

Admission Control Mechanisms for Continuous Queries in the Cloud*

Lory Al Moakar^{1‡}, Panos K. Chrysanthis¹, Christine Chung^{2‡}, Shenoda Guirguis¹,
Alexandros Labrinidis¹, Panayiotis Neophytou¹, Kirk Pruhs¹

¹ *Department of Computer Science, University of Pittsburgh*
{lorym, panos, shenoda, labrinid, panickos, kirk}@cs.pitt.edu

² *Department of Computer Science, Connecticut College*
cchung@conncoll.edu

Abstract— Amazon, Google, and IBM now sell cloud computing services. We consider the setting of a for-profit business selling data stream monitoring/management services and we investigate auction-based mechanisms for admission control of continuous queries. When submitting a query, each user also submits a bid of how much she is willing to pay for that query to run. The admission control auction mechanism then determines which queries to admit, and how much to charge each user in a way that maximizes system revenue while being strategyproof and sybil immune, incentivizing users to use the system honestly. Specifically, we require that each user maximizes her payoff by bidding her true value of having her query run. We design several payment mechanisms and experimentally evaluate them. We describe the provable game theoretic characteristics of each mechanism alongside its performance with respect to maximizing profit and total user payoff.

I. INTRODUCTION

The growing need for *monitoring applications* such as the real-time detection of disease outbreaks, tracking the stock market, sensor networks, and personalized and customized Web alerts, has led to a paradigm shift in data processing paradigms, from Database Management Systems (DBMSs) to Data Stream Management Systems (DSMSs) (e.g., [1], [2], [3]). In contrast to DBMSs in which data is stored, in DSMSs, monitoring applications register Continuous Queries (CQs) which continuously process unbounded data streams looking for data that represent events of interest to the end-user.

There are already a number of commercial stand-alone DSMSs on the market, such as Streambase [4], S-stream [5] and Coral8 [6], aiming to support specific applications. We consider the setting of a business that seeks to profit from selling data stream monitoring/management services. One might imagine that a DSMS rents server capacity to clients similar to the way Amazon, Google, and IBM now sell cloud computing services [7], [8], [9]. Auctions, used for example by Google to sell search engine ad words, are a proven way of both maximizing a system's potential profit, as well as appealing to the end-user (client). Instead of a business selling their services at a set price, an auction mechanism (soliciting bids,

then selecting winners) allows a system to charge prices per client based on what the individual client is willing to pay. And perhaps most compellingly, an auction setting allows the system to subtly control the balance between overloading their servers and charging the right prices. Hence, we investigate auction-based mechanisms for admission control of CQs to be serviced by the DSMS center.

One of the key challenges to designing these auction mechanisms is determining how to best take advantage of the shared processing between CQs. The fact that some queries can share resources obfuscates each query's actual load on the system. Without clear-cut knowledge of each query's load on the system, optimally selecting the queries to admit becomes exceedingly challenging from a combinatorial perspective.

From a business point of view, the most obvious design goal for the admission control mechanism is to maximize profit. Another first class design goal for the mechanism is to not be manipulable by users. Specifically, we desire that the mechanism is *strategyproof* (also known as *incentive compatible* or *truthful*), which means a client always maximizes her payoff by bidding her true valuation for having her query run. Auction-based profit-driven businesses like eBay and Google AdWords attempt to design and use strategyproof auction mechanisms, even at the expense of potential short-term profit, because when users perceive that the system is manipulable, they have less trust in the system and are less likely to continue using it. Hence requiring that the auction based admission control mechanisms be strategyproof is an investment in the long-term success.

Besides users not being truthful about their valuations, another way users may manipulate the system is by submitting bogus queries. Specifically, a user may increase her payoff by submitting queries that she has no interest in. We call a mechanism that is not susceptible to this kind of manipulation *sybil immune*. Hence, toward establishing the DSMS center, our ultimate goal is to design a CQ admission control mechanism which is strategyproof and sybil immune. This led us to develop a number of admission control mechanisms with different properties based on sound principles that allow their formal validation as strategyproof and/or sybil immune. We have also experimentally identified potential tradeoffs in terms of system profit, client payoff and rate of CQ admission[10].

