Cost Models for View Materialization in the Cloud

Thi-Van-Anh Nguyen, Laurent d'Orazio, Sandro Bimonte, Jérôme Darmont

Cost Models for View Materialization in the Cloud

Thi-Van-Anh Nguyen
CNRS, UMR 6072, GREYC
Université de Caen Basse Normandie, France
thi-van-anh.nguyen@unicaen.fr

Laurent d'Orazio
CNRS, UMR 6158, LIMOS
Université Blaise Pascal
Clermont-Ferrand, France
laurent.dorazio@univ-bpclermont.fr

Sandro Bimonte
Irstea, UR TSCF
Clermont-Ferrand, France
sandro.bimonte@irstea.fr

Jérôme Darmont
ERIC Lyon 2, Université de Lyon, France
jerome.darmont@univ-lyon2.fr

ABSTRACT

In classical databases, query performance is casually achieved through physical data structures such as caches, indexes and materialized views. In this context, many cost models help select a "best set" of such data structures. However, this selection task becomes more complex in the cloud. The criterion to optimize is indeed at least two-dimensional, with the monetary cost of using the cloud balancing query response time. Thus, we define in this paper new cost models that fit into the pay-as-you-go paradigm of cloud computing. These cost models help achieve a multi-criteria optimization of the view materialization vs. CPU power consumption problem, under budget constraints. Finally, we present experimental results that provide a first validation of our contribution and show that cloud view materialization is always desirable.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems-Distributed databases, Query processing

General Terms

Algorithms, Design, Economics, Performance

Keywords

Cloud computing, Cost models, Materialized views, Performance and cost optimization

1. INTRODUCTION

Cloud computing has recently emerged as a new challenging paradigm. Cloud computing may be defined as a pay-per-use model for enabling on-demand access to reliable and configurable resources that can be quickly provisioned and released with minimal management. Users only pay for the resources they use. They need not set up the infrastructure, nor buy the software. These features make the design of cloud data management systems desirable. Unfortunately, performance in cloud computing usually relies on scalability, using bigger instances (scale-up) or more nodes (scale-out), making the bill more expensive.

On the other hand, performance optimization in databases has been intensively studied for decades. Reusing methods such as indexing, buffering or view materialization in the cloud could help improving performances without increasing costs. For instance, materialized views, which we particularly focus on in this paper, physically store the results of some relevant, frequent queries. One of the main challenges is then to select what views to materialize. Traditionally, the criteria to consider in the selection process mainly include view storage and maintenance costs [6, 8].

In cloud computing, storage is virtually infinite, enabling to store all possible materialized views. However, each materialized view implies an extra storage cost. Thus, the performance optimization problem becomes finding the best trade-off between raw scalability (i.e., increasing resources) and materialized views under budget constraints. To illustrate the complexity of selecting materialized views in the cloud, let us introduce a simple, fictitious example. Let the storage cost be $0.10 per GB and per month, and the computing cost be $0.24 per hour. A 500 GB dataset is stored in the cloud for a month. Let Q, the monthly query workload, process in 50 hours. Then, storage cost is $50, computing cost is $12, for a total of $62. If some materialized views are used, let us assume that workload processing time becomes 40 hours. Thus, computing cost becomes $9.6. However, materialized views use up additional storage space, e.g., 50 GB. Thus, storage cost becomes $55, for a total cost of $64.6. Overall, performance has improved by 20%, but cost has also increased by 4%.

The selection of views to materialize depends on several parameters, such as the workload (query execution time, database size, etc.), the cloud pricing model (CPU, storage,

Year	Month	Day	Country	Region	Department	Profit
2000	12	31	France	Auvergne	Puy-de-Dôme	\$35.000
2000	1	1	France	Auvergne	Puy-de-Dôme	\$40.000
2000	12	31	Italy	Campanie	Naples	\$23.000
1999	1	1	Italy	Campanie	Naples	\$50.000

Table 1: Sales dataset excerpt

network, etc.). This choice is a trade-off between response time and cost, and depends on the needs of a particular user. At one end of the spectrum, users under a strong budget constraint may accept long response times, whereas at the other end, users may disregard cost if they need fast response.

In this paper, we address the multi-criteria optimization problem of selecting a set of materialized views while optimizing the global monetary cost of storing and querying a database in the cloud. To achieve this goal, our main contribution is the design of cost models for storing, maintaining and querying data with materialized views over a cloud architecture.

The remainder of this paper is organized as follows. In Section 2, we provide the background information that is used throughout the paper. In Sections 3 and 4, we define cost models for cloud data management and materializing views, respectively. In Section 5, we describe the optimization process that is based on these cost models. In Section 6, we present an experimental evaluation and the first performance analyses of our models. In Section 7, we discuss the state of the art and compare it to our approach. Finally, in Section 8, we conclude this paper and hint at future research directions.

