Admission Control Mechanisms for Continuous Queries in the Cloud

Lory Al Moakar *, Panos K. Chrysanthis, Christine Chung, Shenoda Guirguis,
Alexandros Labrinidis, Panayiotis Neophytou, Kirk Pruhs

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

*Authors are listed in alphabetical order.

***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 will commit to paying 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 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 further consider the requirement that the mechanism be sybil-immune, that is, that no user can increase her payoff by submitting queries that she does not value. The main combinatorial challenges come from the difficulty of effectively taking advantage of the shared processing between queries. 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, total user payoff, and rate of admission, showing what tradeoffs may be in store for implementers.**

I. Introduction

The growing need for *monitoring applications* such as the real-time detection of disease outbreaks, tracking the stock market, environmental monitoring via 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. Those who do not get serviced are not denied arbitrarily due to the system's limited resources, but instead feel legitimately excluded because their bid was simply not high enough. 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

TABLE I

PROPERTIES OF OUR PROPOSED AUCTION MECHANISMS

	CAF	CAF+	CAT	CAT+	Two-price
Strategyproof	✓	✓	✓	✓	✓
Sybil Immune	×	×	✓	×	×
Profit Guarantee	×	×	×	×	✓

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. 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 apply techniques and principles from algorithmic game theory to a data streams query admission control problem.
- We introduce the notion of *sybil immunity* for auction mechanisms.
- We propose a number of mechanisms for this problem (four natural, greedy mechanisms and one randomized mechanism) and show that they are *strategyproof*, but only one, called CAT, is also sybil immune. These results are summarized in Table I.
- We experimentally show that greedy mechanisms (which take into account both the bid and the load for each user's query), provide both increased system profits as well as better total user payoff, compared to a randomized algorithm that has a profit guarantee. In particular, CAT provides the best tradeoff with respect to profit.

Road map. We define the system model in Section II and provide a summary of the relevant background in Section III. We present our mechanisms in Section IV, analyze their sybil immunity in Section V, and present their empirical evaluation in Section VI. We discuss extensions of our problem in Section VII and conclude in Section VIII.

II. SYSTEM MODEL

In our model, the DSMS center has an admission control mechanism that, at the end of each subscription period, say a day, accepts the users' bids and relevant information about their CQs, and returns a decision about which CQs to admit and run the next day[1]. The mechanism also returns the price p_i charged to each CQ_i that is admitted. 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.).

[1]Of course, the time span between each auction could just as easily be one week or one month. In Section VII, we discuss how our results can extend to a more general setting where each query may subscribe for a different time span.

We assume an underlying query model similar to the Aurora model [1] where subnetworks are connected via connection points. During the transition phase at the end of each day, the upstream connection points that surround the subnetworks that need to be modified hold any incoming data tuples. The tuples stored inside the queues of these subnetworks are drained through the downstream connection points. Then, the query planner modifies the subnetwork by adding new operators or deleting operators. Once the query planner finishes, the tuples stored at the connection points are input into the subnetwork before the newly arriving tuples are executed. This transition phase ensures the correctness of the results output by CQs that continue to execute for the next day.

For our purposes, it is sufficient to view a CQ as a collection of operators ignoring their dependencies (Figure 2). For example, a CQ might consist of three operators:

- A select operator on a stream of stock quotes that selects out high value transactions,
- A select operator on a stream of news stories that selects stories that mention companies with publicly traded stock
- A join operator that joins the results of the two selection operators on the company name attribute.

We assume that each operator o_j has an associated load c_j that represents the fraction of the system's capacity that this operator will use, and this load can at least be reasonably approximated by the system [1], [3]. It is expected that many CQs may contain the same operator. Shared operator processing has already been proposed and utilized in the literature ([1], [10], [11]). 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. For example, one could imagine many queries want to select news stories on publicly traded companies. So in a stock monitoring application, many aggregate CQs will be defined on few indexes, with similar aggregate functions, but different joins and different windows. Thus, sharing can be expected to be heavy.

We assume that each of n users submits a query q_i along with a bid b_i. The bid expresses a declared bound on how much a user is willing to pay to have the query executed. Further each user has a private value v_i expressing how much having query q_i run is really worth to her. It will be useful to define a variable h to represent the largest valuation of any user. 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.

Example 1: To make these concepts concrete, consider three queries (q_1, q_2 and q_3) which are submitted to a DSMS with a capacity of 10 units. Figure 1 shows their query plan. Note that q_1 and q_2 share operator A. As mentioned above, it is sufficient to abstract away the dependencies between operators of a CQ and retain only the information seen in Figure 2: the set of operators that comprise all the queries, the load of each operator, an indication of which queries each operator belongs to, and the user bids.

From the business' point of view, the most obvious design goal for the mechanism is to maximize profit, which is the

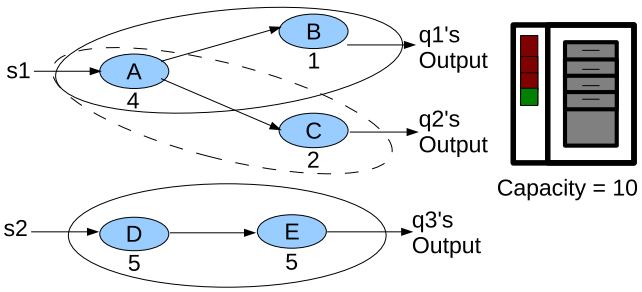


Fig. 1. Queries in Example 1 as seen by the DSMS. Each query has two operators: A and B in q_1, A and C in q_2, D and E in q_3. Operator A is shared between q_1 and q_2. Each operator is also labeled with the load associated with it. The input data streams to operators A and D are $s1$ and $s2$ respectively.

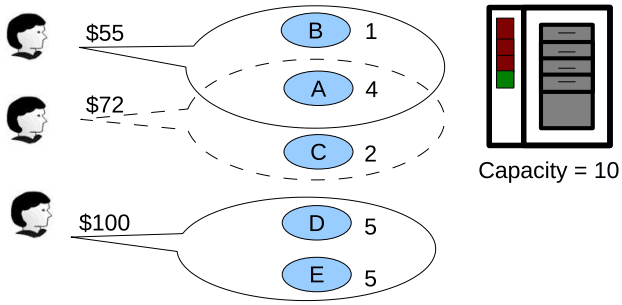


Fig. 2. Simplified query plan of queries in Figure 1 to show only the operators, omitting the order of the operators information. This simplified, abstract information model is the one we use for our problem. The user bids are shown above as dollar amounts.

aggregate prices charged to the accepted queries. Another first class design goal for the mechanism is *strategyproofness*. Intuitively, strategyproofness means that users have no incentive to "lie" about their private values, or bid strategically.

