IEEE International Conference on Data Engineering

An economic model for self-tuned cloud caching

Debabrata Dash
Carnegie Mellon University
5000 Forbes Avenue, PA 15213
ddash@cmu.edu

Verena Kantere
École Polytechnique Fédérale de Lausanne
1015 Lausanne, Switzerland
verena.kantere@epfl.ch

Anastasia Ailamaki
École Polytechnique Fédérale de Lausanne
1015 Lausanne, Switzerland
anastasia.ailamaki@epfl.ch

***Abstract*—Cloud computing, the new trend for service infrastructures requires user multi-tenancy as well as minimal capital expenditure. In a cloud that services large amounts of data that are massively collected and queried, such as scientific data, users typically pay for query services. The cloud supports caching of data in order to provide quality query services. User payments cover query execution costs and maintenance of cloud infrastructure, and incur cloud profit. The challenge resides in providing efficient and resource-economic query services while maintaining a profitable cloud. In this work we propose an economic model for self-tuned cloud caching targeting the service of scientific data. The proposed economy is adapted to policies that encourage high-quality individual and overall query services but also brace the profit of the cloud. We propose a cost model that takes into account all possible query and infrastructure expenditure. The experimental study proves that the proposed solution is viable for a variety of workloads and data.**

I. Introduction

The new trend for service infrastructures in the IT domain is called *cloud computing*, a style of computing that allows Internet users of any expertise to access information services on the web. Information, as well as software is permanently stored in Internet servers and probably cached temporarily on the user side, having little or no centralized infrastructure.

Public archiving of large persistent scientific datasets from various disciplines, such as geophysical data, environmental, biological, astronomical etc, gains more and more credence. Data is massively collected and queried by large groups of scientists. Therefore, cloud computing seems to be the perfect data management infrastructure. Such a cloud necessitates various technological capabilities. Most importantly, it is required that the cloud service runs with minimal capital expenditure, but, nonetheless, can support efficiently multi-user tenancy.

A cloud of databases that archives scientific data supports cloud caching, meaning common unrestricted caching for all cloud data. Users are customers of the cloud that consume its resources as a utility service. Specifically, the users can query the cloud data, paying the price for what they use. User payment is employed for coverage of short and long term cost. Short-term cost refers to the respective query execution, and long-term cost to the self-preservation of the cloud infrastructure and improvement of the cloud services.

Emerging IT business in cloud computing [6]–[8] is an effort to offer such network services. Nevertheless, research on cloud computing currently considers an infrastructure that comprises a set of independent edge caches that cooperate in order to deliver web content [1], [18], [19]. Concerning the management of scientific data, existing research solutions [14], consider network bandwidth to be the only important resource, and, therefore, the sole basis for cost computation. However, cloud businesses usually prorate cost to more types of resources. For instance, GoGrid [8] gives network bandwidth for free.

We aim to coalesce the existing research or business technologies and propose an all-inclusive solution for the management of scientific data in the cloud. This paper proposes a self-tuned economy for a cloud infrastructure that serves scientific data. This economy achieves minimal capital expenditure and, at the same time, the service of multiple users in an efficient but also resource-economic way.

Following the business line in cloud computing, the proposed economy employs a cost model that takes into account all the available resources in a cloud, such as disk space and I/O operations, CPU time and network bandwidth. The economy is self-tuned to the policies that aim to:(i) high quality of individual query service, (ii) increasing overall quality of query services, and (iii) cloud profit. According to the first policy, users are satisfied with the received query services given the amount of money they are charged. According to the second, the overall query service is gradually ameliorated so that individual query charges are minimized. According to the third, the cloud infrastructure remains profitable at all times. This goal is accomplished through a methodical resolution of tradeoffs for imminent profit and for future investment. The experimental study shows that the proposed economy is competitive to existing solutions, but most importantly, very promising for a variety of query-workload and data.

The rest of this paper is structured as follows. Section II summarizes the related work. Section IV describes the details of the proposed economy. Section VI presents a discussion on the viability of the proposed solution. Section V adds details on the infrastructure cost. Section VII presents the experimental results and Section VIII concludes this paper.

II. Related Work

Bypass-yield caching [14] relies on reducing network traffic where querying scientific data. A self-tuned cache [25] reduces query execution costs adaptively to the workload. However, as a step further towards commercial applications, the economic model proposed in this paper takes into consideration the overall cost of the infrastructure beyond network bandwidth:

1084-4627/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDE.2009.143

disk I/O and storage, as well as CPU.

Researchers have considered cloud caching as an infrastructure that comprises a set of edge caches that cooperate in order to deliver web content. Document sharing among caches [1], [18], [19] involves (i) document retrieval from sibling caches instead from the server, (ii) routing and sharing of document updates, and (iii) sharing cache resources, in order to achieve efficient collaborative data placement/ replacement/update/lookups etc. Being more compliant with the business domain [6]–[8], this paper focuses on self-tuned cloud caches that share resources, rather that self-organization of independent caches.