2. BACKGROUND

We present in this section the background information related to view materialization in the cloud. We first introduce a simple fictitious use case that serves as a running example throughout this paper. Then, we describe a typical pricing model in the cloud, illustrated by some of Amazon Web Services (AWS). Then, we briefly recall the principle of view materialization.

2.1 Running example

To illustrate our work, we rely on a simulated dataset storing the sales of an international supply chain. Business users need to analyze the total profit per day, month, and year; and per administrative department, region, and country. Table 1 provides an excerpt of this dataset.

Our full dataset stores 10 years (2000-2010) of sale data. Its size is 500 GB. We run over this dataset a query workload Q that includes such queries as Q_1 = “sales per year and country”, whose processing time is 0.2 hour. The size of the Q's result is 10 GB. A typical materialized view we may consider to optimize overall response time is V_1 = “sales per month and country”, whose processing time is 0.1 hour. The whole set of selected materialized views is denoted V. V's size is 50 GB. Finally, the times to process Q with and without exploiting V are 40 hours and 50 hours, respectively.

2.2 Cloud pricing policies

Cloud Service Providers (CSPs) supply a pool of resources, such as hardware (CPU, storage, networks), development platforms and services. There are many CSPs on the market, such as Amazon, Google and Microsoft. Each CSP offers different services and pricing. This paper relies on a limited, yet representative enough model that includes the main, commonly billed elements, i.e., CPU, storage and bandwidth consumption. In order for the reader to have an overview of such a pricing policy, the following examples present a simplified version of AWS's offer.

In AWS, Elastic Compute Cloud (EC2) [1] provides computing resources. Different instance configurations can be rent (micro, small, large, extra large, etc.) at various prices, as illustrated in Table 2. For example, the cost for a small instance (1.7 GB RAM, 1 EC2 Compute Unit, 160 GB of local storage under Windows) is \$0.12 per hour. Let us apply this pricing model onto our use case running on two small instances. Without materialized views, processing time is 50 hours, and thus computing cost is $2 \times 0.12 \times 50 = \$12$. With materialized views, processing time falls down to 40 hours and computing cost down to $2 \times 0.12 \times 40 = \$9.6$.

Instance configuration	Price per hour
Micro	\$0.03
Small	\$0.12
Large	\$0.48
Extra large	\$0.96
...	...

Table 2: EC2 computing prices

Bandwidth consumption is billed with respect to data volume (Table 3). In this model, input data transfers are free, whereas output data transfer cost varies with respect to data volume, with an earned rate when volume increases. When applying this pricing model onto our use case, the cost of bandwidth consumption (query result of 10 GB) is $(10 - 1) \times 0.12 = \$1.08$.

Data volume	Price per month
Input data	
Any input data	Free
Output data	
First 1 GB	Free
Up to 10 TB	\$0.12 per GB
Next 40 TB	\$0.09 per GB
Next 100 TB	\$0.07 per GB
...	...

Table 3: Amazon bandwidth prices

Finally, the Amazon Simple Storage Service (S3) [2] supplies storage capabilities. In this model, the price varies with respect to data volume, with an earned rate when volume increases (Table 4). In our running example, monthly storage price when not using materialized views (500 GB dataset) is $0.14 \times 500 = \$70$, and $0.14 \times (500 + 50) = \$77$ when using materialized views (additional 50 GB storage).

Data volume	Price per month
First 1 TB	\$0.14 per GB
Next 49 TB	\$0.125 per GB
Next 450 TB	\$0.11 per GB
...	...

Table 4: Amazon storage prices

2.3 Materialized views

In Database Management Systems, a view is a virtual table associated to a query answer. Views help indirectly save complex queries, present the same data in different forms, support logical independence and reinforce security by masking some pieces of data from unauthorized users. Materializing a view, i.e., storing it physically into a table, further helps improve response time by avoiding to recompute the corresponding query each time the view itself is queried. However, materialized views must be refreshed when source data are updated, which induces some maintenance overhead.

In this work, we assume that we have at our disposal a set of candidate materialized views that have already been generated by an existing materialized view selection method (e.g., [8]). We aim at choosing the best candidates with respect to the cloud's pay-as-you-go model, taking pricing constraints into account before any view materialization.

3. DATA MANAGEMENT COST MODELS

This section presents general cost models for data management in the cloud, i.e., without considering the use of materialized views. In cloud computing, customers rent resources to a CSP to run some applications. Figure 1 recalls the costs involved (Section 2.2), i.e., bandwidth consumption for input data transfers and query result retrieval, data storage, and applications' processing time.

