Performance Evaluation of Scheduling Algorithms for Database Services with Soft and Hard SLAs

Hyun Jin Moon Yun Chi Hakan Hacıgümüş
NEC Laboratories America
10080 North Wolfe Rd, SW3-350 Cupertino, CA 95014, USA
{hjmoon, ychi, hakan}@sv.nec-labs.com

ABSTRACT

Database service providers face heterogeneous customer workloads with widely varying characteristics. The query scheduling plays a critical role in serving such workloads and involves the careful consideration of specific requirements introduced in service provisioning environments. As a part of our real platform building process, we have collected specific requirements for the scheduling framework through extensive interactions with business organizations that provide services to real clients. Although there is a very large body of previous work in the scheduling area, there is no single scheduling method that is designed to satisfy all of our requirements. However, some of them may address certain aspects of those requirements. In this work, we rigorously evaluate a comprehensive set of such scheduling methods and present how they perform with respect to the full requirements list. We also propose and evaluate an effective extension to the most promising method, iCBS, we identified through the evaluations in this space.

1. INTRODUCTION

Cloud computing has emerged as a promising computing and business model. By providing on-demand scaling capabilities without any large upfront investment or long-term commitment, cloud computing is attracting a large variety of applications. The database community has also shown a great interest in exploiting this new platform for data management services [1, 3] in a highly scalable and cost-efficient manner [2, 9, 7, 19].

Cloud computing presents many challenges and opportunities for data management services. For instance, database service providers may face heterogeneous customer workloads with widely varying characteristics. To serve such workloads, they may have to use a diverse set of specialized database products and technologies to ensure that customers obtain the benefits of those products specifically tailored for their needs.

With this motivation, we are building a data management platform in the cloud, called *CloudDB* [14, 22, 30, 6, 25, 5, 31] at NEC Labs. One important component of the CloudDB is the Intelligent Cloud Database Coordinator (ICDC), which is responsible for all functions and decisions regarding workload dispatching, workload scheduling, and resource capacity planning, as shown in Figure 1.

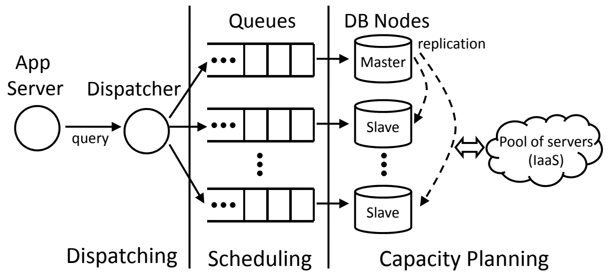


Figure 1: Conceptual system diagram: main functionality of ICDC

When a query[1] arrives, ICDC first estimates its query execution time[2], based on the workload history and prediction techniques [31]. The dispatcher immediately assigns the query to one of the servers, according to a dispatching policy; at each server, a scheduling policy decides which query should be executed first from the queue. The capacity planning component is in charge of determining how many resources (i.e., database servers) to be allocated to the system. The capacity planner is aware of the status of the resource scheduler and that of the query dispatcher and use them for capacity planning decisions. While all of these three components are important for ICDC, in this paper we focus on workload scheduling aspect of ICDC. In order to effectively manage and control CloudDB, a database service offering, ICDC focuses on *Service Level Agreements (SLAs)* and the service provider's *revenue* as the two main metrics to optimize, rather than low level system metrics such as average response time or system throughput.

During the design of such a system, we have collected specific requirements for the scheduling framework through extensive interactions with business organizations that provide services to real clients, as follows:

[1]We use *query* and *job* interchangeably in this paper.
[2]As an alternative, one may estimate the low-level resource demand of individual queries (e.g. CPU, I/O, and memory) and manage scheduling, dispatching, and capacity based on that, as done in [4]. In this paper, we take a higher level approach based on execution time, an approach that we believe to be more manageable and yet quite effective [15].

- Service providers' profit should be the main metric of optimization.
- There should be a capability to model and manage SLAs that translate varying levels of service quality into service providers' profit or penalty, e.g. a step function, which we refer to as *soft SLA* in this paper.
- There should be a capability to define a separate service level objective (i.e. termed *hard SLA* in this paper) for some or all of the jobs in addition to the soft SLAs above: it is highly desired to meet the hard SLA, while its violation may or may not have direct monetary penalties.
- The system should be able to manage the SLAs at the finest granularity level, i.e., per job basis, as opposed to coarser granularity levels, such as a percentile basis.
- The system should be able to support multiple users, who share the same system resources, with multiple job classes, and multiple SLA definitions corresponding to those job classes.
- The complexity of the scheduling framework should be very small to cope with a high job arrival rate or bursts in the real system.

Although there is a very large body of previous work in the scheduling literature [27], to the best of our knowledge there is no single scheduling method that is designed to satisfy all of our requirements. However, some of them may address certain aspects of those requirements. In this work, our goal is to rigorously evaluate a comprehensive set of such scheduling methods and present how they perform with respect to the full requirements list. We also propose and evaluate an extension to the most promising method, iCBS [5], based on our evaluations.

As mentioned in the requirements, we consider two types of SLAs, namely *soft SLAs* and *hard SLAs*: the soft SLAs describe SLA profit as a function of response time. For example, a job that incurs no penalty for the service provider if it finishes within 500 ms and $1 penalty otherwise. As there are a number of users in the system with individual SLAs, the service provider may choose to serve the jobs from different users at varying levels instead of trying to serve them equally to optimize the profit. A hard SLA specifies a single firm deadline objective for each job. The objective is defined either by the user or the service provider. The violation of the objective may (e.g., penalty payment or service fee discount) or may not (e.g., poor user experience or a bad company image) have direct monetary consequences. While some systems have only one of these two SLA types, we consider the case where the two co-exist, which is the case for our system. We seek to manage both SLA profit under soft SLAs and the deadline violation count under hard SLAs.