III. Relevant Background

A mechanism where users always maximize their payoff by truthfully revealing *all* their private information is called *strategyproof*. In many auction settings, clients' true valuations are the only private information, and hence in such settings *strategyproofness* means that clients maximize their payoff when bidding their true valuations. In this paper, we will refer to this property as *bid-strategyproofness*, as we also consider other private information: a user might conceivably lie about which operators are contained in her query, say by adding additional operators that are not part of the query she actually desires. In the context of our CQ admission auction then, we call a mechanism is *strategyproof* when both bidding truthfully and submitting only the operators in the query actually desired by the user maximizes the user's payoff. In this work, all the mechanisms we propose are not only bid-strategyproof, but also strategyproof. Several standard auction problems are special cases of our auction problem for CQs.

Settings without Sharing. In the special case that there are no shared operators, the load of each query (which is the aggregate load of the query's operators) is the same, and there is room for k queries, then this is equivalent to the problem of auctioning k identical goods. Charging each of the k highest bidders the $(k+1)$st highest bid is well known to be bid-strategyproof. When $k = 1$, this is famously known as "Vickrey's second price" auction.

If CQs do not share operators, but the load of each query may be different, then the resulting problem is what is known in the literature as a Knapsack Auction problem, studied by Aggarwal and Hartline in [12].

Operator Sharing. Operators shared between queries greatly complicate the task of the mechanism because the *profit density* of a query, which refers to the ratio of the bid for that query (or potential profit to be obtained from accepting the query) to the load of the query, depends on which other operators are selected. For example, consider a query q_i with low value and high load. In overload situations, query q_i would surely be rejected in a knapsack auction. But if all of q_i's operators were shared by high value queries, then the effective profit density of q_i (given that we know these high value queries were accepted) could be very high. This dependency between queries makes the mechanism's task much more complex in the case of our CQ auction than in the case of a knapsack auction. This complexity is illustrated by the fact that there is a polynomial time approximation scheme for finding the maximum value collection of items to select in a knapsack auction, but even for a special case of our CQ auction, the densest subgraph problem, it is not known how to approximate the optimal solution to within a polynomial factor in polynomial time [13].

Characterizations of Strategyproofness. A CQ auction where the only private information is the amount each user values her query is called a single-parameter setting. In single-parameter settings, an allocation mechanism is called *monotone* if every winning bidder remains a winning bidder when increasing her bid. The *critical value* of user i is the value c_i where if the user bids higher than c_i, she wins, but if she bids lower than c_i, she loses. Note that the existence of a critical value for each user is guaranteed by the preceding monotonicity property. It is shown in [14] that a mechanism is bid-strategyproof if and only if it is both *monotone* and each winning user's payment is equal to her *critical value*.

One final auction setting related to our CQ auction is the *single minded bidders (SMB) auction problem*, studied by Lehmann et al [15]. Each single-minded bidder i is interested in a specific collection S_i from the set of items being auctioned (for a CQ auction the items being auctioned would be server capacity units and S_i would be the units needed to process the collection of operators in the query q_i). In addition to bid-strategyproofness. [16], [15] provide a characterization for strategyproofness in this setting that applies to our setting. Their characterization of a strategyproof mechanism for an SMB auction differs only slightly from the above characterization for bid-strategyproofness: the definition of monotonicity is expanded. In terms of our CQ admission auction, monotonicity

means that not only must a winning bidder remain a winning bidder when increasing her bid, but also must remain a winner when submitting a query comprised of a strict subset of the operators in the admitted query.

IV. Proposed Mechanisms

In this section, we present several greedy CQ auction mechanisms. We show that four of these mechanisms, CAF, CAF+, CAT, and CAT+, 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.

The intuition is that we wish to accept queries with high valuation to load ratio.

We consider two different definitions of the load of the query. CAT assumes that the load of a query is the sum of the loads of its operators. CAF assumes that the 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. So intuitively, CAT operates as though there will be minimal or no operator sharing among the accepted queries, while CAF operates as though there will be maximal operator sharing among the accepted queries. CAT and CAF stop once the first query is encountered that will not fit, while the mechanisms CAT+ and CAF+ continue processing the queries hoping to find later, lower load, queries that will fit. In each of these mechanisms, the price per-unit-load quoted for query q_i is the profit density of a particular rejected query. For each of CAT and CAF the price for each accepted query is based on the profit density of the first rejected query. Hence the mechanisms for CAT and CAF are offering a fixed price per unit of server capacity that a query uses. The price for a query q_i in CAF+ and CAT+ is based on the minimum profit density after which q_i would no longer have been accepted. We show that these mechanisms are all strategyproof. Next we present the mechanisms in detail.

A. Clients Chosen by Remaining Load (CAR)

In order to set the stage, we start by describing a naïve approach that uses the remaining additional load needed to service a CQ for choosing winners and determining payment values. We show how this approach, while it accurately captures the additional load each query will contribute to the total load on the server, is not bid-strategyproof.

Consider the following natural mechanism using the aforementioned greedy scheme for choosing winners. The mechanism first chooses each winner based on a value we define called a query's "remaining load." Then the mechanism charges each winner a payment that also depends on that user's remaining load. We refer to the mechanism as the CAR mechanism (CQ Admission based on Remaining load) We will show that using such a payment scheme is not bid-strategyproof, due to the fact that user's payments are dependent on their bids.

Selecting Winners. We sort the CQs in non-increasing order of priority Pr_i, where $Pr_i = b_i/C_i^R$ and C_i^R is defined as:

Definition 2: (Remaining Load C_i^R) The *remaining load* C_i^R of query i is equal to the total load of all the operators of q_i except those operators that are shared with CQs that have already been chosen as winners.

In every iteration through the loop, the algorithm chooses the query with the highest priority and if there is enough remaining capacity in the system to accommodate it, places it in the set of winners. At the end of each iteration, the remaining loads C_i^R as well as the priorities of the yet-unchosen queries are updated. We demonstrate this mechanism with the example in Figure 2.

Calculating Payments. We naturally base our first payment mechanism on the known bid-strategyproof k-unit $(k+1)$th-price auction. Recall from Section III that a simple strategyproof mechanism for a k-unit auction is to charge each winning bidder the bid amount of the $(k+1)$th highest bidder. Hence, we define q_{lost} to be the CQ with highest priority that is not a winner. Then, the payment of each winning CQ q_i is calculated as follows: $p_i = C_i^R \cdot b_{lost}/C_{lost}^R$. If the query does not belong to the *winners* list, then the payment is zero.

Remaining Load Algorithm Applied to Example 1. The initial remaining loads of q_1 , q_2 and q_3 are 5, 6, and 10 respectively. The priorities of q_1, q_2 and q_3 become 11, 12 and 10. During the first iteration of the above algorithm, q_2 is chosen first. Since operator A is chosen as part of q_2, the remaining load of q_1 becomes the load of operator B (just 1 unit) and its priority becomes 55. Consequently, during the second iteration q_1 is chosen. The remaining capacity in the system is 3. During the third iteration, q_3 is chosen however it does not fit in the remaining capacity in the system. As a result, the winners list is composed of q_1 and q_2, and q_{lost} is q_3. As a result, the payments for q_1 and q_2 is \$10 per unit load, which amount to respective payments of \$10 and \$60.