*This was supported in part by an IBM faculty award, and from NSF grants CNS-0325353, CCF-0514058, IIS-0534531, IIS-0746696 and CCF-0830558.

[‡]Authors are listed in alphabetical order. This work was done while the third author was a graduate student at Pitt.

Clearly the most important of them for our business is system profit and interestingly, the mechanism which is strategyproof and sybil immune offers the best tradeoff with respect to profit.

To summarize, our **contributions** are:

- We introduce the notion of *sybil immunity* for auction mechanisms.
- We propose a number of mechanisms for this problem (four natural, greedy mechanisms) and show that they are *strategyproof*, but only CAT (CQ Admission based on Total load) is also sybil immune.
- We experimentally show that CAT provides the best trade-off between system profit and total user payoff.

II. SYSTEM MODEL

In our model, the DSMS center has an admission control mechanism that supports a subscription period. During each subscription period, say a day, users submit queries q_i ($i = 1 \dots n$) along with a bid b_i . At the end of each subscription period, the mechanism evaluates the users' bids, and returns a decision about which CQs to admit and run the next day. It also returns the price p_i charged to each admitted q_i .

A bid b_i expresses a declared bound on how much a user is willing to pay to have query q_i executed. Further each user has a *private value* v_i expressing how much having query q_i run is really worth to her. The *payoff* (aka utility) u_i of the user that submitted query q_i is $v_i - p_i$ if q_i is accepted, and 0 otherwise.

The aggregate load of the operators in the accepted CQs can be at most the capacity of the server. We model the system capacity as the amount of work that can be executed in a time unit, given the system's resources (CPU, memory, etc.).

For our purposes, it is sufficient to view a CQ as a collection of operators ignoring their dependencies. We assume that each operator o_j has an associated load c_j that represents the fraction of the system's capacity that o_j will use, and this load can at least be reasonably approximated by the system [1], [3].

We consider two different definitions of the load of the query. The *total load* of a query is the sum of the loads of its operators. The *fair share load* of a query is the sum of the fair-share load of its operators, where the *fair-share load* of an operator is the load of the operator divided by the number of queries that share that operator. It is expected that many CQs may contain the same operator. Shared operator processing has already been proposed and utilized in the literature ([1], [11], [12]). Operator sharing is based on the premise that many CQs are monitoring a few hot streams, and many of the CQs are similar, but not identical.

III. PROPOSED MECHANISMS

In this section, we present several greedy CQ auction mechanisms. We show that all of these mechanisms, CAT, CAT+ (an extension to CAT), CAF (CQ Admission based on Fair Share), and CAF+ (an extension to CAF) are strategyproof.

Each of these mechanisms has the following form:

- Sort queries in order of decreasing profit density (bid per unit of required server load), and then
- admit queries until the server is full.

A. Clients Chosen by Total Load (CAT, CAT+)

In the following subsections, we propose two mechanisms using the above form based on the total load of a query: CAT and CAT+.

CAT (CQ Admission based on Total load).

Our first strategyproof mechanism that depends on the total load C_i^T is shown in Algorithm 1.

Algorithm 1 Our basic total load mechanism (CAT). **Input:** A set of queries each with its total load C_i^T and its corresponding bid b_i . **Output:** The set of queries to be serviced and their corresponding payments.

- 1) Set priority Pr_i to b_i/C_i^T for each query i .
 - 2) Sort and renumber queries in non-increasing Pr_i so that $Pr_1 \geq Pr_2 \geq \dots \geq Pr_n$.
 - 3) Add the maximal prefix of the queries in this ordered list that fits within server capacity to the winner list.
 - 4) Let $lost$ be the index of the first losing user in the above priority list.
 - 5) Charge each winner i a payment of $p_i = C_i^T (b_{lost}/C_{lost}^T)$. Charge all other users 0.
-

Selecting winners. Steps 1 through 3 of Algorithm 1 greedily select winners as follows. A priority is assigned to each operator, where the priority is the value-load ratio: $Pr_i = b_i/C_i^T$. Then the list of CQs is sorted in descending order of these priority values. The algorithm admits CQs from the priority list in this order until the remaining load of the next CQ q_j does not cause system capacity to be exceeded. The *remaining load* of query q_j is the total load of all the operators of q_j except those operators that are shared with CQs that have already been chosen as winners.