Cloud computing is the natural dilation of grid computing [4], as it enables an integrated collaborative use of high-end computing owned and managed by multiple organizations. Grid databases [17] are federated database servers over a grid, which are viewed as a virtual database system through a service federation middleware [26]. Querying the grid data guaranteeing high performance is one of the key issues for distributed queries across large data sets. Another issue is the provision of an accounting service that supports a payment scheme for Grid usage.

Accounting in wide-area networks that offer distributed services have long been the target of research in computing. Mariposa [22] discusses an economic model for querying and storage in a distributed database system. In Mariposa clients and servers have an account in a network bank and users allocate a budget to each of their queries. The processing mechanism aims to service the query in the alloted budget by executing portions of it on various sites. The latter place bids for the execution of query parts, and the bids are accumulated in query brokers. The decision of selecting the most appropriate bids is delegated to the user. In contrast, our work recommends to the user an efficient but also profitable for the cloud query plan.

In the spirit of Mariposa, a series of other works have proposed solutions for similar frameworks [2], [3], [11], [12], [15], [16], [27]. These focus on job scheduling and bid negotiation, which are orthogonal issues to those tackled by the present work.

III. An altruistic cloud DBMS

We assume that data are stored in a *cloud of databases* and that cloud caching supports efficient query processing on the data. Users pose queries to the cloud that are charged in order to be served. Service of queries is performed by executing them either in the cloud cache or on the cloud databases. Query performance is measured in terms of execution time. Naturally, query execution is accelerated with the number of available physical structures (indices, views, partitions. etc) and the amount of cached data. The faster the execution, the more expensive the service. The cloud is *altruistic* in the sense that its intention is not to increase the cloud profit, but provide good quality query services at low cost. Quality increment of services is achieved with investments on new cache structures. Possible repay failure f new inventory must be avoided. Formally:

Definition 1: An altruistic cloud of databases holds an economy along the lines of the following policies: (i) individual user satisfaction with the query services they receive w.r.t. the money they are charged, (ii) increasing overall quality of query services based on new investments, and (iii) minimization of risk of investment loss. The prioritization of the policies is from first to last: (i) $\succ$ (ii) $\succ$ (iii).

The user defines her preferences concerning the service of her query by indicating the budget she is willing to spend on the query, according to the respective execution time that the cloud can provide. The cloud computes a set of query plans suitable to run the query. The price of these query plans is estimated and juxtaposed to the preferences of the user. If there are query plans that the user can afford, according to her budget, then the cloud chooses the most appropriate one of them, w.r.t. the supported policies. If all alternative query plans are priced above the user budget, the user is presented with the alternative options and can choose which one she is willing to pay for. The profit from each query service is credited to the cloud account and spent in building physical structures and cache data that can improve the query services.

IV. Economy of the cloud

In this section we describe in detail the economy of the cloud, as well as the cost model employed for the price estimation of the query plans.

A. Rationale of an altruistic economy

The cloud has an account where the user payments for the query services they receive are deposited. Also, money from this account are used in order to invest on new inventory, i.e. cache structures that can be later on used in order to execute queries in the cache, faster and cheaper. The overall credit amount in this account is denoted as CR. It is not the intension of an altruistic cloud to make profit from its investments. However, some moderate profit is made in cases that the user is willing to pay more for the provided services than they really cost, without taking advantage of the difference of offered compensation and real cost for services. Moreover, the cloud aims and functions towards the direction of total amortization of investment cost.

The investment on a specific new cache structure is decided based on the notion of *regret*, inspired by [23]. Essentially, the absence of a structure from the cloud cache leads the cloud to discard query services that employ it. However, in case that this structure is available in the cache, it is possible that the execution of some queries could benefit by employing it, either in terms of time or cost. The regret for not having already built a structure is accumulated and monitored; if this regret becomes substantial, then the cloud decides to invest in the construction of this structure. Formally:

Definition 2: The regret for a structure S that is possible new inventory of the cloud represents the accumulated value of the missed chances to provide better quality query services in terms of either performance or cost.

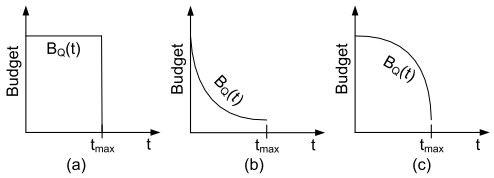

Fig. 1. Typical budget functions; (a) step function: $B_Q(t) = |a|$, (b) convex function: $B_Q(t) < -|a|(1 - t/t_{max}) \cdot t$, (c) concave function: $B_Q(t) > -|a| \cdot (1 - t/t_{max})$.