Strategyproofness. The above payment mechanism at first glance seems bid-strategyproof since it is based closely on the well-known bid-strategyproof second-price auction mechanism. However, it is not, since a winning user i who shares operators with other winning users can gain by bidding lower than her true value. She can strategically bid low enough so that she gets chosen for service *after* the users she shares operators with, but still high enough to win. This will result in a lower remaining load C_i^R and thus in a lower payment.

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

At this point it has become clear that using remaining load $\left(C_i^R\right)$ for setting payments of users is problematic because of the dependence of these values on the user's bid. Therefore, we consider using a fixed load that does not change over the course of the winner selection algorithm, and we use that same fixed load to calculate payments.

We define the *static fair share load* as follows.

Definition 3: Let o_j be an operator that has a load of c_j and is shared among l different CQs, then the static fair share

load of o_j per CQ is defined as $c_j^{SF} = c_j/l$. Hence, the *static fair share load* of a CQ q_i is defined as $C_i^{SF} = \sum_{o_j \in Q_i} c_j^{SF}$.

In the following subsections we propose two bid-strategyproof payment mechanisms using the same greedy scheme, but based on static fair share load: CAF and CAF+.

CAF (CQ Admission based on Fair share). Our first bid-strategyproof mechanism that depends on the static fair share load as defined in Definition 3 is shown in Algorithm 1.

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: $Pr_i = b_i/C_i^{SF}$. 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 as long as the remaining load C_i^R of hosting the next CQ does not cause system capacity to be exceeded. (Note that the load considered while checking capacity constraints is not the static fair share load.) The algorithm stops as soon as the next CQ does not fit within server capacity.

Algorithm 1 Our basic fair share mechanism (CAF). **Input:** A set of queries with their static fair share loads C_i^{SF} and their corresponding bids b_i. **Output:** The set of queries to be serviced and their corresponding payments.

1) Set priority Pr_i to b_i/C_i^{SF} for each query i.
2) Sort and renumber queries in non-increasing Pr_i so that $Pr_1 \geq Pr_2 \geq \ldots \geq Pr_n$.
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 $p_i = C_i^{SF}(b_{lost}/C_{lost}^{SF})$. Charge all other users 0.

Calculating payments. Once we have selected the winners, we calculate the payment for each winning user according to steps 4 and 5 of Algorithm 1.

CAF Applied to Example 1. Since q_1 shares operator A with q_2, C_1^{SF} is 3 and C_2^{SF} is 4. During the first iteration of CAF, the priorities of q_1, q_2 and q_3 are 18.34, 18, and 10. As a result, CAF chooses q_1 first and then q_2. Again, q_3 is q_{lost}. Thus the payments for q_1 and q_2 are \$10 per unit load, which amount to respective payments of \$30 and \$40.

Strategyproofness. We prove the following theorem by using the characterization of bid-strategyproof mechanisms for any single-parameter setting (Section III).

Theorem 4: The CAF mechanism is bid-strategyproof.

Proof: The CAF winner selection is clearly monotone: any winning bidder could not become a loser by increasing her bid since she will only move up in the priority list by doing so. The CAF payments are also equal to the users' critical values. If user i bids $b_i' < C_i^{SF}(b_{lost}/C_{lost}^{SF})$, then we would have $b_i'/C_i^{SF} < b_{lost}/C_{lost}^{SF}$ and we know that both user i and user $lost$ could not fit together on the server with the other winners, so user i will become a loser. ■

We also note that CAF is not just bid-strategyproof, but strategyproof. This results from the fact that the characterization for SMB auctions in [15] carries over to our setting (see Section III), and that CAF satisfies their additional monotonicity requirement that when a winning bidder asks for only a subset of the operators in her query, she still wins.

CAF+: An Extension to CAF.

Selecting winners. CAF+ extends CAF by allowing the algorithm to continue until there are no unserviced CQs left that will fit in the remaining server capacity. While CAF stops as soon as it encounters a query whose load exceeds remaining capacity, CAF+ skips over any queries that are too costly, continuing onto more light-weight queries in the priority list. (See Algorithm 2.)

Calculating payments. The algorithm calculates the payment of each winning user (or serviced query) 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. A more formal definition follows.

Definition 5: In CAF+ every query that is selected to be serviced has a *movement window*. A user's movement window is defined as a sublist of the complete list of queries ordered in descending priority $Pr_i = b_i/C_i^{SF}$. 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 both comes after i and 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, CAF+ 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 6: 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 $null$.

The payment in CAF+ (Algorithm 2) is calculated for each query after the set of queries to be serviced is determined. For each winner i, the algorithm first calculates the identity of $last(i)$. Then the payment for the selected query is defined as $p_i = C_i^{SF} \cdot b_{last(i)}/C_{last(i)}^{SF}$. 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 CAF+ is bid-strategyproof is similar to that of Theorem 4; we again use the characterization of bid-strategyproofness from [14].

Theorem 7: The CAF+ mechanism is bid-strategyproof.

As with CAF, we note that CAF+ is not only bid-strategyproof, but strategyproof. The reasoning is the same as for CAF (see Section IV-B).

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

Because the "fairshare" based mechanisms described above are vulnerable to certain types of user manipulation (see Section V), we design two more robust mechanisms. These

Algorithm 2 Our aggressive fairshare mechanism (CAF+). **Input:** A set of queries with their static fair share loads C_i^{SF} and their corresponding bids b_i. **Output:** The set of queries to be serviced and their corresponding payments.

1) Set priority Pr_i to b_i/C_i^{SF} for each query i.
2) Sort and renumber queries in non-increasing Pr_i so that $Pr_1 \geq Pr_2 \geq \ldots \geq Pr_n$.
3) For $i = 1 \ldots n$, add user i to the winner list if doing so does not exceed capacity.
4) For each winner i, calculate $last(i)$, as defined in Definition 6.
5) Charge each winner i a payment of $p_i = C_i^{SF}(b_{last(i)}/C_{last(i)}^{FS})$. Charge all other users 0.

mechanism are exactly analogous to the mechanism from Section IV-B, except that we replace every incidence of the static fairshare load C_i^{SF} with that total load $C_i^T = \sum_{o_j \in Q_i} c_j$. Thus we have two mechanisms.

- **CAT** (CQ Admission based on Total load): analogous to CAF described in Section IV-B.
- **CAT+**: analogous to CAF+ described in Section IV-B.

CAT Applied to Example 1. In example 1 C_1^T, C_2^T and C_3^T are 5, 6 and 10 units. Thus Pr_1, Pr_2 and Pr_3 are 11, 12, and 10. Consequently, CAT chooses q_1 and q_2 to be serviced. The payments for q_1 and q_2 are \$10 per unit load, which amount to respective payments of \$50 and \$60.

It is easy to verify that the proofs of bid-strategyproofness carry over to these modified versions of the algorithms and payments. We therefore have the following two theorems.

Theorem 8: The CAT mechanism is bid-strategyproof.

