b} \leq C_m(b),\ \forall i \in \mathbb{J}, r \in \mathbb{R}_i, b \in \mathbb{B} \quad (13)$$

$$C_s^R(i) * y_{i,r,b} \leq C_s(b),\ \forall i \in \mathbb{J}, r \in \mathbb{R}_i, b \in \mathbb{B} \quad (14)$$

Each task group is placed on one and only one resource pool:

$$\sum_{a \in \mathbb{A}} x_{i,m,a} = 1,\ \forall i \in \mathbb{J}, m \in \mathbb{M}_i \quad (15)$$

$$\sum_{b \in \mathbb{B}} y_{i,r,b} = 1,\ \forall i \in \mathbb{J}, r \in \mathbb{R}_i \quad (16)$$

In order to ensure inter-job parallelism, two Map (or Reduce) task groups belonging to the same job can not be placed on the same resource pool:

$$\sum_{m \in \mathbb{M}_i} x_{i,m,a} \leq 1,\ \forall i \in \mathbb{J}, a \in \mathbb{A} \quad (17)$$

$$\sum_{r \in \mathbb{R}_i} y_{i,r,b} \leq 1,\ \forall i \in \mathbb{J}, b \in \mathbb{B} \quad (18)$$

Our placement model handle load balancing. In order to ensure an equitable distribution of tasks between resources, we propose to minimize the maximum number of time windows allocated in each resource, which avoids to allocate too many task groups to one resource pool and few to another. To model this in a linear way, we introduce two variables $\alpha \in \{0, 1...., T\}$ and $\beta \in \{0, 1...., T\}$ (T is the number of considered time windows). We add the following two constraints. The objective function that is subsequently presented includes α and β as variables to minimize.

$$\sum_{i \in \mathbb{J}} \sum_{m \in \mathbb{M}_i} Tm_i * x_{i,m,a} + \sum_{t<T} (1 - Fm_{a,t}) \leq \alpha,\ \forall a \in \mathbb{A} \quad (19)$$

$$\sum_{i \in \mathbb{J}} \sum_{r \in \mathbb{R}_i} Tr_i * x_{i,r,b} + \sum_{t<T} (1 - Fr_{b,t}) \leq \beta,\ \forall b \in \mathbb{B} \quad (20)$$

With $Fm_{a,t} = 1$ if the Map resource pool a is initially available at the moment t, $= 0$ if not, $Fr_{b,t} = 1$ if the Reduce resource pool b is initially available at the moment t, $= 0$ if not, Tm_i is the local response time of Map tasks of job i, Tr_i is the local response time of a Reduce task of job i. The family of variables z satisfies the following condition:

$$x_{i,m,a} = 1 \text{ and } y_{j,r,b} = 1 \Rightarrow z_{a,b} \geq Q_{i,j}$$
$$\forall i, j \in \mathbb{J}, m \in \mathbb{M}_i, r \in \mathbb{R}_j, a \in \mathbb{A}, b \in \mathbb{B}, Q_{i,j} > 0 \quad (21)$$

$Q_{i,j}$ is the amount of data transferred between a task of job i and a task of job j. This condition can be expressed linearly as follows:

$$Q_{i,j} * x_{i,m,a} + Q_{i,j} * y_{j,r,b} - z_{a,b} \leq Q_{i,j},$$
$$\forall i, j \in \mathbb{J}, m \in \mathbb{M}_i, r \in \mathbb{R}_j, a \in \mathbb{A}, b \in \mathbb{B}, Q_{i,j} > 0 \quad (22)$$