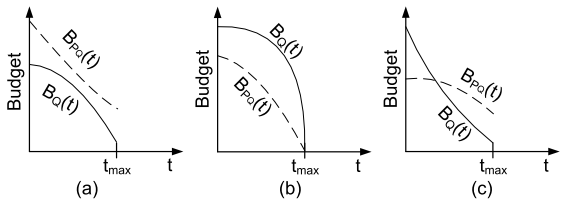

Fig. 2. The types of relationships between the user- and cloud-defined budget functions, corresponding to the respective cases.

The regret leads to improvement of the query services. The cost of the investment on new inventory is payed from the credit in the cloud account. This cost is amortized to prospective users that receive query services which include the new inventory. The aim of the amortization is to reduce the individual cost of these quality query services. Reduction of the cost increases the potential that the offered services are cheaper than the amount that the user is willing to pay for it. In such a case, the cloud benefits from the difference of actual cost and offered user compensation; therefore, the cloud increases its profit, and, thus, its credit CR, which gives the opportunity for more investments directed by regret, leading to even more quality query services.

B. The space of cloud functionality

The cloud maintains a pool of structures relevant to the queries in the recent past. Cache structures considered by the cloud are CPU nodes, table columns and indexes (see Section V). These structures are garbage collected using LRU policy, so that the structure cache can be searched and processed efficiently for each incoming query plan. Upon receiving an incoming query Q, the cloud considers a set of plans, $\mathcal{P}_\mathcal{Q}$. This set consists of two non-overlapping subsets: the set of plans that include only existing cache structures, $\mathcal{P}_{\mathcal{Q}_{exist}}$, and the set of plans that include also possible new cache structures, $\mathcal{P}_{\mathcal{Q}_{pos}}$. The cloud determines the most optimal execution plan in $\mathcal{P}_{\mathcal{Q}_{exist}}$[1] and considers the plans in $\mathcal{P}_{\mathcal{Q}_{pos}}$ for possible investment in new cache structures.

C. Economy management

The user input to the cloud is a query Q as well as a respective budget function $B_Q(t)$, that indicates the price she is willing to pay according to the execution time of the query. There are no limitations for the structure of B_Q, while the user is expected to input a function that is descending with time, and limited to a specific time interval $(0, t_{max}]$: i.e $B_Q(t_1) \geq B_Q(t_2), \forall t_1, t_2 \in (0, t_{max}]\ s.t.\ t_1 < t_2$. Graphically, an expected user budget function is any combination of the three simple types depicted in Figure 1.

The cloud produces a set of alternative query plans for the input query Q, $\mathcal{P}_\mathcal{Q}$[2]. The cost of each query plan $P_Q \in \mathcal{P}_\mathcal{Q}$ is estimated according to the cost model described in Section IV-D, and produces a discrete budget function $B_{P_Q}(t) : t_{\mathcal{P}_\mathcal{Q}} \mapsto \Re^+$, where $t_{\mathcal{P}_\mathcal{Q}}$ is the set of execution time values for all plans in $\mathcal{P}_\mathcal{Q}$. We assume that $B_{P_Q}(t)$ is also descending, i.e. $B_{P_Q}(t_1) \geq B_{P_Q}(t_2), \forall t_1, t_2 \in t_{\mathcal{P}_\mathcal{Q}}\ s.t.\ t_1 < t_2$. The two functions B_Q and B_{P_Q} are compared (for the subspace of time $t_{\mathcal{P}_\mathcal{Q}}$). Figure 2 depicts the comparison of the two functions. A query plan that conforms to the budget of the user, but also supports the three policies defined earlier is chosen, depending on the relationship of B_Q and B_{P_Q}, as follows:

Case A: $B_Q(t) < B_{P_Q}(t) \forall t \in (0, t_{max}] \cap t_{\mathcal{P}_\mathcal{Q}}$
The user budget function defines a budget lower than the actual cost of all possible query plans $\mathcal{P}_\mathcal{Q}$. Therefore, Q cannot be served according to the user's defined budget. The user is presented with B_{P_Q}, i.e. with all the query plans in $\mathcal{P}_{\mathcal{Q}_{exist}}$ accompanied with their cost and the execution time that they guarantee. The user can pick and pay for one or none of $P_{Q_i} \in \mathcal{P}_{\mathcal{Q}_{exist}}$. For all the plans in $\mathcal{P}_{\mathcal{Q}_{pos}}$ that are cheaper that the chosen one, i.e. $\forall P_{Q_j}, j \neq i, P_{Q_j} \in \mathcal{P}_\mathcal{Q}\ s.t.\ B_{P_Q}(t_{P_{Q_j}}) \leq B_{P_Q}(t_{P_{Q_i}})$, the cloud calculates the regret for not investing in this plan, as the difference of the cost of the chosen and the not chosen plan:

$$regret(P_{Q_j}) = B_{P_Q}(t_{P_{Q_i}}) - B_{P_Q}(t_{P_{Q_j}}) \quad (1)$$

In this case, the regret for a rejected query plan concerns the lost chance to offer better query services in terms of cost.