Theorem 9: The CAT+ mechanism is bid-strategyproof.

As with CAF and CAF+, we note that both CAT and CAT+ are not only bid-strategyproof, but strategyproof.

D. A Profit Guarantee

While we will experimentally show that the above greedy mechanisms perform quite well for profit maximization (Section VI), they do not admit *provable* profit guarantees that are reasonable (due to some special, pathological input instances). We thus turn to a basic mechanism that is based only on user bids rather than density: the CQs are simply sorted in decreasing bid order, and then selected from the top until the next CQ does not fit in system capacity. The chosen CQs then pay a price equal to the bid of the first losing CQ. We refer to this basic solution as the Greedy-by-Valuation (GV) mechanism. While GV also does not admit a profit guarantee, we propose a strategyproof randomized mechanism based on GV called Two-Price, that has a provable profit guarantee. Specifically, it is competitive (in expectation) with the best optimal constant pricing mechanism. A *constant pricing* mechanism as defined in [12], is any mechanism (strategyproof or not) where the users are all charged the same price, call it p, and those who bid strictly higher than p are winners, those who bid strictly lower than p are losers, and those who bid equal to p may be designated winners or losers arbitrarily by the mechanism. Winners must all pay p and losers pay 0. *Profit* is defined to be the sum of the payments that the mechanism charges or receives from the users.

A *constant pricing* mechanism is *valid* if all winners fit within server capacity, and so we will only consider valid constant prices. *Optimal constant pricing profit* (OPT_C) then refers to the maximum possible profit that can be attained from any valid constant pricing mechanism (strategyproof or not). We choose to focus on constant pricing optimality in this paper because with the shared processing of queries in our problem, other standard profit benchmarks seem difficult to compete with. Two other natural profit benchmarks include optimal pricing per unit load and optimal monotone pricing, both of which generalize optimal constant pricing and were discussed in the context of Knapsack Auctions in [12]. But because of our shared processing between queries, the processing load required of each query is not clear cut. Hence both proportional and monotone pricing definitions become fuzzy.

The Randomized Mechanism. We now show that by only using two distinct prices, under the assumption that the users all have distinct valuations, we are able to find a bid-strategyproof mechanism that approximates optimal constant pricing profit. We show however that there is a trade-off between the run-time of the mechanism and its profit. We first present a mechanism that runs in time exponential in the number of duplicate valuations, then explain how a polynomial time version of it gives a weaker profit guarantee.

We refer to our mechanism as the *Two-price Mechanism*. The first phase of the mechanism (Steps 1 and 2) follows our greedy scheme (using user valuations), the second phase (Step 3) is an exhaustive search that gives the potential exponential running time in terms of number of duplicate valuations, and the last phase (Steps 4 through 6) contains the randomization and is essentially identical to the Random Sampling Optimal Price auction of [17].

Note that in Step 3 of the mechanism we run an exhaustive search on all possible subsets of the critical set of queries with duplicate valuations. The possibility of sharing of server capacity between queries is what requires us to take this potentially arduous step, as the problem of optimally determining which subset of queries to admit in the face of such sharing seems hard to approximate. For the proof of the following theorem, and any remaining theorems, please see [18].

Theorem 10: The *Two-price* mechanism is bid-strategyproof.

Note that because the Two-price mechanism allocates winners and sets payments entirely independent of each query's load, it is not only bid-strategyproof, but strategyproof. We now state the competitiveness of Two-price for profit maximization. We assume user valuations range from 1 to h and we use OPT_C to refer to the optimal constant pricing profit.

Theorem 11: The expected profit of the *Two-price* mechanism is at least $OPT - 2h$.

The next theorem applies to the polynomial-time mechanism that results when Step 3 of Algorithm 3 is omitted.

Theorem 12: The expected profit of the polynomial-time mechanism defined by the *Two-price* mechanism without Step

Algorithm 3 *Two-price mechanism*. **Input:** Set of n queries and corresponding user valuations $v_1 \dots v_n$. **Output:** Set of winners and their corresponding payments.

1) Sort and renumber the queries in order of decreasing valuation, so $v_1 \geq v_2 \geq v_3 \geq \ldots \geq v_n$, breaking ties arbitrarily.
2) Let H be the ordered set of queries that comprise the maximal prefix of queries from this sorted list that fits within our server capacity. Let L be the ordered set of losers (remaining queries not chosen for H) and let v_L be the valuation corresponding to the first query in L.
3) If the last query in H has valuation v_L, the set of queries in H must be adjusted as follows. Let D be the set of all users with valuation equal to v_L, and let d be the cardinality of D. Let $H' = H - D$. Let D^* be the largest subset of D that fits within capacity along with H'. Redefine $H = H' + D^*$
4) Partition the users from H evenly into two sets, A and B, uniformly at random. Renumber queries separately in each set as in step 1. I.e., $v_1 \geq v_2 \geq \ldots \geq v_a$ for the a queries in set A, and $v_1 \geq v_2 \geq \ldots \geq v_b$ for the b queries in set B, again breaking ties arbitrarily.
5) Calculate the optimal constant price profit of each set of queries: $OPT(A) = \max_{i \in A} iv_i$ and $OPT(B) = \max_{i \in B} iv_i$. Let $k = \arg\max_{i \in A} iv_i$ and let $p_A = v_k$. Similarly, let $j = \arg\max_{i \in B} iv_i$ and let $p_B = v_j$.
6) Use the price p_A to determine the winners from set B and use the price p_B to determine the winners from set A. Specifically, the winners from set B are those users whose valuations are greater than p_A, and these winners are each charged a payment of p_A. Similarly determine winners and payments for users in set A.

3 is at least $OPT - dh$.

In this section we presented mechanisms that are strategyproof. Next, we investigate their sybil immunity.

V. Sybil Attack

In this section we consider a strategic behavior that is well-known in the context of reputation systems like that of eBay and Amazon for rating sellers, buyers and products: a sybil attack. A user who behaves strategically using a sybil attack forges multiple ("fake") identities to manipulate the system. In reputation systems a user might try to boost the reputation of some entity by perhaps adding positive recommendations from false users [19]. In our setting, a sybil attack amounts to creating false identities to submit additional queries that the user does not need or value in order to manipulate the mechanism. (See Figure 3.) Thus we define a mechanism to be *sybil immune* if a user can never increase her payoff by submitting additional fake, no-value queries. We make the natural assumption that if a fake query is chosen to be serviced, the sybil attacker is responsible for making the fake query's payments, so a user's payoff is the aggregate payoff that she gains from the queries of all of her identities. We will show here that CAT is sybil immune, while the rest of the mechanisms are not. To the best of our knowledge, this is the first time that sybil immunity has been proposed, and we note that the notion of sybil immunity can apply to any mechanism design problem.

Definition 13: We define a mechanism to be *vulnerable* to sybil attack if there exists an input instance where there is a user who can increase her payoff by perpetrating a sybil attack.