The economic parameter influenced by task placement is the cost of using resources and communication. The following objective function includes processor, memory, storage and network costs. Reducing the use of these resources allows to release the under-used VMs and thus decrease expenditure. W_{proc} (weight of processor usage), W_{mem} (weight of memory usage), W_{stor} (weight of storage space usage) are deduced from economic model according to the influence of the processor, memory and storage capacity on the price of VMs. The objective function of the placement model is:

$$f = \sum_{i \in \mathbb{J}} \sum_{m \in \mathbb{M}_i} \sum_{a \in \mathbb{A}} C_{map}(a) * Tm_i * x_{i,m,a} + \sum_{i \in \mathbb{J}} \sum_{r \in \mathbb{R}_i} \sum_{b \in \mathbb{B}} C_{reduce}(b) * Tr_i * y_{i,r,b} + \sum_{a \in \mathbb{A}} \sum_{b \in \mathbb{B}} C_{com}(a,b) * z_{a,b} + W_{rep} * (\alpha + \beta) \quad (23)$$

W_{rep} is the weight of the load balancing, $C_{com}(a,b) = W_{com} * Dist(a,b)$, $C_{map}(a) = W_{proc} * C_p(a) + W_{mem} * C_m(a) + W_{stor} * C_s(a)$ and $C_{reduce}(b) = W_{proc} * C_p(b) + W_{mem} * C_m(b) + W_{stor} * C_s(a)$, $Dist(a,b)$ is the distance between the resource pool a and b, W_{com} is the weight of communication. The formulation of the problem to solve is:

minimize f

subject to (6), (7), (8), (9), (10), (11), (12), (13), (14), (15), (16), (17), (18), (19), (20), (22)

3.2 Scheduling Model (2^{nd} phase)

The optimal configuration of the previous placement model is considered as an input for the scheduling model that we present in this section. We now look for the time windows allocated to each task group. The following variables are introduced:

- $v_{i,m,t} = 1$ if the Map task group m of the job i started at or before time t; $= 0$ if not.

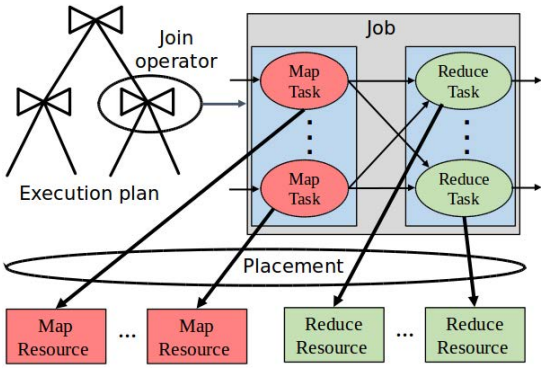


Figure 2: Task placement under MapReduce

- $w_{i,r,t} = 1$ if the Reduce task group r of the job i started at or before time t; $= 0$ if not.

So we have:

$$v_{i,m,t} \in \{0,1\},\ \forall i \in \mathbb{J}, m \in \mathbb{M}_i, t < T \tag{24}$$

$$w_{i,r,t} \in \{0,1\},\ \forall i \in \mathbb{J}, r \in \mathbb{R}_i, t < T \tag{25}$$

$$v_{i,m,t} \leq v_{i,m,t+1},\ \forall i \in \mathbb{J}, m \in \mathbb{M}_i, t < T \tag{26}$$

$$w_{i,r,t} \leq w_{i,r,t+1},\ \forall i \in \mathbb{J}, r \in \mathbb{R}_i, t < T \tag{27}$$

The economic parameter influencing the task scheduling over time is penalty cost. The goal is therefore to find the combination of variable families v an w that minimize the cost of penalties. A cumulative penalty weight W_i is associated to each execution time window after the deadline. For example, the provider may propose two offers : (1) minimum and (2) premium. It is clear that the violation of the SLAs of the offer (2) is less tolerated than offer (1). So if a job i is in offer (1) and the job j is in offer (2) then $W_j > W_i$. Each query has a deadline, which is specified in the SLA. The accumulation of penalties starts from the moment when the execution of the query jobs exceeds this deadline. The objective function to minimize is the following (D_i is the deadline of job i, $\mathbb{F}$ is the set of final jobs of the submitted queries):

$$g = \sum_{i \in \mathbb{F}} \sum_{D_i - Tr_i < t < T} W_i * \max_{r \in \mathbb{R}_i} (1 - w_{i,r,t}) \tag{28}$$