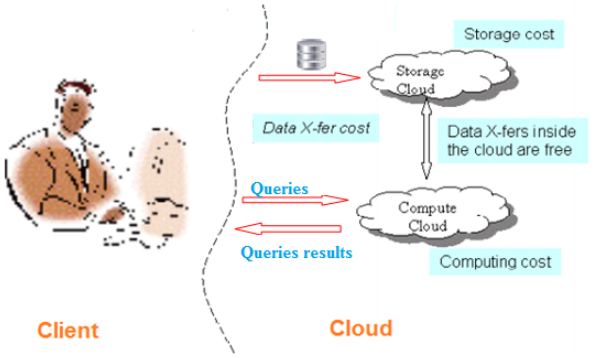


Figure 1: Costs involved in cloud data management

Let C_c be the sum of computing costs, C_s be the sum of storage costs and C_t be the sum of data transfer costs. Then, the total cost C for cloud data management is:

$$C = C_c + C_s + C_t. \quad (1)$$

Let us define the general parameters that we use to express our cost models (Table 5), as well as two functions. Function $s()$ returns the size in GB of any of these parameters, e.g. $s(DS)$ is the size of the dataset. Function $ts()$ returns the storage time of any dataset, e.g. $ts(DS)$ is the storage time of the dataset in the cloud.

Parameter	Description
DS	Dataset
$Q = \{Q_i\}_{i=1..m}$	Query workload
$R = \{R_i\}_{i=1..m}$	Set of query answers

Table 5: General parameters

3.1 Data transfer cost

Data transfer cost depends on several parameters: the size of the initial dataset, the amount of data related to queries and query results, the volume of inserted new data and the pricing model applied by the CSP. The total data transfer cost C_t is the product of the CSP's atomic transfer cost c_t by the total size of transferred data:

$$C_t = (\sum_{i=1}^{m}(s(R_i) + s(Q_i)) + (s(DS) + s(insertedData)) \times c_t. \quad (2)$$

If we adopt Amazon's EC2 like pricing model, which does not charge for input data transfers, we may ignore input queries, the initial data set and inserted data. As a consequence, the total data transfer cost becomes:

$$C_t = \sum_{i=1}^{m} s(R_i) \times c_t. \quad (3)$$

Moreover, recall that Amazon EC2's cost is variable. It is null for the first GB. Then, it becomes \$0.12 up to 10 TB, and so on (Section 2.2).

Example 1: In our running example, with 10 GB (with the first one being free) of bandwidth consumption, data transfer cost is: $C_t = s(R_Q) \times c_t = (10 - 1) \times 0.12 = \$1.08$.

3.2 Computing cost

Let us pose that queries are executed on computing instances $\{IC_j\}_{j=1..n}$. Each instance may bear different performances (with respect to its number of CPUs, its available RAM, etc.), and thus different costs. The cost for renting instance IC_j is denoted $c(IC_j)$. It must be paid at each connection to the cloud. Processing time of query Q_i on instance IC_j is denoted t_{ij}. Then, the processing cost of running the set of queries $Q = \{Q_1, ..., Q_m\}$ is:

$$C_c = \sum_{i=1}^{m} t_{ij} \times c(IC_j). \quad (4)$$

Example 2: In our running example, let us consider that the query workload Q is processed on two small instances. Then, its processing cost is: $C_c = t \times c(IC) + t \times c(IC) = RoundUp(50) \times 0.12 + RoundUp(50) \times 0.12 = 50 \times 0.12 + 50 \times 0.12 = \$12$. We must use a function to round processing time up because every started hour is charged.

3.3 Storage cost

Storage cost depends on parameters such as the CSP's pricing policy, the size of the data (original dataset and inserted data) and the storage time. We assume that the storage period in the cloud is devided into intervals. In each interval, the size of the stored data is fixed. The total storage cost is the CSP's storage cost function cs(s(DS)) (where DS is the data storage size, that is to say the size of the original data set and the inserted data) multiplied by the sum of sizes of the initial dataset and inserted data multiplied by their respective storage time during the intervals.

$$C_s = \sum_{intervals} cs(DS) \times (t_{end} - t_{start}) \times s(DS) \qquad (5)$$

Where t_{start}, t_{end} are start point and end point of an interval.

Example 3: Using Amazon S3 for storage pricing (referred to table 4) and considering that 0.5 TB (512 GB) data has been stored for 12 months. At the beginning of the eighth month, we insert 2 TB (2048 GB) of new data in the cloud. Thus we have two intervals. The storage cost is: $C_s = \sum_{intervals} cs(DS) \times (t_{end} - t_{start}) \times s(DS) = (512 \times 0,14 \times (7-0) + (512+2048) \times 0,125 \times (12-7) = \$2131.76$.

4. COST MODELS FOR MATERIALIZING VIEWS IN THE CLOUD

This section presents cost models for materializing views in the cloud, relying on the costs models developed in Section 3. We assume here that queries are executed on a constant number, nb_{IC} of identical instances IC. In future work, we shall consider the evaluation process on multiple, variable instances.

Let $V_{cand} = \{V_k\}_{k=1..p}$ be a set of candidate materialized views output by any existing materialized view selection technique.