Case B: $B_Q(t) \geq B_{P_Q}(t) \forall t \in (0, t_{max}] \cap t_{\mathcal{P}_\mathcal{Q}}$
The user's budget covers the cost of any query plan in $\mathcal{P}_\mathcal{Q}$. The cloud chooses the query plan in $\mathcal{P}_{\mathcal{Q}_{exist}}$ that minimizes the gain from the user's payment diminished by the actual cost of the plan, i.e. the plan $P_{Q_i} \in \mathcal{P}_\mathcal{Q}$ is picked, such that $B_Q(t_{P_{Q_i}}) - B_{P_Q}(t_{P_{Q_i}}) \leq B_Q(t_{P_{Q_j}}) - B_{P_Q}(t_{P_{Q_j}}), i \neq j, P_{Q_j} \in \mathcal{P}_\mathcal{Q}$. The profit $B_Q(t_{P_{Q_i}}) - B_{P_Q}(t_{P_{Q_i}})$ is credited to the cloud account, and can be invested on new cache structures.

For each plan in $\mathcal{P}_{\mathcal{Q}_{pos}}$ that is more expensive than the chosen one, i.e. $\forall P_{Q_j}, j \neq i, P_{Q_j} \in \mathcal{P}_\mathcal{Q}\ s.t.\ B_{P_Q}(t_{P_{Q_j}}) \geq B_{P_Q}(t_{P_{Q_i}})$, the cloud calculates the regret for not investing in this plan:

$$regret(P_{Q_j}) = B_Q(t_{P_{Q_j}}) - B_{P_Q}(t_{P_{Q_j}}) \quad (2)$$

In this case, the regret for a rejected query plan concerns the lost opportunity to add credit in the cloud account.

Case C: $(\exists t \in (0, t_{max}] \cap t_{\mathcal{P}_\mathcal{Q}}\ s.t.\ B_Q(t) > B_{P_Q}(t)) \wedge (\exists t' \neq t \in (0, t_{max}] \cap t_{\mathcal{P}_\mathcal{Q}}\ s.t.\ B_Q(t') < B_{P_Q}(t'))$
The user's budget covers the cost of some query plans $\mathcal{P}_{\mathcal{Q}_\mathcal{S}} \subset \mathcal{P}_\mathcal{Q}$. The plan to run the query and the regret for the

[1]The optimum plan can be determined by querying the optimizer or by looking up a set of plans cached from earlier queries.

[2]We assume that $\mathcal{P}_\mathcal{Q}$ holds only the skyline query plans (w.r.t. execution time and overall cost); i.e. if there are two plans P_{Q_1} and P_{Q_2} with the same execution time, only the cheapest one is encompassed in $\mathcal{P}_\mathcal{Q}$.

query plans are calculated as in Case B, taking into account only the plans in $\mathcal{P}_{\mathcal{Q}_\mathcal{S}}$.

Once the regret of a plan is computed, it is distributed uniformly to every physical structure used by the plan. The cloud maintains the array $regret_\mathcal{S}$, which stores the accumulated regret values for each physical structure or set of cached data $S \in \mathcal{S}$ that is employed by any plan $P_Q \in \mathcal{P}_\mathcal{Q}$, for any input user query Q. Specifically, the regret for a non-chosen query plan P_Q is added to the positions in $regret_\mathcal{S}$ that correspond to the $S \in \mathcal{S}$ that are employed by P_Q. The accumulated regret value for each S shows the overall regret of the cloud for not employing it in executed query plans. A high regret value indicates that S could have been employed in either numerous or expensive, or both, query plans. When the regret for S, $regret_\mathcal{S}[S]$ reaches a high value, then the cloud makes an investment and constructs S. Assuming that the overall credit in the cloud account is CR, then the $regret_\mathcal{S}[S]$ must be risen to a fraction a of CR, in order for S to be considered for imminent investment:

$$InvestIn(S) = round(\frac{regret_\mathcal{S}[S]}{a \cdot CR}), 0 < a < 1, \quad (3)$$

In the current model we compute and distribute regret of all plans, and distribute their regrets to the physica structures. This allows the cloud to identify the commonly used structures and use them first.

The selection criterion of a query plan in case A is the minimization of user charge and, in cases B and C, the criterion is the minimization of the cloud profit. These criteria serve the policies that require user satisfaction in terms of her budget definition, as well as maintenance of the cloud credit. The accumulation of the regret values and the procedure of investment into new inventory serves the policy that requires increasing overall quality of query services, without contravening the other two policies.

The cloud aims to increase the possibility that the relationship between B_Q and B_{P_Q} falls into the case B. This is achieved by amortizing the cost of new structures to numerous prospective selected query plans. Amortization of cost is described below.