This objective function is not linear. To have a linear form, we introduce a family of variables γ such that:

$$\gamma_{i,t} \in \{0,1\},\ \forall i \in \mathbb{J}, t < T \tag{29}$$

$$1 - w_{i,r,t} \leq \gamma_{i,t}\ \forall i \in \mathbb{J}, r \in \mathbb{R}_i, t < T \tag{30}$$

The objective function can be formulated linearly as follows:

$$g' = \sum_{i \in \mathbb{F}} \sum_{D_i - Tr_i < t < T} W_i * \gamma_{i,t} \tag{31}$$

For a given job, Reduce tasks can not start before the end of the Map tasks. This constraint can be expressed linearly:

$$v_{i,m,t-Tm_i} \geq w_{i,r,t},\ \forall i \in \mathbb{J}, m \in \mathbb{M}_i, r \in \mathbb{R}_i, t - Tm_i \geq 1 \tag{32}$$

$$w_{i,r,t} \leq 0,\ \forall i \in \mathbb{J}, m \in \mathbb{M}_i, t - Tm_i < 1 \tag{33}$$

A resource can not contain more than one task at a same time (exclusivity constraint). From the definition of the families of variables u and y and knowing that a task can not be interrupted before its termination, we can deduce that:

- $v_{i,m,t} - v_{i,m,t-Tm_i} = 1$ if the Map task group m of the job i uses the pool of resources a at time t ; $= 0$ if not.
- $w_{i,r,t} - w_{i,r,t-Tr_i} = 1$ if the Reduce task group r of the job i uses the pool of resources b at time t ; $= 0$ if not.

Am_{im} (or Ar_{ir}) indicates the allocated resource pool on which the Map task group m (resp. Reduce task group r) of the job i was placed following the placement phase. The linear formulation of the exclusivity constraint is thus the following:

$$\sum_{i \in \mathbb{J}} \sum_{\substack{m \in \mathbb{M}_i \\ Am_{im}=a}} \sum_{t - Tm_i \geq 1} (v_{i,m,t} - v_{i,m,t-Tm_i}) + \sum_{i \in \mathbb{J}} \sum_{\substack{m \in \mathbb{M}_i \\ Am_{im}=a}} \sum_{t - Tm_i < 1} v_{i,m,t} \leq Fm_{a,t},\ \forall a \in \mathbb{A}, t < T \tag{34}$$

$$\sum_{i \in \mathbb{J}} \sum_{\substack{r \in \mathbb{R}_i \\ Ar_{ir}=b}} \sum_{t - Tr_i \geq 1} (w_{i,r,t} - w_{i,r,t-Tr_i}) + \sum_{i \in \mathbb{J}} \sum_{\substack{r \in \mathbb{R}_i \\ Ar_{ir}=b}} \sum_{t - Tr_i < 1} w_{i,r,t} \leq Fr_{b,t},\ \forall b \in \mathbb{B}, t < T \tag{35}$$

The parameter family O indicates the precedence between jobs. If $O_{i,j} = 1$ then the job j can not start before the end of the job i. We propose the following linear formulation for the precedence constraint:

$$v_{j,m,t} - w_{i,r,t-Tr_i} \leq 1 - O_{i,j},\ \forall i,j \in \mathbb{J}, m \in \mathbb{M}_j, r \in \mathbb{R}_i, t - Tr_i \geq 1 \tag{36}$$

$$v_{j,m,t} \leq 1 - O_{i,j},\ \forall i,j \in \mathbb{J}, m \in \mathbb{M}_j, t - Tr_i < 1 \tag{37}$$

The problem to solve is:

minimize g'

subject to (24), (25), (26), (27), (29), (30), (32), (33), (34), (35), (36), (37)