Toward this goal, we make the following contributions:

Identification of Dual-SLA Problem We identify and formalize the profit and deadline violation management under soft and hard SLAs.

Scheduling for both Soft and Hard SLAs We propose a scheduling policy called iCBS-DH, which is an extension of existing cost-aware scheduler called iCBS [5]. By adding a small but effective feature called deadline hint to the deadline-unaware iCBS, we create a new scheduler that can handle both soft and hard SLAs.

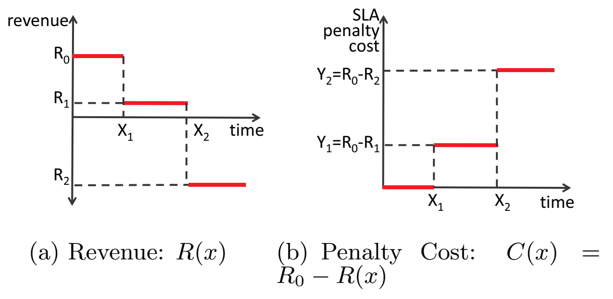


Figure 2: SLA revenue and penalty cost functions

Experimental study We evaluate a comprehensive set of scheduling policies in an extensive experimental study. The experimental study covers a wide range of variables in real systems, including various shapes of SLA functions and deadlines, varying workload characteristics, system load levels, and service cost structures etc.

2. SLA AND PROFIT MODELS

2.1 Service Level Agreements

SLAs in general may be defined in terms of various criteria, such as service latency, throughput, consistency, security, etc. In this paper, we focus on service latency, or response time. Even for the response time alone, there can be different specification methods such as based on i) the average query response time [34], ii) the tail distribution of query response time [21], or iii) the individual query response time [15]. We choose the last one, as we outlined in our requirements

For each specification method, we can also design SLA either as a soft SLA or a hard SLA as follows.

- **Soft SLA**: A soft SLA corresponds to agreed levels of service in the contract. An SLA penalty (cost) function may have various shapes, such as a linear function, a stepwise function, piecewise linear function, or any general curve. As in [34], we believe that the stepwise function shown in Figure 2(a) is one of the most natural choices for the real-world contracts since it is easy to be described in natural language. In the SLA revenue function shown in Figure 2(a), the client pay i) R_0 if the corresponding query finishes within X_1, ii) R_1 if the query finished between X_1 and X_2, or iii) R_2 if the query response time is more than X_2. [3]
- **Hard SLA**: A hard SLA has a single *hard deadline* to meet, and if the deadline is missed, it is counted as a *violation*. The specification of this type of SLAs (also referred to as *constraints* in this paper) may come from the client or the cloud service provider. For example, there are cases where a cloud provider needs to use hard SLAs as a tool to control various business objectives, e.g., controlling the worst case user experience. Therefore the violation of a hard SLA may or may

[3]Note that when the response time becomes long enough to enter the last cost step, service providers may choose to drop the job since they already incurred the highest SLA cost possible, and they have no economic incentive to continue serving the job. In this paper, for simplification, we never intentionally drop a job.

not correspond to direct financial terms in the client contracts.

2.2 Profit Model

In our system, we support multiple clients, where each client has its own database and an independent workload. Each client has one or more *job classes*, where each job class has a common SLA, shared by all jobs in the job class.

Given the SLA revenue function, $R(x)$, defined as above, we derive the *SLA penalty cost function*, $C(x)$, as:

$$C(x) = R_0 - R(x)$$

An example is shown in Figure 2(b). Note that profit maximization based on $R(x)$, which is our objective, is equivalent to minimizing SLA penalty cost based on $C(x)$. In the rest of the paper, we focus on minimization of $C(x)$.

3. SCHEDULING ALGORITHMS

In this section, we introduce the scheduling algorithms.

3.1 Cost- and Deadline-unaware Scheduling

FCFS: First-Come First-Served. This is the most popular type of scheduling policy, where jobs are executed in the order of arrival.

SJF: Shortest Job First. Among the jobs in the queue, the one with the shortest execution time (i.e. service time) is scheduled first. Once scheduled, the job runs to finish without preemption. SLA cost function or deadline information is not used for the scheduling decision.

3.2 Deadline-aware Scheduling

EDF: Earliest Deadline First. The job with the earliest deadline is executed first. SLA cost function is not used for the scheduling decision.

AED: Adaptive EDF, proposed in [15]. It tries to avoid the weakness of EDF, which is the domino effect under the overload situation, where all jobs misses the deadline. It does not use SLA cost function information or execution time information. Two parameters in the paper, HITbatch and ALLbatch, are both set to 200, which performed well for our workload.

3.3 Cost-aware Scheduling

BEValue2: A scheduling algorithm proposed in [18]. BEValue2 is a modified version of EDF that addresses EDF's weakness at overload. We consider BEValue2 as cost-aware scheduler, rather than deadline-aware one, since they use the term *deadline* to refer to the time where the SLA cost increases suddenly, similar to our soft SLA cost step time, and it is different from our hard deadline, which does not have any explicit cost associated. So we give SLA cost function information to BEValue2, and not the deadline. It also exploits execution time information. For the scheduler parameter, overload probability threshold, we use 0.4, which is suggested in the paper.