D. Cost Model

The cost C of a query plan P_Q is the sum of the cost of executing the query plan, $C_e(P_Q)$ and the amortized cost of any structure $S \in \mathcal{S}$ used by the query plan, $C_a(P_Q)$.

$$C(P_Q) = C_e(P_Q) + C_a(P_Q) \quad (4)$$

The execution cost of a query plan is analyzed in Section V-B. The amortized cost of a query plan comprises the respective cost for all the structures employed:

$$C_a(P_Q) = \sum_{S \in \mathcal{S}} C_a(S) \quad (5)$$

The amortized cost of a structure S depends on the initial infrastructure cost that is necessary to build it, $Build_S(S)$. This dependence is expressed by a function that also considers

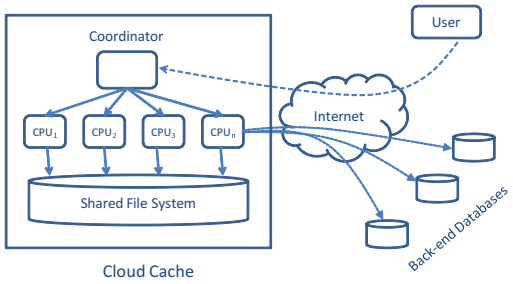

Fig. 3. The architecture of the cloud cache

the number of queries n that benefit from each S, and to which the initial building cost is disseminated to:

$$C_a(I) = f_S(n, Build_S(S)) \quad (6)$$

The building cost $Build_S(S)$ is analyzed in Section V-C. The function f_S quantifies the manner of cost amortization to the n queries that use S. In this work we consider that the initial building cost of S is amortized equally to the n queries, thus:

$$f_S(n, Build_S(S)) = Build_S(S)/n \quad (7)$$

Selecting n is a challenging problem in itself, as it depends on the provider's risk aversion, arrival pattern of the queries, and infrastructure costs. We intend to study this problem in our future research.

V. Infrastructure Details

In this section we describe the infrastructure details of the cloud setting and we analyze the cost for building cache structures and executing query plans.

A. Cloud Infrastructure

The architecture of the cloud which incorporates the proposed economy is shown in Figure 3. The user requests for query execution from the Internet and contacts the Coordinator node. The latter distributes the query to the appropriate CPU node, or to the back-end databases. We assume that the cloud infrastructure provides unlimited amount of storage space, CPU nodes, and very high speed intra-cloud networking. Also, the CPU nodes in the architecture are all identical to each other. Compared to TCP bandwidth on the Internet, the inter-cloud bandwidth is orders of magnitude faster and we ignore the overhead associated with it. The storage system is based on a clustered file system, such as, [5], where the disk blocks are replicated and stored close the CPU nodes accessing them. With this infrastructure we can assume that the virtual disk is a shared resource for all the CPU nodes in the cache.

B. Cost of Queries

Since the cache considers many possible plans, i.e. design configurations, for executing a query, accurate and fast estimation of the cost associated with each plan is very important. The execution time of a query C_e is estimated based on a respective query plan P_Q. For the current cloud setting, we assume that the query runs completely either in the back-end or in the cache. In other words, plans that run partially in the

back-end and then transfer the partially computed results to the cache in order to complete the execution, are out of the scope of this work.

Concerning queries that run in the cache completely, the estimation of the execution cost is determined based on a plan that is suitable for the cache. If the total cost of running the query is q_{tot}, and total I/O cost for in the execution plan is io_{tot}, the cost of running the query is:

$$C_{e_C}(P_Q) = l_{cpu} \cdot f_{cpu} \cdot q_{tot} \cdot c + f_{io} \cdot io \cdot io_{tot} \quad (8)$$

where l_{cpu} is a factor that indicates the overload of the CPU node, and, f_{cpu} and f_{io} are factors that convert the numbers reported by the execution plan to actual CPU time and actual IO operations, respectively. If these factors are stable, their values can be estimated by running a fixed set of simple queries and plotting the actual CPU time and logical disk reads. If these factors change, their values are determined by the actual execution plan after running the queries.

For network queries, the cost is the total cost of running in the back-end database and transferring the result to the cache.

$$C_{e_N}(Q) = C_{e_C}(Q) + f_n \cdot (l + \frac{S(Q)}{t}) + S(Q) \cdot c_b \quad (9)$$

The function $S(Q)$ determines the size of the results for the query Q, t represents the throughput of the network, f_n is the fraction of CPU required to process the incoming results, c_b is the cost of transferring one byte across the network, and l is the network latency.

C. Cost of Structures

In the current caching infrastructure the columns of the original tables in the back-end databases are cached, in order to facilitate a comparison with [14]. Moreover, indexes, constructed in the cache, accelerate the queries. Future work includes the expansion of the infrastructure with caching of materialized views, similar to [25], or partial columns, similar to [21]. Furthermore, additional CPUs are employed to speed up queries. Summarizing, the cache needs to decide on building and maintaining three different types structures: 1) CPU nodes N, 2) table columns T, and 3) indexes I. As discussed in Section IV, the cache needs to spend money for building these structures (and maintain them). For a structure $S \in \{N, T, I\}$ we define the building cost $Build_S$, as well as the maintenance cost $Maint_S$[3].