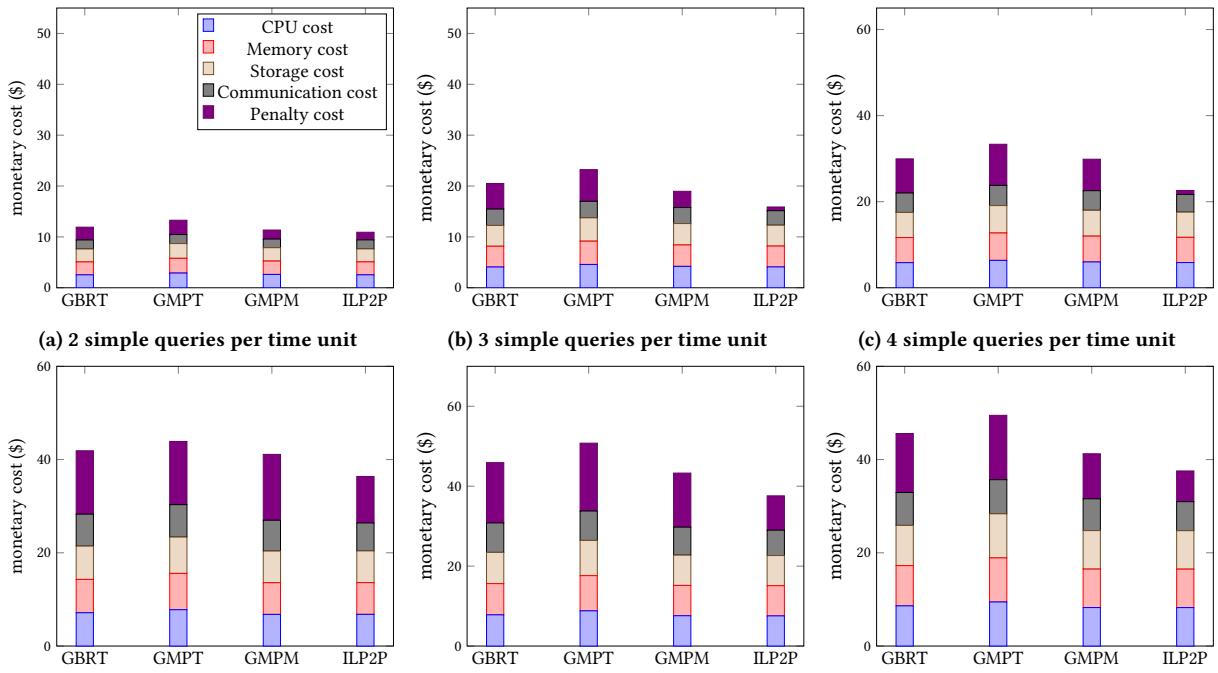
The global resource allocation manager periodically reports on the cost of penalties to a capacity management module. By analyzing the reports, the capacity management module may decide to reserve new VMs if the penalties are high over a long time interval.

4 EXPERIMENTAL RESULTS

We have performed extensive simulations to evaluate our allocation method. ILP models were implemented with GNU Linear Programming Kit[5] (GLPK). The job graphs we used for the tests were defined by observing the execution of some queries with Hive [7]. Observed queries were retrieved from the hive-testbench tool[6].

The method proposed in this paper (**ILP2P**) is first compared with three heuristics presented in [3]. The heuristics follow the same generic greedy algorithm but differ in the criterion to choose the tasks to place as well as the target resources. In each iteration :

[5]https://www.gnu.org/software/glpk/
[6]https://github.com/hortonworks/ hive-testbench



(a) 2 simple queries per time unit **(b) 3 simple queries per time unit** **(c) 4 simple queries per time unit**

(d) 2 complex queries per time unit **(e) 3 complex queries per time unit** **(f) 4 complex queries per time unit**

Figure 3: Monetary cost comparison (GBRT,GMPT,GMPM,ILP2P)

(1) **G-BRT** assigns the task with maximum execution time to the resource that minimizes the standard deviation of the utilization of resources, (2) **G-MPT** assigns the task with the maximum execution time to the resource that minimizes completion time, (3) **G-MPM** assigns the task with the maximum output size to the resource that minimizes monetary cost.