FirstReward: A scheduling algorithm proposed in [17]. While this is a highly sophisticated scheduling policy considering benefit and opportunity cost of each scheduling choice, each scheduling has a high overload of $O(n^2)$, where n is the number of jobs in the queue, and this is often very slow. Note that iCBS below avoids this problem. FirstReward uses SLA cost function information and execution time information, but not the deadline. For the parameters, alpha and discountRate, we use 0.3 and 0.01.

iCBS: Peha and Tobagi have proposed a heuristic-based cost-based scheduling policy, called CBS [23, 24], which has superior performance in terms of query cost minimization. The high level idea is to first re-evaluate the priorities of all jobs in the queue at the moment when a scheduling decision is needed and then pick the query with the highest priority, at that given moment, which in turn maximizes the expected global total profit. To evaluate the priority of job i, CBS considers two possible cases: i) the job is chosen to be scheduled now, or ii) some other job is chosen. The former case will incur a cost of $c_i(t)$ to the job i, where $c_i(t)$ is the cost function of job i and t is the time job i has been staying in the queue so far. For the case ii, it is assumed that the job gets delayed by an additional wait time, τ, before it is served, which will result in a cost of $c_i(t+\tau)$. Since the value of τ is not known, CBS uses a probability density function of τ, $a(\tau)$, and compute the *expected cost* using it. Based on these two scenarios, the CBS priority for a job i is defined as:

$$p_i(t) = \int_0^\infty a(\tau) \cdot c_i(t+\tau)\ d\tau - c_i(t) \qquad (1)$$

After the $p_i(t)$ value is computed, it is then divided by the job's service time, since longer job occupies the server for a longer time, delaying other jobs for a longer time. CBS chooses the job with the highest priority among all jobs in the queue.

In general, it is difficult to find an optimal $a(\tau)$, but the authors show that the exponential function, $a(\tau) = 1/\beta \cdot e^{-\tau/\beta}$, works well in practice. β needs to be set to average execution time and we use β=1 msec in our evaluation.

CBS has a time complexity of $O(n)$, where n is the number of jobs in the queue. This is because CBS examines all the jobs in the queue in order to pick the one with the highest priority. In real systems where queues can grow arbitrarily long and job service time can be very short, O(n) may not be acceptable. To address this problem, [5] proposes iCBS, that incrementally maintains CBS priority score, and keeps the scheduling overhead at $O(log^2 n)$.

3.4 Cost- and Deadline-aware Scheduling

iCBS-DH: While CBS and iCBS considers the SLA cost function and tries to minimize the cost, they cannot support the hard deadlines. In this section, we discuss how we extend iCBS into iCBS-DH, to address this problem.

We uses iCBS as is, to leverage its cost-optimizing feature, and make some modification to the SLA cost function as follows. For a given job, we shift up $cost(t)$ to $cost(t) + C_{hint}$, at response time $t > deadline$, with a given constant value, C_{hint}, the *hint cost*. This is illustrated in Figure 3. With this modified SLA cost function, iCBS performs its original cost-minimizing scheduling algorithm, which effectively minimizes deadline violation as well as soft SLA cost. We call this scheduling method, iCBS-DH (i.e. deadline hint).

Note that we introduce a parameter, hint cost. The hint cost value determines the importance of the hard deadline: high hint cost will make the deadline a higher priority, compared to the soft SLA cost steps, and low hint cost will make the deadline a lower priority. In general, we use the hint cost of \$1000 for our evaluation, which sets a high priority on the deadline enforcement given that soft SLA cost

Query	ExTime	CostDensity			CostStepTime (msec)				HardDeadlineTime (msec)				
Type	(msec)	1	2	3	1	2	3	4	1	2	3	4	5
Q1	0.23	3	1	5	20	10	30	20-40	10	20	30	10	30
Q2	0.23	3	2	4	20	15	25	20-40	10	20	30	15	25
Q3	0.30	3	3	3	20	20	20	20-40	10	20	30	20	20
Q4	0.41	3	4	2	20	25	15	20-40	10	20	30	25	15
Q5	0.54	3	5	1	20	30	10	20-40	10	20	30	30	10

Table 1: Experiment parameters for soft and hard deadlines. ExTime is the average query execution time for the given query type. CostDensity has the unit of \$/msec.

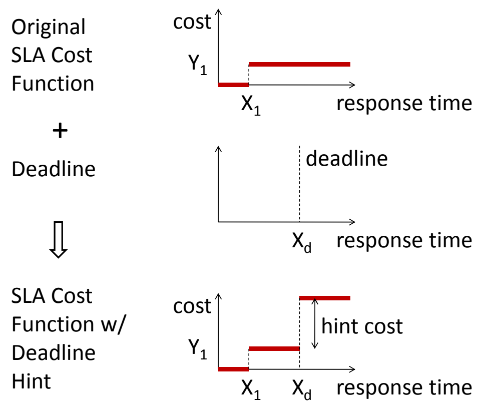


Figure 3: iCBS-DH: iCBS with Deadline Hint

steps are lower than \$2. We experimentally study the effect of the deadline hint cost in Section 4.2.6.

4. EVALUATION

4.1 Setup

Systems and Workload The server machine has Intel Xeon 2.4GHz, two single-core CPUs and 16GB memory. We use MySQL 5.5 and InnoDB 1.1.3, and 1GB of memory is used for InnoDB bufferpool. The client machine has Xeon 2.4GHz, two quad-core CPUs. The client code is in Java.