4.1 Data transfer cost

We assume that materialized views are selected on the client's side, outputting final set of materialized views V that are materialized in the cloud. This tactic helps save bandwidth and benefit from the computing performance of the cloud. With materialized views created in the cloud, transfer costs of materialized views are null. As a consequence, total transfers cost C_t is not impacted and remains expressed by Formula 3.

4.2 Computing cost

Using materialized views implies modifying computing costs, since query processing may exploit materialized views, and views must be materialized and maintained. Computing cost expressed by Formula 4 thus becomes:

$$C_c = C_{processingQ} + C_{maintenanceV} + C_{materializationV} \qquad (6)$$

where $C_{processingQ}$ is the cost of processing queries, $C_{maintenanceV}$ is the cost of materialized views' maintenance, and $C_{materializationV}$ is the cost of materializing views.

4.2.1 Materialization cost

If a view is materialized, its associated query must be executed, which must be paid for in the cloud. Let the materialization time of view V_k be $t_{materialization}(V_k)$. We assume that all views are materialized on a constant number of identical instances, nb_{IC}.

The total materialization time is:

$$T_{materializationV} = \sum_{k=1}^{p} t_{materialization}(V_k). \qquad (7)$$

According to Formula 4, materialization cost is:

$$C_{materializationV} = T_{materializationV} \times c(IC) \times nb_{IC}. \qquad (8)$$

Example 4: In our running example with two nodes, we assume that $V = \{V_1\}$, the total materialization time is $T_{materializationV} = \sum_{k=1}^{p} t_{materialization}(V_k) = t_{materialization}(V_1) = 1$ hour. Then, materialization cost is: $C_{materialisationV} = t_{materialisation}(V_1) \times c(IC) \times nb_{IC} = 1 \times 0.12 \times 2 = \$0.24$.

4.2.2 Query processing cost

When using materialized views, query processing time is defined by two main parameters: query workload Q and set of materialized views V. Queries may use the contents of materialized views instead of recomputing their result. Note that we consider that Q is fixed. Since views are usually materialized with respect to a given workload and ours is fixed, then V is also fixed.

Let t_{iV} be the processing time of query Q_i when exploiting the set of materialized views V.Thus, the total processing time of Q against V is:

$$T_{processingQ} = \sum_{i=1}^{m} t_{iV}. \qquad (9)$$

Example 5: In our running example, total processing time when using V is 40 hours.

After taking view materialization into account, Formula 4 becomes:

$$C_{processingQ} = T_{processingQ} \times c(IC) \times nb_{IC}. \qquad (10)$$

Example 6: So, the computing cost becomes: $C_{processingQ} = \sum_{i=1}^{m} t_{iV} \times c(IC) \times nb_{IC} = 40 \times 0.12 \times 2 = \$9.6$.

4.2.3 Maintenance cost

The maintenance cost of materialized views is directly proportional to the time required for updating materialized views when they are impacted by modifications of the source dataset. Note that we consider that querying and maintenance do not occur at the same time. For example, queries are posed during day-time and maintenance is performed during night-time. Let the maintenance time of view V_k be $t_{maintenance}(V_k)$. Then, the total maintenance time of V is:

$$T_{maintenanceV} = \sum_{k=1}^{p} t_{maintenance}(V_k). \qquad (11)$$

Example 7: The maintenance time in the running example is 5 hours.

As a consequence, the cost of maintaining the set of materialized views V is:

$$C_{maintenanceV} = T_{maintenanceV} \times c(IC) \times nb_{IC}. \qquad (12)$$

Example 8: Thus, the maintenance cost is: $C_{maintenanceV} = T_{maintenanceV} \times c(IC) \times nb_{IC} = 5 \times 0.12 \times 2 = \$1.2$.

4.3 Storage cost

Using materialized views does not impact the storage cost as presented by Formula 5. Exploiting materialized views to enhance query performance implies storing them in the cloud and pay the corresponding cost. As a consequence, some data can be duplicated. In that case DS represents the data set, the duplicated data due to materialized views and possible inserted data. Note that original data and materialized views are stored for the whole considered storage period.

Example 9: In our running example, the data set has been stored for a year, the size of duplicated data due to materialized views is 50 GB. In addition, no data are inserted during the considered period. Thus, we have a single interval and storage cost is $C_s = ((500+50) \times (12-0) \times 0.14 = \$924$

5. OPTIMIZATION PROCESS

In this section, we investigate how the use of materialized views in the cloud impacts query performance, while trying to minimize overhead storage cost. We propose an algorithm to select the best set of materialized views by exploiting the cost models introduced in Section 4.

5.1 Objective functions

Base on the ideas in [20], we distinguish in this section three scenarios, i.e., three objective functions, with respect to the needs and capacity of customers (budget limit, response time limit).