Calculating payments. Once the algorithm selects the winners, it calculates the payment for each winning user according to Steps 4 and 5 of Algorithm 1.

Strategyproofness. The CAT mechanism is strategyproof. Please refer to [10] for the proof.

CAT+: An Extension to CAT. CAT+ extends CAT by allowing the algorithm to continue until there are no unserved CQs left that will fit in the remaining server capacity.

Selecting winners. While CAT stops as soon as it encounters a query whose load exceeds remaining capacity, CAT+ skips over any queries that are too costly, continuing onto more light-weight queries down in the priority list.

Calculating payments. The algorithm calculates the payment of each selected query (or winning user) based on each user's *movement window*. Intuitively, the movement window of a winning user is the amount of freedom the user has to bid lower than her actual valuation without losing:

Definition 1: In CAT+, a user's *movement window* is defined as a sublist of the complete list of queries ordered in

descending priority $Pr_i = b_i/C_i^T$. We will refer to this list as the *priority list*. The movement window of winning user i begins with the user just after user i in the priority list, and ends at the first user j in the priority list that satisfies the following property: if user i 's bid was changed so that it directly followed the position of user j in the priority list, CAT+ would no longer choose query i as a winner. If such a user j does not exist, then user i 's movement window spans the entire remainder of the priority list.

Definition 2: For each winning query q_i , $last(i)$ is defined to be the first query which is outside q_i 's movement window. If there are no queries remaining outside the movement window of q_i , then $last(i)$ is set to *null*.

Since $last(i)$ is the lowest position user i can lower her bid to without losing, intuitively using $last(i)$ to set the payment of user i insures that she has no incentive to lie about her bid. Consequently, for each winner i , the algorithm first calculates the identity of $last(i)$. Then the payment for the selected query is defined as $p_i = C_i^T \cdot b_{last(i)}/C_{last(i)}^T$. If user i 's movement window included all remaining queries in the priority list, i.e., if $last(i) = null$, then the payment of user i is 0.

Strategyproofness. The proof that CAT+ is strategyproof is similar to that of CAT (see [10]).

B. Clients Chosen by Fair Share Load (CAF, CAF+)

We developed two more mechanisms that are exactly analogous to the mechanism from Section III-A, except that we replace every incidence of the static total load C_i^T with that fair share load C_i^{SF} :

- **CAF** (CQ Admission based on Fair share load): analogous to CAF described in Section III-A.
- **CAF+**: analogous to CAT+ described in Section III-A.

Strategyproofness. Under CAF and CAF+, a user might be tempted to lie about which operators are contained in her query because of the concept of fair share load. However, both CAF and CAF+ are *bid-strategyproof* (clients maximize their payoff when bidding their true valuations) and *strategyproof* (clients maximize their payoff when both bidding truthfully and submitting only the operators in the query actually desired by the user). Refer to [10] for the proofs of these properties.

IV. SYBIL ATTACK

Recall that a mechanism is *sybil immune* when no user can increase her payoff by submitting queries that she does not value. Both the CAF or CAF+ mechanisms are not sybil immune. A user i can create fake users with negligible valuations whose queries share operators with q_i . This will lower the attacker's fair share load, and enabling her to be selected as a winner while decreasing her payment.

In contrast to the fair share load, a user's total load is not dependent on the number of other users sharing her load, and therefore CAT is sybil immune. However, CAT+ is not because the arrival of additional queries might cause a loser to become a winner with positive payoff. Proofs for these properties can be found in [10].

V. EXPERIMENTAL EVALUATION

In this section, we demonstrate the behavior of our proposed auction-based admission control mechanisms using simulation. All of them were implemented in Java.

Metrics. For each mechanism, we measured the following performance metrics:

- **Profit:** the sum of the payments of the admitted queries.
- **Total user payoff:** the sum of the valuations (bids) of the admitted queries minus the payments. Total user payoff can be seen as an indication of total user satisfaction under each mechanism.
- **System utilization:** the used capacity of the server.