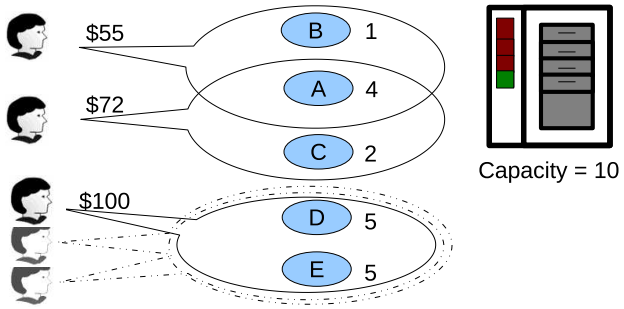


Fig. 3. This figure illustrates the third user from Example 1 perpetrating a sybil attack by forging two additional fake users. The real queries are indicated in solid lines while the fake queries are indicated in dashed lines. The presence of these fake queries creates the illusion that user 3's operators are in higher demand, which could conceivably influence the mechanism to either charge the third user less, or service her when she would not have otherwise been serviced.

Definition 14: We define a mechanism to be *universally vulnerable* to sybil attack if in every input instance, every user has a way to improve her payoff by perpetrating a sybil attack.

A. Attacks Against the Fair Share Mechanisms

Unfortunately, our proposed fair-share schemes of Section IV-B are vulnerable to sybil attack. A user i can employ the following strategy using a sybil attack to improve her payoff: simply create fake users with negligible valuations whose queries share operators with q_i. A sybil attack of this kind will lower the attacker's fair share load, improving her ranking and enabling her to be selected as a winner while simultaneously decreasing her payment to an affordable amount. Note that it is always possible for the attacker to set her fake users' valuations low enough so that they are not in danger of being selected as winners, and hence will require no additional payment from the attacker.

Indeed, we can prove that in *any* given instance of the CQ admission problem, *any* user can gain from employing a sybil attack against our fair share mechanism.

Theorem 15: Both the CAF or CAF+ mechanisms are *universally vulnerable* to sybil attack.

B. Attacks Against the Total Load Mechanisms

In contrast to this vulnerability of our fair share mechanisms, the total load payment mechanisms (CAT and CAT+), described in Section IV-C, seem at first to be robust to such attacks. While we've seen that a user's fair share can easily be reduced by creating fake identities, a user's total load is not dependent on the number of other users sharing her load, and therefore CAT and CAT+ should not (at least at first glance) be prone to such sybil attack strategies.

Definition 16: We say that a mechanism is *immune to sybil attack* if for every input instance, no user can increase her

payoff by perpetrating a sybil attack (i.e., it is not vulnerable). We also use the term *sybil immunity* to refer to this property.

However, one of our total load mechanisms is not immune to sybil attack. We begin by giving the following characterization of sybil immunity. A mechanism is sybil immune if and only if both of the following properties hold:

1) The arrival of additional queries will never cause a loser to become a winner with positive payoff.
2) If the arrival of additional queries reduces a winner's payment by δ, the additional queries that become winners must be charged a total of at least δ by the mechanism.

We now show that CAT+ is vulnerable to sybil attack because it does not satisfy the above property 1.

Theorem 17: For the CQ admission problem, CAT+ is vulnerable to sybil attack.

TABLE II
AN EXAMPLE OF A SYBIL ATTACK THAT BEATS CAT+.

User	1	2	"3"
v_i	100	89	$100\epsilon + \epsilon$
C_i^T	1	0.9	ϵ
Pr_i	100	< 100	> 100
Round 1	100	< 100	picked
Round 2	exceeds cap.	picked	picked
Payments p_i	0	0	$\leftarrow 100\epsilon$
Payoffs	0	$89 - 100\epsilon$	N/A

To see why, consider the example in Table II, in which a sybil attacker defeats our total load algorithm, CAT+. User 2 is a sybil attacker, creating a fake query that appears to the system as "user 3". Here, ϵ represents an arbitrarily small positive value. In this example, if user 2 does not perpetrate the attack, user 1 will get chosen for service, and then server capacity will be reached, so user 2 would not get serviced. Whereas when user 2 introduces the fake "user 3," she is able to trick the system into choosing her instead of user 1. While user 2 is responsible for the fake user's payment, user 2 carefully created "user 3" so that its payment would be a negligible amount. Note that user 2's payment for query 2 is 0 since there is no one left after she is chosen.

Note that in this kind of sybil attack, the danger for user 2 (the attacker) is that when the fake "user 3" was chosen for service, user 2 had to make user 3's payment. Hence user 3's fake valuation and fake load had to be carefully chosen by user 2 so that user 2 found paying user 3's fee worthwhile. (Recall from Section IV-C that payment of a winning user i is calculated as $C_i^T v_{lost}/C_{lost}^T$, so in our example, that makes $p_3 = 100\epsilon$). In this particular instance, user 2 had no payment of her own to pay because there are no users that have lower priority than user 2. This makes paying "user 3"'s payment affordable to user 2.

The good news is: our total load mechanisms are not always bad. First, while our fair share mechanisms are *universally vulnerable* to attack, there are instances under the total share mechanism that are robust to sybil attack. Second, and more notably, the CAT mechanism is immune to sybil attack. In fact we can make an even stronger claim. Thus far in our discussion of sybil attacks, we have been considering sybil attack in isolation from bid-strategyproofness. However, it is possible that a user can use a sybil attack *in conjunction* with lying about her valuation in order to increase her payoff. This possibility raises the question of whether adding sybil attacks to each user's set of possible strategies has removed our mechanism's bid-strategyproofness.

It turns out that our CAT mechanism remains bid-strategyproof even if we allow sybil attacks, *and* it remains immune to sybil attack, even if we allow users to lie about their valuations.

Definition 18: We define a mechanism to be *sybil-strategyproof* if no user can improve her payoff by either lying about her valuation, perpetrating a sybil attack, or doing both simultaneously.

We now give a characterization of sybil-strategyproof mechanisms. A mechanism is sybil-strategyproof if and only if both of the following properties hold:

1) It is bid-strategyproof.
2) The arrival of additional users (e.g., via a sybil attack) cannot decrease anyone's critical value by an amount more than the total payment charged to the additional users.

The above characterization is used to prove that CAT is sybil-strategyproof.

Theorem 19: For the CQ admission problem, the CAT mechanism is sybil-strategyproof.

C. Attacks Against the Randomized Mechanism

Our randomized *Two-price* mechanism, however, is not immune to sybil attack. This fact is proven by showing that the mechanism violates property 2 of our characterization of sybil immunity: a winning user can reduce her payment (in expectation) by introducing fake queries such that the fake queries incur less expected total charges than the amount her payment was reduced by.

Theorem 20: The *Two-price* mechanism is vulnerable to sybil attack.

Finally, we note that even if we modify Step 4 of the mechanism so that each query is placed in set A or B based on independent coin flips (so that H may not be evenly partitioned), the mechanism is still vulnerable to sybil attack. Again, the vulnerability is due to a violation of property 2 of our characterization of sybil immunity. Consider the instance where user 1 has valuation b and n_c users (which get placed into H along with user 1) all have a valuation of $c < b$. Set sizes for the users in H so that server capacity is exactly filled.