For CPU nodes, we exploit the scalability of cloud infrastructure and dynamically boot up a system on demand. If b is the time it takes to boot a system and u is the cost of using a system per unit time, then the cost of building the CPU node, N structure is constant:

$$Build_N(N) = b \cdot u \quad (10)$$

[3]As soon as a structure is built in the cache, the query plans that are selected for execution and employ this structure, pay also for its maintenance cost. Each newly selected query plan pays for the accumulated maintenance cost from the time point of the previous query plan that payed off the previously accumulated maintenance cost. Excessive maintenance cost of a structure due to non-usage of it in selected query plans, can be the reason of structure failure.

and the cost of maintaining the node per unit time is also constant:

$$Maint_N(N) = c \quad (11)$$

For a table column, T, the building cost $Build_T(T)$ is the cost of transferring T from the back-end database and combining it with the current columns in the cache. For simplicity, we ignore the cost of integrating T into the existing table in the cache. Therefore the cost of building a table column is primarily the cost of transferring the respective data over the network. Therefore the cloud has to pay for the bandwidth used to transfer that data, and the CPU time taken to manage that transfer. If f_n is the fraction of CPU taken to manage transfer, and t, l are the throughput and latency of the network, the total cost for building a column C in the cache is:

$$Build_T(T) = f_n \cdot (l + \frac{size(T)}{t}) + size(T) \cdot c_b \quad (12)$$

where c_b is the cost of transferring a byte across the network. The maintenance cost of the table column is just the cost of using disk space in the cloud, so if c_d is the cost of disk storage in the cloud, then the maintenance cost for T is:

$$Maint_T(T) = size(T) \cdot c_d \quad (13)$$

For indexes, the building cost involves fetching the table data across the Internet and then building the index in the cache. Since sorting is the most important step in building an index, the cost of building an index is approximated to the cost of sorting the indexed columns. For example, the cost for building an index $I(A, B)$ on table T using columns A and B is approximated to the cost of running the following query:

```
Q = select A, B from T order by A, B
```

The total cost building an index is therefore:

$$Build_I(I) = C_e(P_Q) + \sum_{T \in I \wedge T \notin Cache} Build_T(T) \quad (14)$$

where P_Q is the query plan for query Q and T is a column in the index I but not in the cache.

The function $C_e(P_Q)$, introduced in Section IV-D, determines the cost of running a query Q in the cache based on a plan P_Q, and is described in Section V-B. The last term in the equation, $Build_T(T)$, determines the cost of building the table column in the cache. We assume that the cloud databases in the current setting are static, therefore there is no need for index updating. Hence, the maintenance cost for an index is:

$$Maint_I(I) = size(I) \cdot c_d \quad (15)$$

where c_d is the cost of storing a byte of data on cache per unit time.

VI. Discussion

The viability of the proposed economy depends on the following properties of the workload. The workload running on the databases should be amenable to caching: First, queries have data access locality, i.e. they mostly target a specific part of the data; second, queries have temporal locality, i.e. similar queries are posed close in time. Furthermore, the cache access

is beneficial if the queries are "result heavy", i.e., produce significant amount of data to be transported from the back-end databases to the cache. If there is high update rate in the back-end databases, the overhead of maintaining the cache structures becomes overwhelming, causing detriment to the service provider. Moreover, the cache economy implies that the expected profit depends on the stability of the data; thus, rare or loosely predictable data changes are desired. Finally, to exploit the scalability of the cloud service to the maximum, the queries should be parallelizable.

Scientific data fits the description of the above requirements [10], [20]. A small portion of the data is of intense interest to the users, and the workload is characterized by access patterns that can be exploited by the cache to improve performance by pre-fetching. Furthermore,the queries are parallelizable. Therefore, this work is aims to service massive querying of scientific data collections.

VII. Experiments

We show simulation-based experimental results for a cloud cache system using the economic model developed in Section IV.

A. Experimental Setup and Methodology

We simulate the cloud with just one back-end database. This is the best possible scenario for the back-end database as it eliminates the inter-database communication to answer the queries. The cache is operated under a TPCH-based workload [13], which consists of 7 TPCH query templates and simulates the query evolution of a million SDSS-like [9] queries against a 2.5TB back-end database.

The proposed economic model is compared with bypass-yield cache [14]. The latter is emulated by associating cost only with network bandwidth, therefore setting costs for CPU, disk and I/O to zero. This cache, denoted as *net-only*, tries to reduce the network bandwidth and caches only table columns. The experiments employ the ideal cache size for *net-only*, which is 30% of the total database size [14]. The *net-only* cache avoids using indexes to speed up queries, since the space occupied by the indexes requires some of the cached data to be removed, causing the cache to answer more queries in the back-end database. The *net-only* cache is compared against three variations of the economic model, i.e., *econ-col, econ-cheap, and econ-fast*. The *econ-col* cache is similar to the *net-only* cache, in which query plan execution employs only cached columns and no indexes. The *econ-cheap* cache builds and uses indexes, and adds extra CPU nodes to speed up the queries. Given a set of query plans, the plan with the least cost is chosen. Finally, the *econ-fast* cache is similar to *econ-cheap*, but selects the query plan with the fastest response time.