5.1.1 Budget limit

Given a predefined financial budget Bl, our objective in this scenario (denoted MV_1) is to select, from the set of candidate materialized views V_{cand}, the set of views to materialize in the cloud V that minimizes query processing time (Figure 2: The chosen solution are colored in red).

$$MV_1 = \begin{cases} \text{Minimize } T_{processingQ} \\ C = C_c + C_t + C_s \leq Bl \end{cases} \tag{13}$$

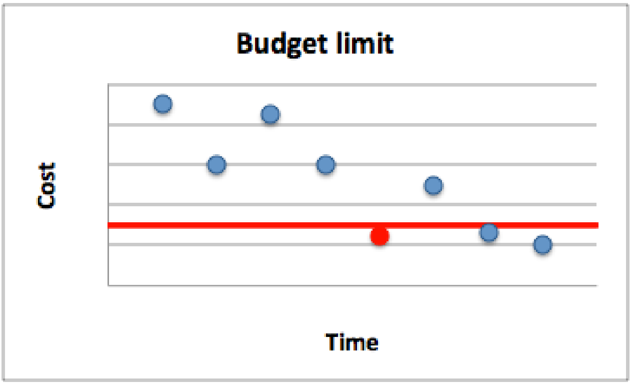


Figure 2: Minimizing processing time under budget constraint

5.1.2 Response time limit

Given a predefined response time limit Tl, our objective in this scenario (denoted MV_2) is to select, from the set of candidate materialized views V_{cand}, the set of views to materialize in the cloud V that minimizes financial cost (Figure 3).

$$MV_2 = \begin{cases} \text{Minimize } C = C_c + C_t + C_s \\ T_{processingQ} \leq Tl \end{cases} \tag{14}$$

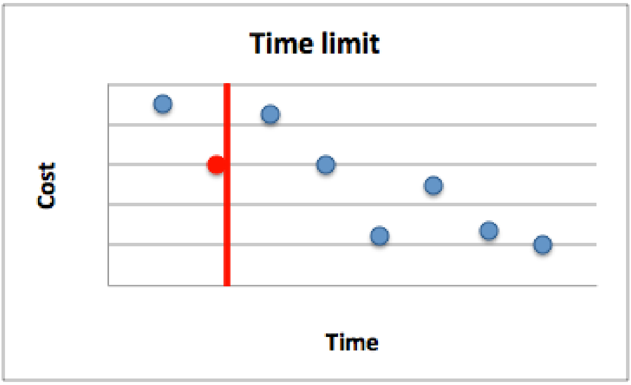


Figure 3: Minimizing cost under response time constraint

5.1.3 Response time vs. cost tradeoff

In this scenario (denoted MV_3), our objective is to select, from the set of candidate materialized views V_{cand}, the set of views to materialize in the cloud V that offers the best tradeoff between query processing time and financial cost (Figure 4). To provide the user with control over this process, we introduce a weight parameter on processing time (α) and cost $(1-\alpha)$.

$$MV_3 = \text{Minimize } (\alpha \times T_{processingQ} + (1-\alpha) \times C) \tag{15}$$

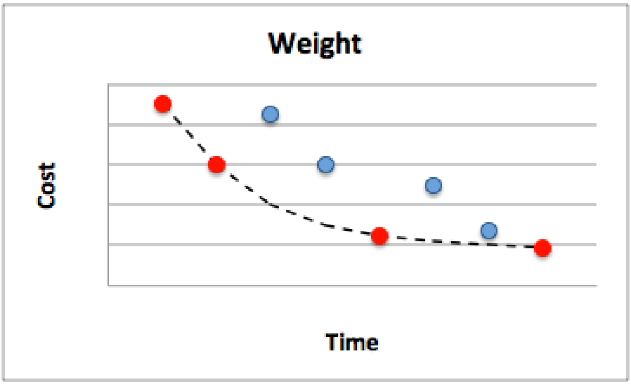


Figure 4: Optimizing response time and cost

5.2 Optimization algorithm

In order to select the set of views to materialize, we rely on an existing algorithm such as [8] enabling to obtain a set of candidate views V_{cand}. Then we solve the Knapsack 0/1 problem [14] considering this set. To solve this problem, we have opted for a dynamic programming approach.

As a input, we provide elements of our cost models. For example, in the case of MV1 (budget limit), the result is: $V = Knapsack(V_{cand}, F_{Ct}, F_{Cc}, F_{Cs}, Bl)$, where F_{Ct}, F_{Cc}, F_{Cs} are functions which represent the parameters of the models described in the section 4. In particular, F_{Ct} involves transfer cost parameters (section 4.1), F_{Cc} involves computing cost parameters(section 4.2), and F_{Cs} involves storage cost parameters (section 4.3).

6. EXPERIMENTAL VALIDATION

In this section, we describe the overall setup of our preliminary experimentation effort and the results we have obtained from it. This experiment is realized on both sides: in the cloud and on the client's side. The dataset and materialized views are stored in the cloud, and queries are processed in the cloud. The view selection algorithms are implemented in Java on the client's side. The idea is to select views on the client's side and to materialize them in the cloud.