User 1 creates a fake user with valuation $d = c+\epsilon$, with size equal to the combined size of all the users with valuation c, kicking them out of H. While before user 1 was charged c with probability $1 - (1/2)^{n_c}$ and 0 with probability $(1/2)^{n_c}$, now that only user 1 and the fake user are in H, user 1 and her fake user is charged 0 with probability 1/2, and d with probability 1/2. For choice of ϵ that ensures $d/2 < c(1 - (1/2)^{n_c})$, user 1's expected payoff has decreased.

VI. Experimental Evaluation

In this section, we experimentally demonstrate the behavior of the different proposed auction-based admission control mechanisms. First we present the experimental setup. Then we discuss the results.

A. Experimental Platform

Mechanisms. We implemented all the proposed admission control mechanisms in Java, including the strategyproof GV (Greedy by Valuation) mechanism (described in Section IV). We also implemented the optimal constant pricing profit (OPT_C) algorithm, described in Section IV-D.

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

- Profit: the sum of the payments of the admitted queries.
- Admission rate: The percentage of queries admitted.
- 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 runtime for each mechanism.

The reported results are the average of running each algorithm on 50 different sets of workload. Note that, for clarity, our figures do not show GV or OPT_C as they echo the behavior of Two-price in all experiments.

TABLE III
Workload Characteristics

Number of workload sets	50
Number of queries	2000
Number of operators	$700 \sim 8800$
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	5K-10K-15K-20K

Workload. We summarize the workload parameters in Table III. 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:

- A 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. To achieve this, we generate a workload with the highest maximum degree of sharing (i.e. 60) and then gradually split the operators of the highest degree and distribute the resulting operators into other varying degrees within a workload. For example, to generate an input instance of maximum degree of sharing 7, using the input instance of max degree of sharing 8, if there were 100 operators with degree 8, we split each one of them to degrees 4,2,1,1 (four operators). This will generate 400 new operators with the same load as the original operators. The queries associated with that operator will be distributed among the resulting operators. Each input instance consists of 2000 queries and between 700 and 8800 operators (the number of operators decreases as the degree of sharing increases).

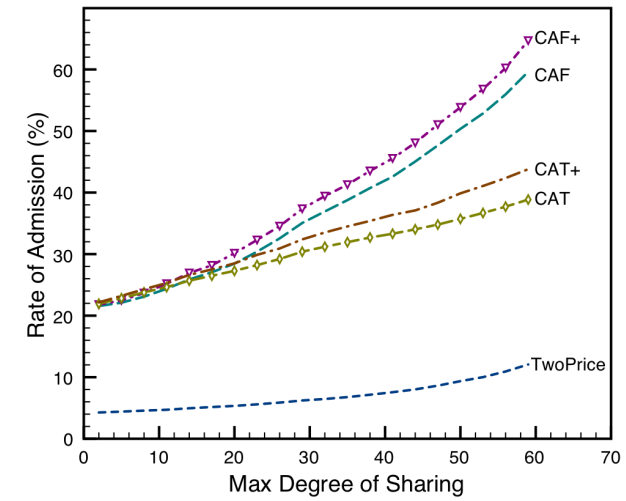
The bids of each query are randomly generated according to a Zipf distribution with maximum bid value set to 100 and skewness parameter set to 0.5. The load of each operator is also randomly generated according to a Zipf distribution with the maximum operator load set to 10 units and skewness parameter set to 1. Operators are assigned to queries randomly, where for each operator, the number of queries sharing it is drawn from a Zipf distribution with skewness parameter set to 1 and the maximum degree of sharing changes from 1 to 60.

B. Experimental Results.

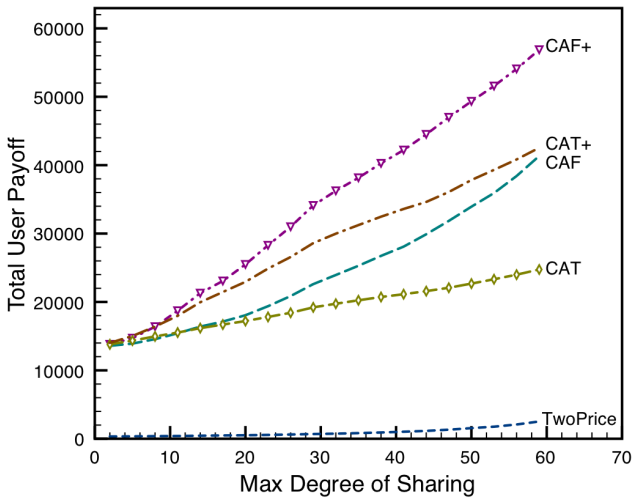
Figure 4(a) shows the percentage of admitted queries as the degree of sharing ranges from 1 to 60, for a system with capacity 15,000. All mechanisms admit more queries as the degree of sharing increases. This is due to the fact that the mechanisms are able to take advantage of the shared processing between queries, so more queries can be serviced using the same system capacity. Two-price always admits a smaller percentage of the queries than the density-based mechanisms (CAF, CAF+, CAT, CAT+) because it chooses queries by user bid only, without regard to query load.

Interestingly, profit (in Figure 4(e)) does not follow the same trend. 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 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 admitted queries continues to be driven downward. Due to the fact that 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 means higher prices, more queries means lower prices. The Two-price mechanism provides profit that consistently improves as degree of sharing increases because its profit is close to the optimal constant pricing profit, which only improves as the number of queries that can fit within capacity increases. At the point where Two-price crosses over CAF and CAT, we observe the same phenomenon that caused decreasing profit in CAF+ and CAT+. At the crossover point, CAF and CAT begin to admit such a high rate of queries that the prices they are charging are being driven dramatically downward (remember, 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 in particular 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.