We assume that the CPU nodes are never overloaded ($l_{cpu} = 1$), and the CPU is fully utilized during data transfer ($f_n = 1$). Also, there is no latency in the network ($l = 0$) and the network throughput between the cache and the back-end database is 25Mbps. The throughput parameter is the maximum throughput between two database nodes for SDSS [24].

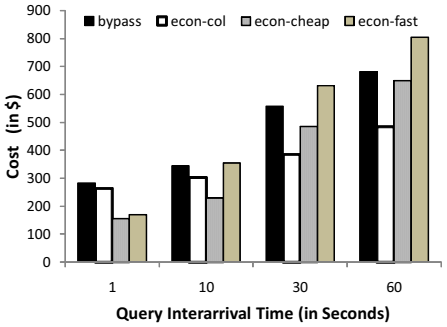

Fig. 4. Comparison of operating costs for caching schemes

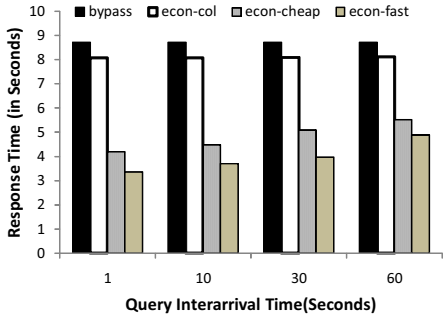

Fig. 5. Comparison of average response time for caching schemes

The response time for the SDSS queries is emulated by setting $f_{cpu} = 0.014$. Query execution scaling to multiple CPU nodes follows the scaling property of a prototypical SDSS query [17]: a query can be sped up 2x using only 25% extra CPU overhead using 3 CPU nodes in parallel. We use 65 potentially useful indexes from DB2's "recommend indexes" mode recommendations. Finally, the cost values for the caching service are imported from Amazon EC2 [6].

The cache provider is conservative and builds structures only when her profit exceeds the cost of building them. The user defines a step preference function B_Q and accepts query execution in the back-end.

B. Experimental Results

Figures 4 and 5 show the operating cost of the caching infrastructure and the average query response time for all four caching schemes using different inter-query time intervals. Measurements are shown for two opposite extreme cases: 1 and 60 seconds duration of inter-query intervals, as well as for the moderate case of 10 seconds. The disc space cost for storage of indexes is very small and significant for the 1 second and 60 seconds measurements, respectively, compared to CPU and network costs. For the case of 10 seconds the disk cost is not prohibitive, but it is enough to allow the algorithms to evict the structures in order to save space.

Figure 4 clearly shows that the cost of operating a cache is reasonable for all caching schemes. Hence, the caching service for SDSS-like workload is viable. Figure 5 shows that the response time of *net-only* and *econ-col* are similar. This is not surprising since they both use only table data to answer the queries. The cost for using these structures, however, is lower for *econ-col*: *net-only* is conservative in terms of caching table

columns and answers many queries over the network before loading the data and answering queries into the cache. Yet, *econ-col* reduces the number of queries answered over network and the CPU overhead of transferring their results, because it considers the overall CPU cost for data transfer and query answering, which leads to earlier data caching. The results for the 1 second query interval show that the disk cost is negligible for this scenario; therefore, the overall reduced cost of the economic model is directly proportional to the cost saved by reduced CPU usage, i.e. approximately 7%. Since *econ-cheap* uses indexes on top of the cached data, the response time is about 50% of *econ-col*. Using inexpensive disks to speed up the queries actually diminishes the operating cost of *econ-cheap* scheme and is about 45% cheaper than *net-only*. *econ-fast* further reduces the response time by approximately 10%, employing extra CPU nodes to parallelize the queries. Yet, the coordinator pays the overhead for the initialization of the extra CPU nodes.

As the time interval increases, the response times of *net-only* and *econ-col* remain similar, due to the fact that they use only cached columns, which are small compared to indexes and they are less eligible for eviction. The response times for *econ-cheap* and *econ-fast* increase with the increment of the inter-query interval, which indicates that these schemes adapt to the workload change efficiently. As the time interval increases, the cost increases, too, because of the extra cost of disk storage for cached data. The cost of *econ-col* is lower than that of *econ-cheap* for the 60-seconds interval, because the first uses less disk space, (the most expensive structure for this scenario), than the latter. Furthermore, the evolution of the workload leads *econ-cheap* to evict indexes already built in the cache, before being able to exploit them sufficiently.