We use TPC-W 1GB dataset and six queries obtained from the same benchmark. Each query gives us a query template or a query type, from which we generate query instances using randomly generated parameters. We use uniform distribution among the query types used in each experiment. We run open-system workload [28], where new queries arrive at the system queue, independent of finishing of existing queries. In order to vary the system load, i.e. the query arrival rate divided by the service rate, or equivalently, the average query execution time divided by the mean interarrival time, we control the arrival rate as described further later. As a default, 85% system load is used, except in Section 4.2.3 where we vary the load and observe its effect. Given the arrival rate, Poisson distribution is used to generate individual query arrival timestamps. We report the average of five repeats for each data point, where individual run duration is 5 seconds: this may sound short, but for our queries that are shorter than 1 msec, we get more than 10,000 queries finished during this interval, which is a large enough sample size for scheduling performance study.

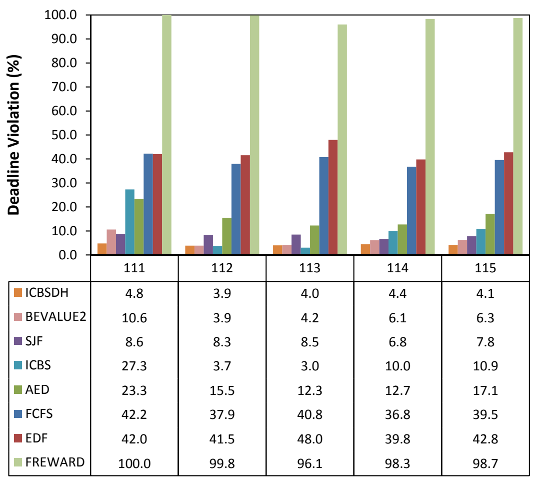
Query Execution Time Estimate Some scheduling policies, i.e. SJF, FirstReward, BEValue2, iCBS, iCBS-DH, rely on the query execution time estimates for their scheduling decisions. While there are recent works on query execution time estimate [10, 8], in our experiments, we use a simple, but highly effective method for per-template execution time estimation, as follows. For each query template, we continuously maintain the mean and the standard deviation (SD) of all query execution times. From these measurements, we get $mean + SD$[4] and use it as the query execution time estimate. This worked well across different load and MPL (multi-programming level) values in our evaluation.

SLA Design In order to evaluate scheduling algorithms against various SLA cost functions and deadlines, we design the experiment parameters as follows. First, we model the SLA cost function as a step function with two (or three) steps, where the response time greater than a cost-step-time incurs a certain penalty, or monetary SLA cost. We use two parameters for soft SLA cost, *CostDensity* and *CostStepTime*, and one for hard deadline, *HardDeadlineTime*. The full parameter code-value mapping is shown in Table 1.

- **CostStepTime (msec)**: SLA cost increases at the point where the query response time becomes greater than CostStepTime. It corresponds to the time X_1 in Figure 3. As shown in Table 1, CostStepTime takes one code between 1 and 4. Code 1, for example, means that all queries have the second cost step at X_1=20 msec (and no more steps). Code 4 is a special case where we have three cost steps, like X_1 and X_2 in Figure 2(b): X_1=20 msec and X_2=40 msec.

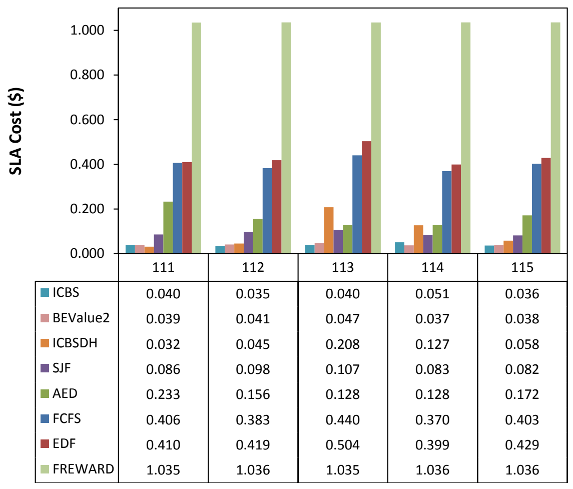
- **CostDensity (\$/msec)**: When the response time becomes greater than CostStepTime, the cost jumps: as high as Y_1 in the example of Figure 3. The cost jump is determined as the query execution time, i.e. ExTime, multiplied by CostDensity. For example, given CostDensity code 1, Q1 has the cost jump of Y_1=0.23 msec $\times$ \$3/msec = \$0.69, at the cost step X_1. Note that we control and use cost density rather than cost value itself, since we consider density as a more meaningful parameter: queries have different execution times and long-running queries often have more value/cost than short queries. For the special case of CostStepTime code 4, ExTime$\times$CostDensitythe determines the second step's cost, i.e. Y_1 in Figure 2(b), and the third step's cost is twice that, i.e. $Y_2 = 2 \cdot Y_1$.

[4]We tried *mean* as the estimate as well, which gave us a poor result: schedulers like iCBS makes scheduling decisions with an absolute trust on the estimate, and the estimate without margin often result in missed cost steps.



	111	112	113	114	115
ICBSDH	4.8	3.9	4.0	4.4	4.1
BEVALUE2	10.6	3.9	4.2	6.1	6.3
SJF	8.6	8.3	8.5	6.8	7.8
ICBS	27.3	3.7	3.0	10.0	10.9
AED	23.3	15.5	12.3	12.7	17.1
FCFS	42.2	37.9	40.8	36.8	39.5
EDF	42.0	41.5	48.0	39.8	42.8
FREWARD	100.0	99.8	96.1	98.3	98.7