6.1 Experimental setup

The experiments were conducted on a virtual cluster composed of five virtual machines with a 50 GB disk, 2 GB of RAM and 4 vCPU. This virtual cluster is deployed on a physical architecture consisting of two bi-pros 493 GHz with 32 GB of RAM, interconnected by a 1 GB/s network in a 8x2 DRS cluster; and a bi-pro 2,526 GHz with hyperthreading enabled, outside of the DRS cluster, with 16 logical cores with 72 GB of RAM on a 1 GB/s network. All machines feature Hadoop (version 0.20.2) [3] and Pig Latin (version 0.7.0) [23]. Queries are written in Pig Latin and are executed by the Pig compiler on the Hadoop cluster within the MapReduce framework [17]. On the client's side, algorithms are implemented on a PC Dual Core 2 T6660 2.2GHz, with 2 GB of RAM.

We used in our experiments a subset of the dataset from our case study (Section 2.1), whose size is 10 GB. The workload we applied on the dataset was made of 10 queries that calculate the total profit per day, month, year and per country, department, and region such as "per year and per country".

6.2 Experimental results

We experimented the three scenarios MV_1, MV_2 and MV_3 described in Section 5.1, with a variable workload made of 3, 5, and 10 queries, with and without exploiting materialized views. Our results are plotted on Figure 5. They clearly show that, in all scenarios, creating materialized views in the cloud is desirable.

In scenario MV_1, a significantly better response time is achieved with materialized views, under the same budget constraint than without (Figure 5(a)). The improved performance rates are shown in the Table 6.

Number of queries	Budget limit	IP Rate
3	\$0.8	25%
5	\$1.2	36%
10	\$2.4	60%

Table 6: Improved performance rates between with and without materialized views under the same budget limit in the scenario MV_1

In scenario MV_2, a significantly lower global cost is achieved with materialized views, under the processing time constraint than without (Figure 5(b)). The improved cost rates are shown in the Table 7.

Number of queries	Time limit	IC Rate
3	0.57h	75%
5	0.99h	72%
10	2.24h	75%

Table 7: Improved cost rates between with and without materialized views under the same time limit in the scenario MV_2

Similarly, in scenario MV_3, materialized views help achieve a tradeoff (chosen by the client) between cost and response time, whether the priority is put on cost (Figure 5(c)) or response time (Figure 5(d)). Improved tradeoff rates are shown in the Table 8.

Number of queries	Rate ($\alpha = 0.3$)	Rate ($\alpha = 0.7$)
3	55%	32%
5	50%	35%
10	68%	45%

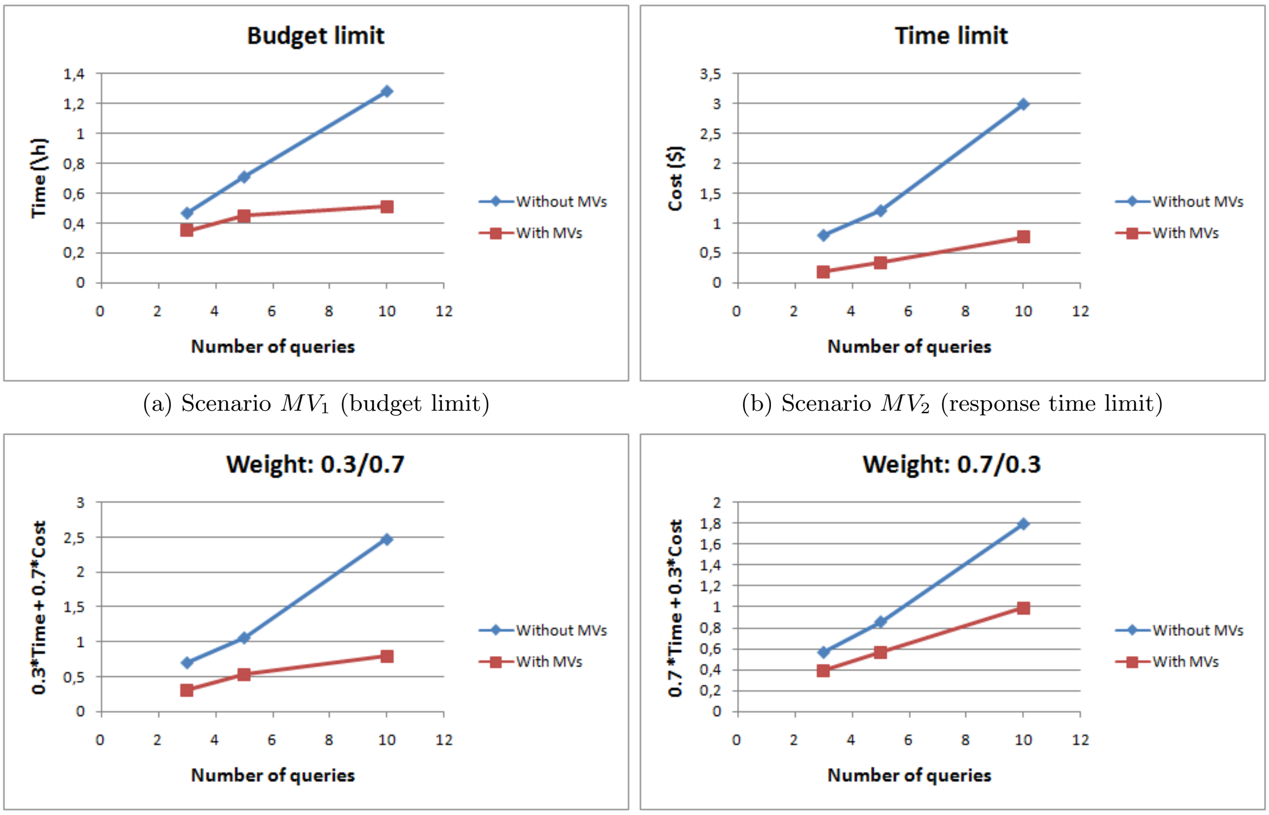
Table 8: Improved tradeoff rates between with and without materialized views in the scenario MV_3