Summarizing, the results show that a comprehensive economic model that considers costs for all resources performs better, in terms of response time and cost, than a model that considers only one resource. The all-inclusive model allows the cloud system to exploit the cheaper resource in order to save on the more expensive ones. Also, it allows the user to trade off cost with faster response times.

VIII. Conclusion

In this paper we propose an economic model for a cloud cache suitable for the querying service of large scientific datasets. The proposed economy is self-tuned to three policies. These ensure high and increasing quality of individual and overall query service, but also, guarantee profit for the cloud. The economy is based on a cost model that takes into account all the necessary infrastructure resources, namely: network bandwidth, disk space and CPU time. The cloud profit from the query services is invested in building physical structures in the cache, which expedite query execution. The presented experimental study shows that the proposed economy is viable for a variety of workloads and data.

References

[1] Samrat Bhattacharjee, Kenneth L. Calvert, and Ellen W. Zegura. Self-organizing wide-area network caches. In *IEEE Infocom'98*, 1998.

[2] Chunming Chen, Muthucumaru Maheswaran, and Michel Toulouse. Supporting co-allocation in an auctioning-based resource allocator for grid systems. In *IPDPS*, pages 89–96, 2002.

[3] Carsten Ernemann, Volker Hamscher, and Ramin Yahyapour. Economic scheduling in grid computing. In *Scheduling Strategies for Parallel Processing*, pages 128–152. Springer, 2002.

[4] I. Foster. What is the grid? a three point checklist. http://www-fp.mcs.anl.gov/ foster/articles/whatisthegrid.pdf, 2002.

[5] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.

[6] http://aws.amazon.com/ec2/.

[7] http://code.google.com/appengine/.

[8] http://www.gogrid.com/.

[9] http://www.sdss.org/.

[10] M. Ivanova, N. Nes, R. Goncalves, and M. Kersten. Monetdb/sql meets skyserver: the challenges of a scientific database. In *SSDBM '07*, page 13, 2007.

[11] Markus Kradolfer and Dimitrios Tombros. Market-based workflow management. *IJCIS*, 7, 1998.

[12] Jiadao Li and Ramin Yahyapour. Negotiation model supporting co-allocation for grid scheduling. In *Intern. Conference on Grid Computing*, pages 254–261, 2006.

[13] T. Malik, X. Wang, R. Burns, D. Dash, and A. Ailamaki. Automated physical design in database caches. In *SMDB*, 2008.

[14] Tanu Malik, Randal C. Burns, and Amitabh Chaudhary. Bypass caching: Making scientific databases good network citizens. In *ICDE*, pages 94–105, 2005.

[15] Vladimir Marbukh and Kevin Mills. Demand pricing & resource allocation in market-based compute grids: A model and initial results. In *ICN*, pages 752–757, 2008.

[16] Rafael A. Moreno. A.B.: Job scheduling and resource management techniques in economic grid environments. In *Across Grids 2003*, pages 25–32, 2004.

[17] Mara A. Nieto-Santisteban, Jim Gray, Alexander S. Szalay, James Annis, Aniruddha R. Thakar, and William J. Omullane. When database systems meet the grid. In *CIDR*, pages 154–161, 2005.

[18] Lakshmish Ramaswamy, Ling Liu, and Arun Iyengar. Cache clouds: Cooperative caching of dynamic documents in edge networks. In *ICDCS-2005*, pages 229–238, 2005.

[19] Lakshmish Ramaswamy, Ling Liu, and Arun Iyengar. Scalable delivery of dynamic content using a cooperative edge cache grid. *IEEE Trans. Knowl. Data Eng.*, 19(5):614–630, 2007.

[20] Arie Shoshani, Frank Olken, and Harry K. T. Wong. Characteristics of scientific databases. In *VLDB*, pages 147–160, 1984.

[21] Jeff Sidell, Paul M. Aoki, Adam Sah, Carl Staelin, Michael Stonebraker, and Andrew Yu. Data replication in mariposa. In *ICDE*, pages 485–494, 1996.

[22] Michael Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin, and Andrew Yu. Mariposa: A wide-area distributed database system. *VLDB J.*, 5(1):48–63, 1996.

[23] Ambuj Tewari and Peter Bartlett. Optimistic linear programming gives logarithmic regret for irreducible mdps. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1505–1512. MIT Press, Cambridge, MA, 2008.

[24] Xiaodan Wang, Randal C. Burns, Andreas Terzis, and Amol Deshpande. Network-aware join processing in global-scale database federations. In *ICDE*, 2008.

[25] Xiaodan Wang, Tanu Malik, Randal C. Burns, Stratos Papadomanolakis, and Anastasia Ailamaki. A workload-driven unit of cache replacement for mid-tier database caching. In *DASFAA*, pages 374–385, 2007.

[26] Paul Watson. Databases and the grid. Technical report, Grid Computing: Making The Global Infrastructure a Reality, 2001.

[27] Michael P. Wellman, William E. Walsh, Peter R. Wurman, and Jeffrey K. Mackie-mason. Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35:2001, 1998.