In the simulation, we consider two types of VMs. A **type1** VM contains 32 CPU and 8GB of RAM, its price per hour of use is 1.5$. A **type2** VM contains 16 CPU and 4GB of RAM, its price per hour of use is 0.75$. We consider the arrival of simple queries (< 6 jobs per query) in sub-figures (3a),(3b),(3c) and complex queries (≥ 6 jobs per query) in sub-figures (3d),(3e),(3f). Each job contains 16 to 40 Map (resp. Reduce) tasks. The size of each Map block is 256 or 512 MB. The initial resource availability rate is generated randomly. Each sub-figure represents the average monetary cost per time unit for different arrival rates.

Figures (3a),(3b),(3c),(3d),(3e),(3f) show that G-MPM and ILP2P have a lower cost than G-BRT and G-MPT. The latter two methods handle load balancing and execution time reduction but it is not sufficient to reduce monetary costs. Indeed, when we have a set of queries to place and schedule, and we want to reduce costs, we should schedule first the query with the most restrictive deadline and penalty weight and not the query that minimizes the global execution time. G-MPM handles monetary cost but uses a greedy method in which a part of the solution is determined at each step of the algorithm. This part is determined with the available information in the current step and without taking into account all possible placement and scheduling configurations. This may give rise to choices that would seem interesting given the information available in the step where the choice was made, but it will turn out that it is not a good choice later. Its results are thus worse than that of ILP2P which adopts an exact approach.

In a second step, we compare our two-phase ILP method (**ILP2P**) with another ILP method (**ILP1P**) designed to show the advantages of adopting a two-phase approach. ILP1P is based on a single phase approach, i.e. one ILP model that handles both placement and scheduling at the same time.

Table 1: Allocation cost (seconds)

	average	minimum	maximum
G-BRT	0.020	0.017	0.052
G-MPT	0.223	0.178	0.401
G-MPM	0.228	0.176	0.483
ILP2P	2.272	0.931	19.405
ILP1P	376.043	54.763	1201.742

The results of Figure 4 shows that ILP1P has obviously the best monetary cost. Indeed, if the problems of placement and scheduling are treated at the same time then the search space is significantly larger. It is therefore likely to find a better solution in terms of monetary cost. On the other hand, dealing with scheduling and placement problems at the same time gives rise to a more complex problem. Table 1 illustrates the average, minimal and maximum allocation times of the different methods. Given the complicated nature of ILP1P, its allocation time is very long and unreasonable in practice. Although ILP2P is slower than the greedy methods, its

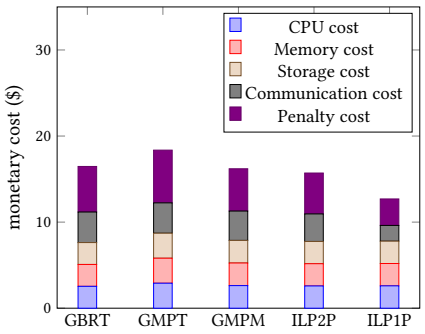


Figure 4: Monetary cost comparison (GBRT, GMPT, GMPM, ILP2P, ILP1P) - 2 simple queries per time unit

allocation time remains reasonable and significantly better than ILP1P. Indeed, the allocation time is negligible compared to the execution time of the query. As shown in Figure 4, ILP2P makes it possible to gain about 1$ per time unit compared to G-MPM. ILP2P presents thus a good trade-off between the allocation cost and the monetary cost.

5 RELATED WORK

Several methods have been proposed in the literature to improve the resource allocation for MapReduce applications. Some solutions are more suitable for classical parallel environments [5][8][9][11], while others are dedicated to the cloud [3][6]. The goal of resource allocation for classical parallel environments is to minimize execution time and maximize throughput. The allocation in the cloud, on the other hand, supposes the existence of a provider and several clients with different needs, the goal is to meet client requirements (specified in SLAs) while maximizing profit.