The reported results are the average of running each algorithm on 50 different sets of workload.

TABLE I
WORKLOAD CHARACTERISTICS

Number of workload sets	50
Number of queries	2000
Number of operators	700 ~ 8800
Max Degree of Sharing	[1 - 60] - Zipf, skewness: 1
Maximum Bid	100 - Zipf, skewness: 0.5
Maximum Operator Load	10 - Zipf, skewness: 1
System Capacity	15K

Workload. We summarize the workload parameters in Table I. We generated 50 sets of workload for four different system capacities. Each set contains a number of different input instances. An input instance consists of users' queries along with their bids, and is parameterized by:

- **System capacity.**
- **Maximum degree of sharing:** The degree of sharing of an operator is the number of queries that share a single operator, and the maximum is taken over all the operators.

We varied the maximum degree of sharing from 1 to 60. We keep the average query load the same throughout a workload set, while varying the maximum degree of sharing. Operators are assigned to queries randomly, where for each operator, the number of queries sharing it is drawn from a Zipf distribution.

Experimental Results. Figure 1(a) shows the system profit as the degree of sharing ranges from 1 to 60, for a system with capacity 15,000. CAF and CAT are the best for profit, as they do not admit queries as greedily as CAF+ and CAT+, do, which means the prices they charge the admitted queries are much higher than CAF+ and CAT+. The profit of CAF+ and CAT+ decrease as degree of sharing increases because they are simply admitting so many queries (as sharing increases) that the prices they are charging continue to be driven downward. Because the queries are selected in decreasing order of density and charged a per-unit price equal to the per-unit bid of the first losing query, very few queries lead to higher prices, more queries lead to lower prices.

At the second crossover point between CAF and CAT, CAF begins to admit such a high rate of queries that the prices it is charging are being driven dramatically downward (remember,

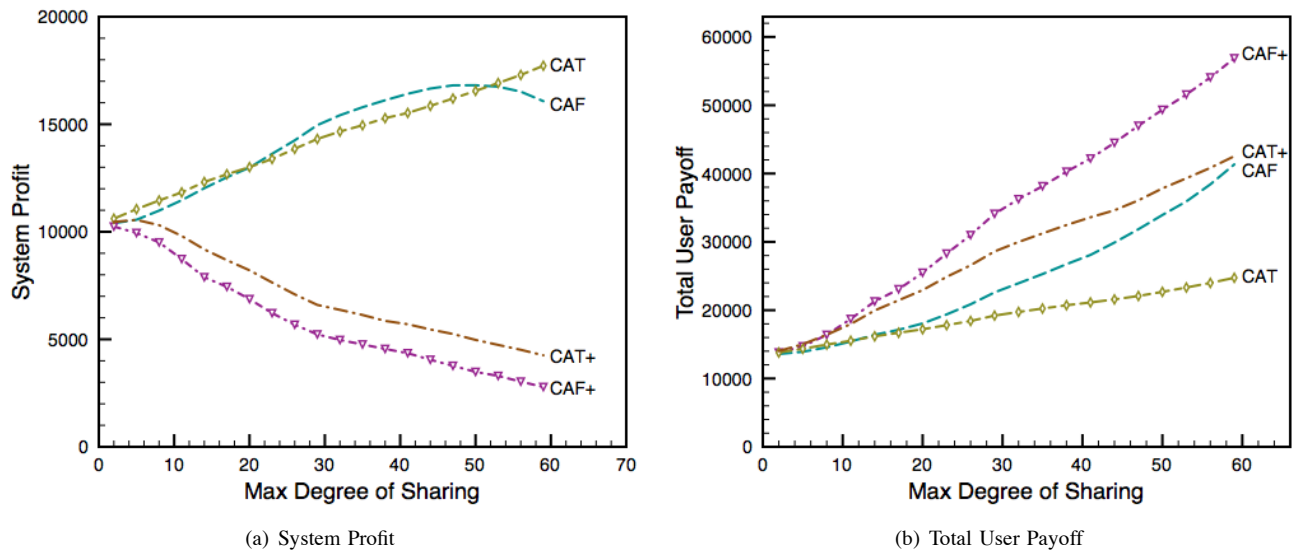


Fig. 1. Figure 1(a) shows the system profit for a system capacity of 15,000. Figure 1(b) shows total user payoff, which can be interpreted as a measure of total user-satisfaction. A user's payoff is defined as her valuation minus her payment. Seen here is the sum of winning users' payoffs.