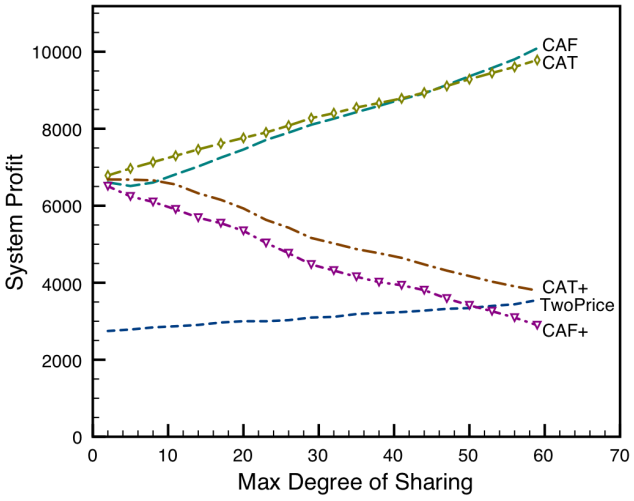
With respect to maximizing total user payoff (Figure 4(b)), the density based mechanisms always perform better than Two-price because they are able to admit more queries and



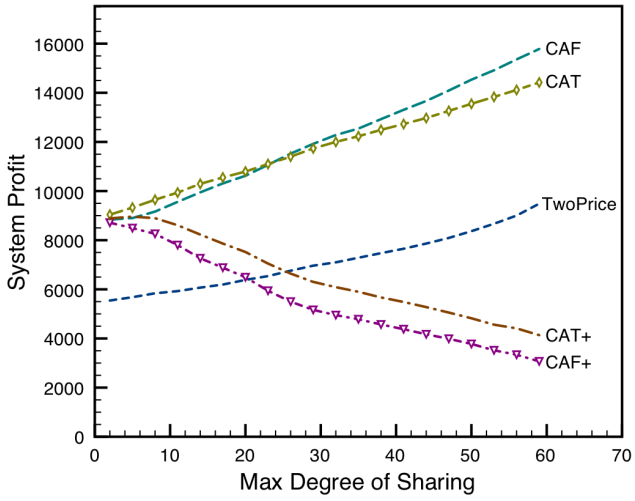
(a) System Capacity = 15,000



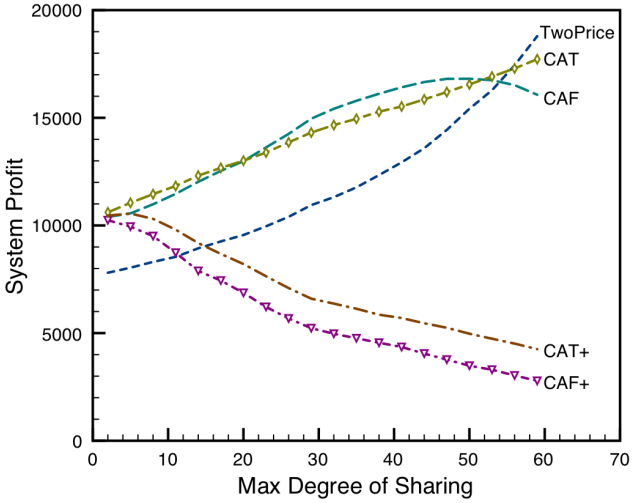
(b) System Capacity = 15,000



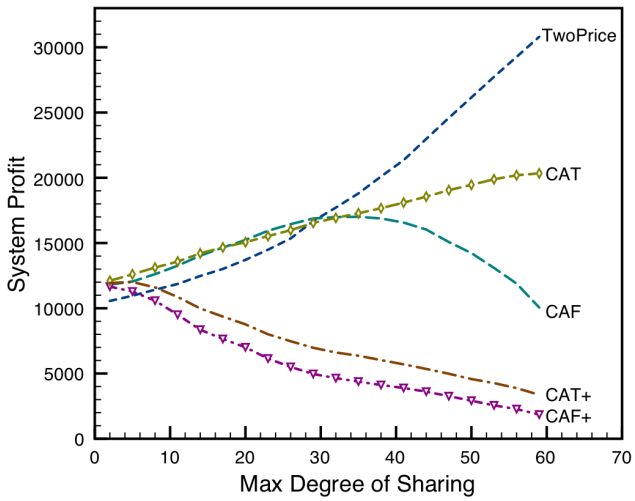
(c) System Capacity = 5000



(d) System Capacity = 10,000



(e) System Capacity = 15,000



(f) System Capacity = 20,000

Fig. 4. Figure 4(a) shows the percentage of queries serviced under each mechanism. Figure 4(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. The sequence of figures in 4(c) through 4(f) shows system profit as system capacity varies from 5000 to 20,000 in increments of 5000.

satisfy more customers. 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.

In terms of utilization, we found that all proposed mechanisms admit queries so as to utilize more than 98 percent of the system capacity, except for Two-price which utilizes between 96 percent and 98 percent.

In Figure 4, we show the system profit for three other system capacities. As system capacity increases, it is apparent that the crossover points (between CAF+, CAT+ and Two-price and between CAF, CAT and Two-price) are shifted to the left, to lower degrees of sharing. Indeed, as capacity increases, the picture as a whole seems to shift and scale downward to the lower end of max degree of sharing. When system capacity is close to the total query demand and sharing is high, the Two-price mechanism has clearly overtaken all the density mechanisms for highest profit. As described above, this is due to the fact that so many of the queries are being serviced by the density mechanisms, driving down the prices being charged.

TABLE IV

RUNTIME PERFORMANCE AVERAGES FOR EACH ALGORITHM ON 50 WORKLOADS WITH 2000 QUERIES

Random	GV	Two-price	CAF	CAF+	CAT	CAT+
0.92	2.003	3.72	7.088	12555.5	7.26	10091.2

We list the average runtime performance of each mechanism over all workloads with 2000 queries and capacity 15K in Table IV. As a baseline, we also implemented a randomly admitting algorithm, which picks queries at random and stops at the first query that does not fit in the remaining capacity. The algorithms ran on an Intel Xeon 8 core 2.3GHz, with 16GB of RAM. The mechanisms only utilized one core. It is clear that the more aggressive mechanisms (CAF+ and CAT+) cannot scale compared to the simple ones. We note here that even though the density based mechanisms' runtime is only three to seven times more than the baseline random algorithm, they provide strategyproofness, and moreover CAT also provides sybil-immunity.

Manipulation of the System. Finally, we evaluate CAR for profit both in a setting where users are being truthful about their valuations, and in a setting where they strategize and bid less than their true valuations (i.e., "lie"). Since CAR is the only mechanism that is not strategyproof, such lying under CAR is to be expected.

To simulate strategizing users, we add an alternative bid to each client, which represents a lower bid than her valuation, and it is the product of her query valuation (original bid) and a lying factor. If a user's query shares many operators with other queries, she would strategize by bidding lower than her valuation thus lowering her payment and increasing

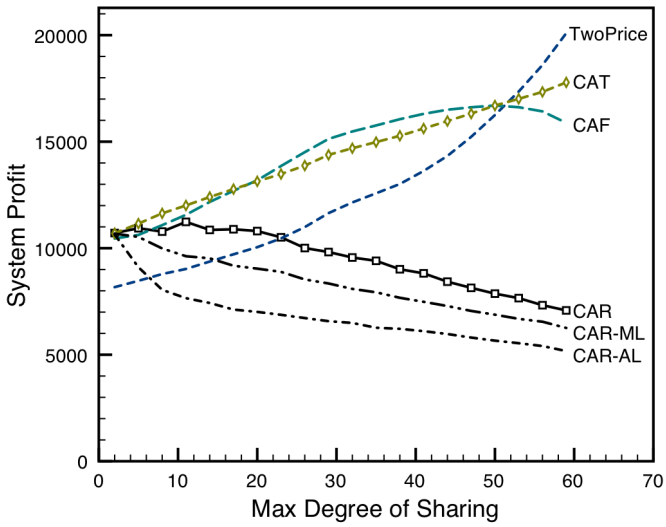


Fig. 5. Profit of the three strategyproof mechanisms, (CAF, CAT, and Two-price), in comparison with the following different representations of the non-strategyproof mechanism CAR: CAR when no user lies, CAR-ML(CAR running the Moderate Lying workload) and CAR-AL(CAR running the Aggressive Lying workload). System capacity = 15,000.

her payoff. Therefore, if the ratio of *Static Fair Share/Total Load* is less than a certain threshold, the client will lie (i.e., submit the alternative bid) with a certain probability. We generated two workloads: a moderately lying workload and an aggressively lying workload. In the moderately lying workload, the threshold is set to 0.25, the probability of lying to 0.5, and the lying factor to 0.5, while in the aggressively lying workload, they are set to 0.35, 0.7 and 0.3 respectively.