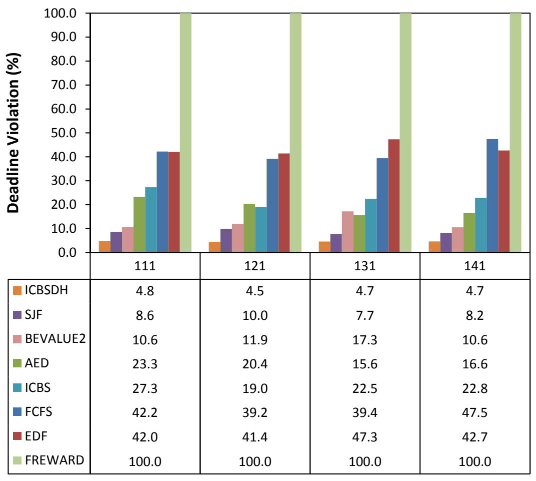
(a) Deadline Violation



	111	112	113	114	115
ICBS	0.040	0.035	0.040	0.051	0.036
BEValue2	0.039	0.041	0.047	0.037	0.038
ICBSDH	0.032	0.045	0.208	0.127	0.058
SJF	0.086	0.098	0.107	0.083	0.082
AED	0.233	0.156	0.128	0.128	0.172
FCFS	0.406	0.383	0.440	0.370	0.403
EDF	0.410	0.419	0.504	0.399	0.429
FREWARD	1.035	1.036	1.035	1.036	1.036

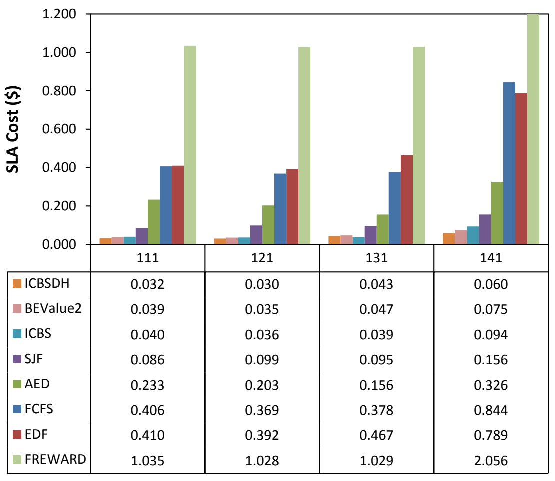
(b) SLA Cost

Figure 4: Performance of Scheduling Policies under Varying SLA Cost Functions and Deadlines (DTH=11x)



	111	121	131	141
ICBSDH	4.8	4.5	4.7	4.7
SJF	8.6	10.0	7.7	8.2
BEVALUE2	10.6	11.9	17.3	10.6
AED	23.3	20.4	15.6	16.6
ICBS	27.3	19.0	22.5	22.8
FCFS	42.2	39.2	39.4	47.5
EDF	42.0	41.4	47.3	42.7
FREWARD	100.0	100.0	100.0	100.0

(a) Deadline Violation



	111	121	131	141
ICBSDH	0.032	0.030	0.043	0.060
BEValue2	0.039	0.035	0.047	0.075
ICBS	0.040	0.036	0.039	0.094
SJF	0.086	0.099	0.095	0.156
AED	0.233	0.203	0.156	0.326
FCFS	0.406	0.369	0.378	0.844
EDF	0.410	0.392	0.467	0.789
FREWARD	1.035	1.028	1.029	2.056

(b) SLA Cost

Figure 5: Performance of Scheduling Policies under Varying SLA Cost Functions and Deadlines (DTH=1x1)

- **HardDeadlineTime (msec)**: For different types of soft and hard deadline interaction, we design different cases using HardDeadlineTime: HardDeadlineTime is earlier than CostStepTime (i.e. a preventative deadline designed to avoid SLA cost), or it is later than CostStepTime (i.e. a deadline for worst-case quality control even after incurring SLA cost).

Since these three parameters decide the soft and hard SLA used in an experiment, so we concatenate the three codes together and call it *DTH code*[5], or simply DTH. For instance, with DTH=131 means that CostDensity code is 1, CostStepTime code is 3, and HardDeadlineTime code is 1. Given these parameter codes, different queries have different parameter values: in case of Q1, this indicates CostDensity value of \$3/msec, CostStepTime value of 30 msec, and HardDeadlineTime value of 10 msec.

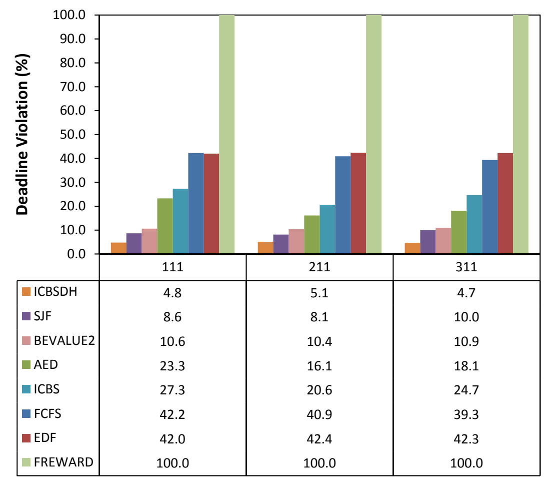
[5]DTH: Cost**D**ensity, CostStep**T**ime, **H**ardDeadlineTime

4.2 Results

4.2.1 *Varying SLA and Deadlines*

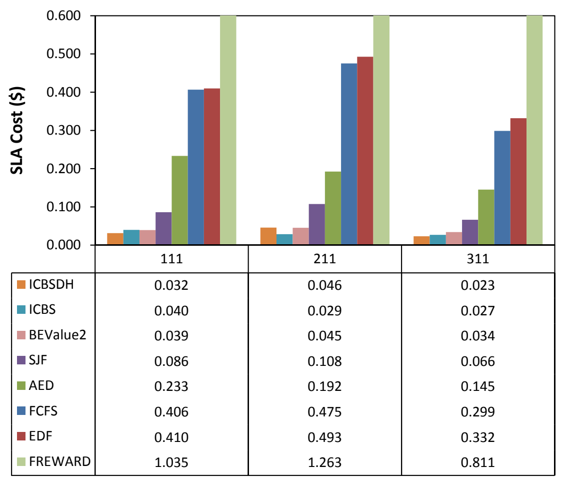
We first show the performance of eight scheduling algorithms under varying SLA cost function and hard deadlines. Figures 4, 5, and 6 show the results. With the default DTH=111, we vary one parameter at a time in each figure: HardDeadlineTime in Figure 4 (i.e. DTH=11x), CostStepTime in Figure 5 (i.e. DTH=1x1), and CostDensity in Figure 6 (i.e. DTH=x11).