query valuations are drawn from a skewed distribution), reducing overall profit faster than the gain in profit from admitting more queries. The profit of CAF begins to really dive, as the payments are an increasing function of each query's fair share load, which also shrinks as the degree of sharing increases.

With respect to maximizing total user payoff (Figure 1(b)), CAF+, of course, has the highest payoff because not only are the most queries admitted under CAF+, but users are only paying for their fair share load, rather than for their total load. As the degree of sharing increases, CAF begins to overtake CAT+ in total user payoff because fair share load per user is decreasing, which decreases payments, increasing payoffs. Each query's total load on the other hand, remains constant as the degree of sharing increases.

The tradeoff between the System Profit to the Total User Payoff can be quantified using their ratio. Table II shows this ratio for all the mechanisms at the degree of sharing of 40 where CAF provides the best system profit and at 60 where CAT provides the best system profit. Clearly, CAT provides the best tradeoff between profit and user payoff.

TABLE II
TRADEOFF BETWEEN SYSTEM PROFIT AND TOTAL USER PAYOFF

Degree of sharing	CAT	CAT+	CAF	CAF+
60	0.85	0.125	0.41	0.05
40	0.75	0.176	0.57	0.11

In terms of utilization, we found that all proposed mechanisms admit queries so as to utilize more than 98 percent of the system's capacity.

VI. CONCLUSION

In this paper, we are able to explore a novel way of describing user preferences in the CQ admission control problem by using an auction model. Although, most data stream admission

control (load shedding) algorithms work at the tuple level, we believe that focusing on the query level, as we do in this work, is equally important.

We provided a model for the problem that allows us to establish its difficulty and complexity. We introduced the notion of *sybil immunity* for auction mechanisms and designed greedy and randomized auction mechanisms for this problem which are all *strategyproof*. We conducted experiments to evaluate the performance of these mechanisms for metrics such as profit, admission rate, and total user payoff, and we showed that one of the mechanisms namely CAT is sybil immune and offers the best tradeoff with respect to profit.

REFERENCES

- [1] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management," *VLDBJ*, vol. 12, no. 2, pp. 120–139, 2003.
- [2] T. S. Group, "Stream: The stanford stream data manager," *IEEE Data Engineering Bulletin*, 2003.
- [3] M. A. Sharaf, P. K. Chrysanthos, A. Labrinidis, and K. Pruhs, "Algorithms and metrics for processing multiple heterogeneous continuous queries," *ACM Trans. Database Syst.*, vol. 33, no. 1, pp. 1–44, 2008.
- [4] "Streambase," 2006. [Online]. Available: <http://www.streambase.com>
- [5] System S. [Online]. Available: http://domino.research.ibm.com/comm/research_projects.nsf/pages/esps.index.html
- [6] Coral8. [Online]. Available: <http://www.coral8.com/>
- [7] S. Reiss, "Cloud computing. available at amazon.com today," *Wired*, April 2008.
- [8] S. Baker, "Google and the wisdom of clouds," *Business Week*, Dec. 2007.
- [9] P. McDougall, "Google, ibm join forces to dominate 'cloud computing'," *Information Week*, May 2009.
- [10] C. Chung, "Evolutionary solutions and internet applications for algorithmic game theory," Ph.D. dissertation, U. Pittsburgh, Aug. 2009.
- [11] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman, "Continuously adaptive continuous queries over streams," in *SIGMOD 2002*, pp. 49–60.
- [12] S. Krishnamurthy, C. Wu, and M. Franklin, "On-the-fly sharing for streamed aggregation," in *SIGMOD 2006*, pp. 623–634.