7. RELATED WORK

Although there has been a lot of research related to the cloud for a couple of years, relatively few approaches relate to the optimization of data management in the cloud. In this section, we provide a brief overview of cost models, database-like features and the use of optimization techniques in the cloud.

Cost models have been proposed in the cloud for scheduling of dataflows with regard to monetary cost and/or completion time [20], and cost amortization of data structures to ensure the economic viability of the provider [19], particularly for self-tuned caching [16]. Other cost models have been developed for a real-life astronomy application using the Amazon cloud fee structure [9, 18]. The cost performance tradeoffs of different execution and resource provisioning plans have been simulated, showing that by provisioning the right amount of storage and computing resources, cost can be significantly reduced with no significant impact on application performance [18]. The performance of three workflow applications with different I/O, memory and CPU requirements has also been compared on Amazon EC2 and a typical high-performance cluster (HPC) to identify what applications achieve the best performance in the cloud at the lowest cost [9].

Recent research takes interest in various aspects of database and decision support technologies in the cloud. For example, different studies investigate the storage and processing of structured data [13], the optimization of join queries, and how to support analysis operations such as aggregation [15]. Cloud data warehousing and OLAP systems also raise various problems related to storage and query performance [22]. Adaptations of these technologies to the cloud's specificities are also addressed, e.g., pay-as-you-go pricing and elasticity [7], or the calculation of OLAP cuboids using the MapReduce runtime environment [25].



(a) Scenario MV_1 (budget limit)

(b) Scenario MV_2 (response time limit)

(c) Scenario MV_3 (tradeoff) with $\alpha = 0.3$

(d) Scenario MV_3 (tradeoff) with $\alpha = 0.65$

Figure 5: Experimental results of application of materialized views

Finally, although massive scale distributed database systems such as BigTable have begun to incorporate indexes and materialized views [11], the many techniques developed for maintaining materialized views in traditional, relational databases [5], such as incremental view maintenance [12], deferred maintenance [27] and distributed view maintenance [21], have not been incorporated into the cloud yet. This is thus the main incentive for our own approach.

8. CONCLUSION

In this paper, we proposed an approach for decreasing the cost of data management in the cloud, by using a classical database performance optimization technique, i.e., view materialization. Our main contributions are novel cost models that complement the existing materialized view cost models with a monetary cost component that is primordial in the cloud. Then, we exploit these cost models into an optimization process that achieves a tradeoff between computing power enhancement and view materialization, under budget constraints. Our first experimental validation hints that view materialization is always desirable.

Many perspectives are opened by this preliminary research. We indeed have many plans on improving our cost models. First, we should include pricing models from several CSPs but Amazon, to render our cost models more versatile. Second, it is well-known that optimization techniques are the most efficient when combined, e.g., when materialized views and index are jointly exploited [6]. Thus, we should incorporate indexing, caching and/or fragmentation into our approach. Third, we should exploit the specificities of cloud languages (such as HiveQL [26], Jaql [10] and Pig Latin [23]) with respect to semantics and parallelism to further improve performance. Finally, in this paper, we worked on one single cloud instance, thus minimizing the primordial elasticity characteristic of the cloud. Thus, we should expand our cost models on variable resources.

Another line of future work relates to the experimental validation of our proposals. Here, due to limited resources, we experimented our cost models on a small cluster and a toy dataset. Thus, we plan to design a wider-scale experimentation plan that would exploit a full-fledged database or data warehouse benchmark, such as TPC-E [4] or the Star Schema Benchmark [24], onto AWS or other infrastructure providers.