Observations on deadline violations are as follows: i) for deadline violation, iCBS-DH performs the best, keeping the violation at 5.1% or lower in all cases. After that follow SJF (as high as 10.0% violation), BEValue2 (up to 17.3%), iCBS (up to 27.3%), and AED (up to 23.3%). Note that their deadline performance fluctuate depending on their SLA cost function shape and deadline while iCBS-DH maintains low violation overall all ranges. ii) FCFS and EDF have rather high deadline violations at around 40% as both are vulnerable to (temporary) overloads, causing domino effects,



	111	211	311
ICBSDH	4.8	5.1	4.7
SJF	8.6	8.1	10.0
BEVALUE2	10.6	10.4	10.9
AED	23.3	16.1	18.1
ICBS	27.3	20.6	24.7
FCFS	42.2	40.9	39.3
EDF	42.0	42.4	42.3
FREWARD	100.0	100.0	100.0

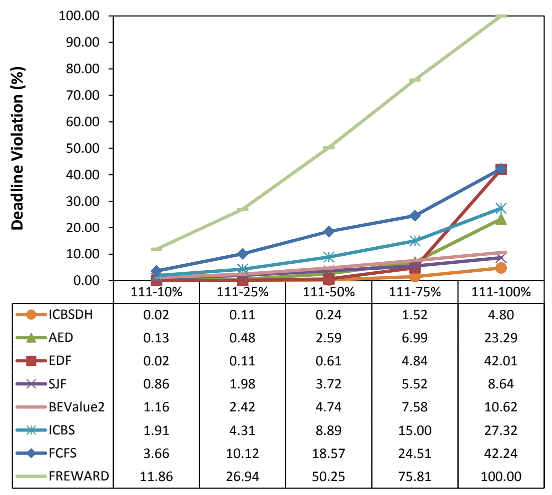
(a) Deadline Violation



	111	211	311
ICBSDH	0.032	0.046	0.023
ICBS	0.040	0.029	0.027
BEValue2	0.039	0.045	0.034
SJF	0.086	0.108	0.066
AED	0.233	0.192	0.145
FCFS	0.406	0.475	0.299
EDF	0.410	0.493	0.332
FREWARD	1.035	1.263	0.811

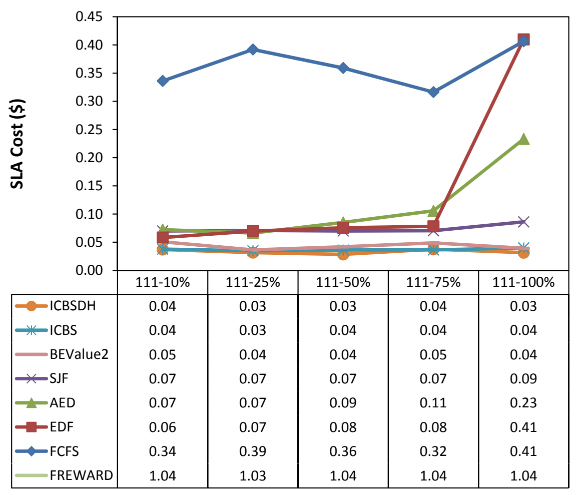
(b) SLA Cost

Figure 6: Performance of Scheduling Policies under Varying SLA Cost Functions and Deadlines (DTH=x11)



	111-10%	111-25%	111-50%	111-75%	111-100%
ICBSDH	0.02	0.11	0.24	1.52	4.80
AED	0.13	0.48	2.59	6.99	23.29
EDF	0.02	0.11	0.61	4.84	42.01
SJF	0.86	1.98	3.72	5.52	8.64
BEValue2	1.16	2.42	4.74	7.58	10.62
ICBS	1.91	4.31	8.89	15.00	27.32
FCFS	3.66	10.12	18.57	24.51	42.24
FREWARD	11.86	26.94	50.25	75.81	100.00

(a) Deadline Violation



	111-10%	111-25%	111-50%	111-75%	111-100%
ICBSDH	0.04	0.03	0.03	0.04	0.03
ICBS	0.04	0.03	0.04	0.04	0.04
BEValue2	0.05	0.04	0.04	0.05	0.04
SJF	0.07	0.07	0.07	0.07	0.09
AED	0.07	0.07	0.09	0.11	0.23
EDF	0.06	0.07	0.08	0.08	0.41
FCFS	0.34	0.39	0.36	0.32	0.41
FREWARD	1.04	1.03	1.04	1.04	1.04

(b) SLA Cost

Figure 7: Performance of Scheduling Policies under Varying Percentage of Deadline-Having Queries

and they run queries even when they already missed the deadline. iii) FirstReward misses almost all deadlines, due to its high scheduling overhead of $O(n^2)$.