Figure 5 shows the profit for three strategyproof mechanisms, CAF, CAT and Two-price, along with three different representations of the profit of CAR: CAR when no user lies, CAR-ML (CAR running the Moderate Lying workload) and CAR-AL (CAR running the Aggressive Lying workload). We see that when some users lie, the system profit decreases, motivating the need and desire of the system for a strategyproof mechanism. The profit of the three strategyproof mechanisms is dependable, while the profit from CAR is manipulable.

VII. DISCUSSION

Table VII summarizes the desirable characteristics of each mechanism alongside its performance for various metrics like profit maximization, total user payoff, and rate of admission.

To extend the proposed solutions to the more general setting of different queries wanting different minimum subscription lengths, we propose the following. Assume without loss of generality that the minimum subscription lengths the system wishes to offer are one day, one week, one month, and one year. Let each of these lengths be referred to as a subscription category. Partition system resources so that an appropriate fraction of total system capacity is allocated to each subscription category. For the queries in each category, run the strategyproof auction mechanism of your choice (see Table VII) with the amount of system capacity allotted to that

category. At the end of each day, reclaim the system capacity from those whose subscriptions expire that day and iterate: partition the currently remaining available system resources among the different categories of subscriptions and again run a separate auction mechanism for each category.

TABLE V
PROPERTIES OF OUR PROPOSED AUCTION MECHANISMS.

Mechanism[a]	Sybil Immune	Profit Guarantee	Admiss Rate	User Payoff	Profit
CAF	×	×	High	Med	High
CAF+	×	×	High	High	Low
CAT	✓	×	Med	Med	High
CAT+	×	×	Med	High	Low
Two-price	×	✓	Low	Low	Med

[a] All mechanisms listed here are strategyproof. Admission Rate, Total User Payoff, and Profit are in terms of relative performance as degree of sharing increases. For Profit, in the special case that degree of sharing is high and system capacity is almost as high as total system demand, the profit from CAF and CAT begins to dwindle and the profit from Two-price is highest.

The good news is that because each auction is being run independently and separately, and all our auctions are bid-strategyproof, this scheme as a whole is still bid-strategyproof. However, introducing these repeated rounds of auctions introduces a new type of potential strategic behavior. Under such a scheme, users may not be honest about the subscription periods they are interested in. For example, a user who wants to run a CQ for one month in July may instead bid for a two month subscription starting in June if she believes demand is low enough in June to get charged a sufficiently low price that paying for two months is cheaper than paying for one month starting in July. Guarding against this sort of strategic behavior in addition to maintaining bid-strategyproofness would be a challenging problem for future work.

Another issue to consider is the energy consumption of the DSMS center. Different levels of system operation incur different energy costs. This can be coupled with the observation that it might be more profitable not to fully utilize the available capacity. Indeed, this is what our experiments clearly suggest. Hence, an extension is to decide what is the most beneficial capacity for a given auction, while considering both the profit as well as the savings from energy reduction.

VIII. CONCLUSION

This work sits at the intersection of two different lines of data management research, namely user-centric data management and data stream management, and utilizes techniques from the domain of game theory. By using an auction model, we are able to explore a novel way of describing user preferences in the CQ admission control problem. 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 is sybil immune.

Our results show that, generally speaking, CAT and CAF are the best mechanisms to use for profit maximization. However, if you have a high degree of operator sharing, and your system capacity is close to the total demand of the queries requesting service, then Two-price performs better for profit maximization. As expected, the greedy mechanisms (CAF, CAF+, CAT, and CAT+) provide better admission rate and payoff than Two-price. CAF+ and CAT+ are best for total user payoff, while CAF and CAF+ have the highest query admission rate as the degree of sharing increases. We showed that CAT, the one mechanism which is sybil immune, offers the best tradeoff with respect to profit. All things considered, this mechanism currently provides most of the desirable properties to be used for admitting CQs in the cloud.

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

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*, 12(2):120–139, 2003.
[2] T. S. Group, "Stream: The stanford stream data manager," *IEEE Data Engineering Bulletin*, 2003.
[3] M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, and K. Pruhs, "Algorithms and metrics for processing multiple heterogeneous continuous queries," *ACM Trans. Database Syst.*, 33(1):1–44, 2008.
[4] "Streambase," 2006. Available: http://www.streambase.com
[5] System S. Available: http://domino.research.ibm.com/comm/research_projects.nsf/pages/esps.index.html
[6] Coral8. Available: http://www.coral8.com/
[7] S. Reiss, "Cloud computing. available at amazon.com today," *Wired*, April 2008. Available: http://www.wired.com/techbiz/it/magazine/16-05/mf_amazon
[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] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman, "Continuously adaptive continuous queries over streams," in *SIGMOD 2002*, pp. 49–60.
[11] S. Krishnamurthy, C. Wu, and M. Franklin, "On-the-fly sharing for streamed aggregation," in *SIGMOD 2006*. ACM, pp. 623–634.
[12] G. Aggarwal and J. D. Hartline, "Knapsack auctions," in *SODA*, 2006.
[13] U. Feige, D. Peleg, and G. Kortsarz, "The dense -subgraph problem," *Algorithmica*, 29(3):410–421, 2001.
[14] N. Nisan, "Introduction to mechanism design," in *Algorithmic Game Theory*, 2007.
[15] D. J. Lehmann, L. O'Callaghan, and Y. Shoham, "Truth revelation in approximately efficient combinatorial auctions," *J. ACM*, 49(5):577–602, 2002.
[16] L. Blumrosen and N. Nisan, "Combinatorial auctions," in *Algorithmic Game Theory*, 2007.
[17] A. V. Goldberg, J. D. Hartline, A. R. Karlin, M. Saks, and A. Wright, "Competitive auctions," in *Games and Economic Behavior*, 2006.
[18] C. Chung, "Evolutionary solutions and internet applications for algorithmic game theory," Ph.D. dissertation, U. of Pittsburgh, Pittsburgh, PA, Aug. 2009.
[19] E. Friedman, P. Resnick, and R. Sami, "Manipulation-resistant reputation systems," in *Algorithmic Game Theory*, 2007.