Acknowledgment

This work is sponsored by the ANR under grant SYSEO ANR-10-TECSAN-005-01. We would like to sincerely thank all the colleagues in IRSTEA, LIMOS and ERIC laboratories for the interesting discussions and the reviewers for their extremely insightful remarks for improving this paper.

9. REFERENCES

[1] Amazon ec2. http://aws.amazon.com/ec2/, February 2012.

[2] Amazon s3. http://aws.amazon.com/s3/, February 2012.

[3] Hadoop. http://hadoop.apache.org/, February 2012.

[4] Tpc-e. http://www.tpc.org/tpce, February 2012.

[5] P. Agrawal, A. Silberstein, B. F. Cooper, U. Srivastava, and R. Ramakrishnan. Asynchronous view maintenance for vlsd databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 179–192, Providence, USA, 2009.

[6] K. Aouiche and J. Darmont. Data mining-based materialized view and index selection in data warehouses. *Journal of Intelligent Information Systems*, 33(1):65–93, 2009.

[7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[8] X. Baril and Z. Bellahsene. Selection of materialized views: A cost-based approach. In *International Conference on Advanced Information Systems Engineering*, pages 665–680, Klagenfurt, Austria, 2003.

[9] G. B. Berriman, E. Deelman, G. Juve, M. Regelson, and P. Plavchan. The application of cloud computing to astronomy: A study of cost and performance. *CoRR*, abs/1010.4813, 2010.

[10] K. S. Beyer, V. Ercegovac, R. Gemulla, A. Balmin, M. Y. Eltabakh, C.-C. Kanne, F. Özcan, and E. J. Shekita. Jaql: A scripting language for large scale semistructured data analysis. *PVLDB*, 4(12):1272–1283, 2011.

[11] M. J. Cafarella, E. Y. Chang, A. Fikes, A. Y. Halevy, W. C. Hsieh, A. Lerner, J. Madhavan, and S. Muthukrishnan. Data management projects at google. *SIGMOD Record*, 37(1):34–38, 2008.

[12] S. Ceri and J. Widom. Deriving production rules for incremental view maintenance. In *Proceedings of the International Conference on Very Large Data Bases, September*, pages 577–589, Barcelona, Spain, 1991.

[13] D. Chatziantoniou and E. Tzortzakakis. Asset queries: a declarative alternative to mapreduce. *SIGMOD Record*, 38(2):35–41, 2009.

[14] V. Chvatal. Hard knapsack problems. *Operations Research*, 28:1402–1411, 1980.

[15] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, J. Gerth, J. Talbot, K. Elmeleegy, and R. Sears. Online aggregation and continuous query support in mapreduce. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1115–1118, Indianapolis, USA, 2010.

[16] D. Dash, V. Kantere, and A. Ailamaki. An economic model for self-tuned cloud caching. In *Proceedings of the International Conference on Data Engineering*, pages 1687–1693, Shanghai, China, 2009.

[17] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[18] E. Deelman, G. Singh, M. Livny, G. B. Berriman, and J. Good. The cost of doing science on the cloud: the montage example. In *Proceedings of the ACM/IEEE Conference on High Performance Computing*, page 50, Austin, USA, 2008.

[19] V. Kantere, D. Dash, G. Gratsias, and A. Ailamaki. Predicting cost amortization for query services. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 325–336, Athens, Greece, 2011.

[20] H. Kllapi, E. Sitaridi, M. M. Tsangaris, and Y. E. Ioannidis. Schedule optimization for data processing flows on the cloud. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 289–300, Athens, Greece, 2011.

[21] G. Luo, J. F. Naughton, C. J. Ellmann, and M. Watzke. A comparison of three methods for join view maintenance in parallel rdbms. In *Proceedings of the International Conference on Data Engineering*, pages 177–188, Bangalore, India, 2003.

[22] H. Mahboubi and J. Darmont. Enhancing xml data warehouse query performance by fragmentation. In *Proceedings of the ACM Symposium on Applied Computing*, pages 1555–1562, Honolulu, USA, 2009.

[23] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1099–1110, Vancouver, BC, Canada, 2008.

[24] P. E. O'Neil, E. J. O'Neil, X. Chen, and S. Revilak. The star schema benchmark and augmented fact table indexing. In *Proceedings of the TPC Technology Conference on Performance Evaluation and Benchmarking*, pages 237–252, Lyon, France, 2009.

[25] K. Sergey and K. Yury. Applying map-reduce paradigm for parallel closed cube computation. In *Proceedings of the International Conference on Advances in Databases, Knowledge, and Data Applications*, pages 62–67, Gosier, France, 2009.

[26] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive - a petabyte scale data warehouse using hadoop. In *Proceedings of the International Conference on Data Engineering*, pages 996–1005, Long Beach, USA, 2010.

[27] J. Zhou, P.-Å. Larson, and H. G. Elmongui. Lazy maintenance of materialized views. In *Proceedings of the International Conference on Very Large Data Bases*, pages 231–242, Vienna, Austria, 2007.