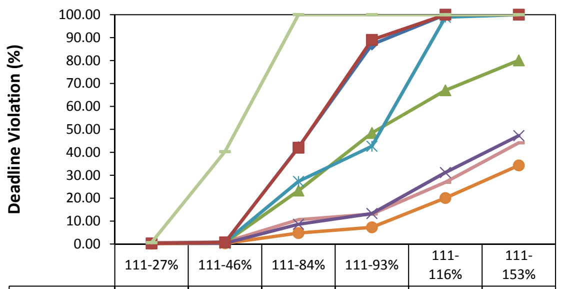
SLA cost performance is summarized as follows: i) iCBS achieves minimum SLA cost in most cases. Compared with FCFS, the most popular and simple scheduling, SLA cost is reduced by the factor of 10. ii) BEValue2 follows iCBS, and iCBS-DH, SJF, AED comes next. Note that iCBS-DH has a fluctuating cost performance over different DTH codes: when deadline observance helps SLA cost (e.g. DTH=111, 112), iCBS-DH cost is as low as that of iCBS. When deadline observance does not necessarily help SLA cost (e.g. DTH=113), iCBS-DH cost is rather high. iii) FCFS, EDF, and FirstReward incur high SLA cost for the same reason mentioned in deadline violations above.

4.2.2 *Varying Portion of Deadline-Having Queries*

In the previous experiment, we have assumed that all queries have hard deadlines. Depending on the business requirements, however, only a portion of queries may have hard deadlines. For example, 25% of all queries are queries coming from important users or applications (e.g. VIPs, shopping applications) and have deadlines, while the others can be served best effort under the soft SLA. We study the scheduling performance where a subset of queries have a deadline, and report the corresponding results in Figure 7. On the x-axis, we vary the portion of deadline-having queries, i.e. 10%, 25%, 50%, 75%, and 100%. The deadline violation percentage on the y-axis indicates the percentage out of all jobs.

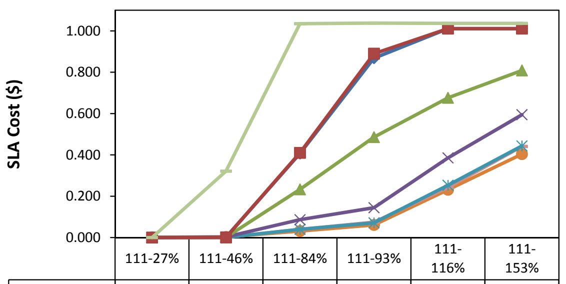
Highlights on deadline performance are as follows: i) iCBS-DH consistently performs the best in most of the cases. ii) EDF performs as well when 50% or less jobs have deadlines, where no (temporary) overload causes a domino effect. With 75% or higher jobs having deadline, however, EDF is hit by the overload problem and gets very high deadline violations.

For cost performance, note that EDF has a similar behavior as in deadline performance mentioned above. iCBS-DH



	111-27%	111-46%	111-84%	111-93%	111-116%	111-153%
ICBSDH	0.17	0.32	4.80	7.31	20.08	34.28
BEVALUE2	0.64	0.91	10.62	13.11	26.95	44.14
SJF	0.33	0.35	8.64	13.31	31.18	47.25
AED	0.31	0.70	23.29	48.42	67.00	80.07
ICBS	0.26	0.87	27.32	42.64	98.88	100.00
FCFS	0.34	0.74	42.24	87.00	100.00	100.00
EDF	0.28	0.77	42.01	89.02	100.00	100.00
FREWARD	0.65	40.31	100.00	100.00	100.00	100.00

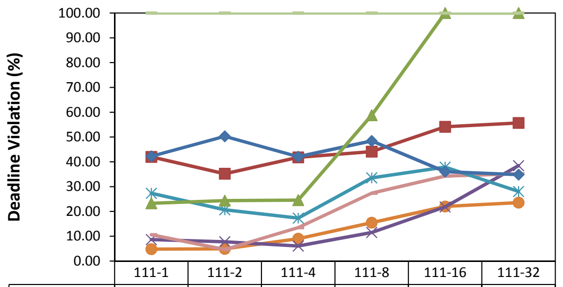
(a) Deadline Violation



	111-27%	111-46%	111-84%	111-93%	111-116%	111-153%
ICBSDH	0.001	0.003	0.032	0.061	0.231	0.404
BEVALUE2	0.000	0.001	0.039	0.074	0.243	0.441
ICBS	0.000	0.000	0.040	0.071	0.253	0.443
SJF	0.001	0.003	0.086	0.144	0.385	0.595
AED	0.000	0.001	0.233	0.486	0.676	0.809
FCFS	0.000	0.001	0.406	0.868	1.011	1.011
EDF	0.000	0.001	0.410	0.890	1.011	1.011
FREWARD	0.000	0.321	1.035	1.038	1.036	1.036

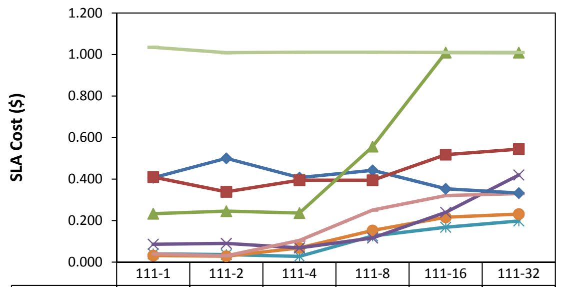
(b) SLA Cost

Figure 8: Performance of Scheduling Policies under Varying Load



	111-1	111-2	111-4	111-8	111-16	111-32
ICBSDH	4.80	4.91	9.01	15.46	22.00	23.55
SJF	8.64	7.78	6.06	11.52	21.81	38.44
BEVALUE2	10.62	4.64	13.48	27.38	34.26	35.20
ICBS	27.32	20.68	17.31	33.54	37.87	28.05
EDF	42.01	35.20	41.81	44.11	54.12	55.68
FCFS	42.24	50.29	42.06	48.42	36.05	34.84
AED	23.29	24.36	24.54	58.78	100.00	100.00
FREWARD	100.00	100.00	100.00	100.00	100.00	100.00