Most of the existing work for parallel environments is limited to independent tasks. The basic allocation algorithm for these environments is FIFO. The allocator assigns the oldest waiting task to the first available resource. This solution is unfair. Indeed, when long tasks are submitted the later short tasks must wait until the earlier finish. FAIR [11] is a resource allocation algorithm that solves this problem by considering fairness. This algorithm ensures that each user queries receive a minimum resource capacity as long as there is sufficient demand. When a user does not need its minimum capacity, other users are allowed to take it. Despite its advantages, FAIR does not offer mechanisms to handle deadlines. ARIA [8] is a framework that manages this problem. For this purpose, ARIA builds a job profile that reflects performance characteristics of the job for Map and Reduce phases, then it defines a performance model that estimates the amount of map and reduce tasks for the job and its deadline. Finally, ARIA determines the job order for meeting deadlines based on the earliest deadline first policy.

The above resource allocation algorithms do not consider cloud features. Tan et al [6] position their work in the context of multi-tenant parallel databases. Their solution handles SLAs. Nevertheless, they consider only performance metrics in the allocation decision but not economic aspects. Among the existing resource allocation work dedicated to the cloud, Kllapi et al [3] is the closest to our context. This work considers economic aspects. Authors explore three different problems: (1) minimize execution time given a fixed budget, (2) minimize the monetary cost given deadlines and (3) find the right trade-off between execution time and monetary cost. They propose some greedy methods and a local search algorithm to allocate resources to dependent tasks. They show that the local search does not significantly improve the results compared to the greedy methods. However, it is known that the greedy approaches do not theoretically guarantee the quality of the solution. This can generate a negative impact on the provider benefit. Unlike greedy methods, we propose in our work an ILP formulation for the problem, so an exact solution can be found. Our work is compared with greedy methods in the experimental section.

6 CONCLUSION

We addressed in this work the resource allocation problem for SQL-like queries in the cloud. We proposed an ILP based method that ensures placement and scheduling. We implemented the models and compared our work with some existing methods. The results showed that our method provides a higher monetary gain compared to the greedy algorithms with a reasonable allocation time. As a future work, we plan to consider parameter estimation errors and design efficient dynamic strategies that detect estimation errors during execution time, then change the allocation plan to reduce the impact of these errors on the monetary cost.

REFERENCES

[1] Foto N Afrati and Jeffrey D Ullman. 2010. Optimizing joins in a map-reduce environment. In *Proceedings of the 13th International Conference on Extending Database Technology*. ACM, 99–110.

[2] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[3] Herald Kllapi, Eva Sitaridi, Manolis M Tsangaris, and Yannis Ioannidis. 2011. Schedule optimization for data processing flows on the cloud. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 289–300.

[4] Minghong Lin, Li Zhang, Adam Wierman, and Jian Tan. 2013. Joint optimization of overlapping phases in MapReduce. *Performance Evaluation* 70, 10 (2013), 720–735.

[5] Zhihong Liu, Qi Zhang, Mohamed Faten Zhani, Raouf Boutaba, Yaping Liu, and Zhenghu Gong. 2015. Dreams: Dynamic resource allocation for mapreduce with data skew. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 18–26.

[6] Zilong Tan and Shivnath Babu. 2016. Tempo: robust and self-tuning resource management in multi-tenant parallel databases. *Proceedings of the VLDB Endowment* 9, 10 (2016), 720–731.

[7] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu, and Raghotham Murthy. 2010. Hive-a petabyte scale data warehouse using hadoop. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*. IEEE, 996–1005.

[8] Abhishek Verma, Ludmila Cherkasova, and Roy H Campbell. 2011. ARIA: automatic resource inference and allocation for mapreduce environments. In *Proceedings of the 8th ACM international conference on Autonomic computing*. ACM, 235–244.

[9] Weina Wang, Kai Zhu, Lei Ying, Jian Tan, and Li Zhang. 2016. Maptask scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality. *IEEE/ACM Transactions on Networking* 24, 1 (2016), 190–203.

[10] Sai Wu, Feng Li, Sharad Mehrotra, and Beng Chin Ooi. 2011. Query optimization for massively parallel data processing. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 12.

[11] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. 2010. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems*. ACM, 265–278.