(a) Deadline Violation



	111-1	111-2	111-4	111-8	111-16	111-32
ICBS	0.040	0.036	0.028	0.126	0.168	0.198
ICBSDH	0.032	0.029	0.068	0.152	0.216	0.232
SJF	0.086	0.090	0.069	0.116	0.240	0.420
BEVALUE2	0.039	0.030	0.105	0.251	0.321	0.330
FCFS	0.406	0.500	0.407	0.442	0.353	0.333
EDF	0.410	0.339	0.395	0.394	0.518	0.544
AED	0.233	0.246	0.236	0.556	1.009	1.009
FREWARD	1.035	1.009	1.011	1.011	1.010	1.010

(b) SLA Cost

Figure 9: Performance of Scheduling Policies under Varying Multi-Programming Level (MPL)

achieves low SLA cost, comparable to that of iCBS, when the portion of deadlined jobs is 25% or lower. This is a nice adaptive feature that iCBS-DH gives just enough attention for the deadline satisfaction, and puts rest of its effort on cost reduction like iCBS.

4.2.3 *Varying Load*

We now vary the query arrival rates, to observe the behavior of scheduling algorithms under different system loads. We set the arrival rates at six different levels, and measure the average execution time, from which we compute the load as arrival rate divided by service rate, or arrival rate multiplied by average execution time. Given a query type, note that the execution times are slightly different for the different arrival rates: with higher arrival rate, individual queries run faster, which seems to be affected by the performance behavior of JDBC connection layer. We report the computed load number and the corresponding deadline and cost performance.

Figure 8 shows the results, where the x-axis values, or the table column headers, indicate DTH code and the load in percentage. For deadline violation, iCBS-DH again performs the best across all load ranges. BEValue2 and SJF come close to iCBS-DH, while iCBS, AED, FCFS, EDF, FirstReward has high violations for the reasons mentioned above.

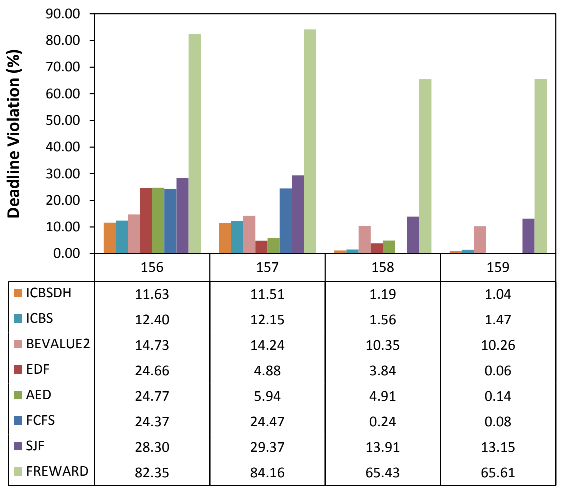
For cost performance, iCBS and BEValue2 are consistently the best. While iCBS-DH shows a good cost performance in Figure 8(b) with DTH=111, in general it has a varying performance: when meeting deadline helps cost, e.g. DTH=111,112, it performs well on cost, and otherwise not (e.g. DTH=113).

4.2.4 *Varying MPL*

In the real-world database systems, it is rarely the case that one query runs at a time. Instead, multiple queries are run concurrently to exploit parallelism in the computing resources, such as CPU and IO. We vary the concur-

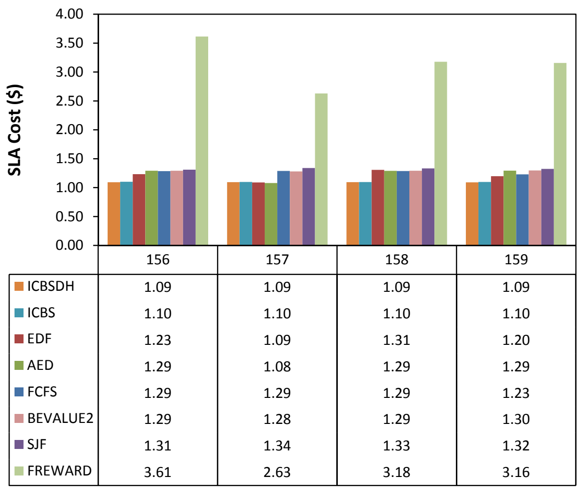
Query Type	ExTime (msec)	CostDensity			CostStepTime (msec)								HardDeadlineTime (msec)								
		1	2	3	1	2	3	4	**5**	**6**	**7**	**8**	1	2	3	4	5	**6**	**7**	**8**	**9**
Q1	0.23	3	1	5	20	10	30	20-40					10	20	30	10	30				
Q2	0.23	3	2	4	20	15	25	20-40					10	20	30	15	25				
Q3	0.30	3	3	3	20	20	20	20-40					10	20	30	20	20				
Q4	0.41	3	4	2	20	25	15	20-40					10	20	30	25	15				
Q5	0.54	3	5	1	20	30	10	20-40	**250**	**250**	**500**	**500**	10	20	30	30	10	**250**	**250**	**500**	**500**
Q6	**158**	**3**	**1**	**5**					**250**	**500**	**250**	**500**						**250**	**500**	**250**	**500**

Table 2: Experiment parameters for hard and soft SLAs. Addition to Table 1 is in bold.



	156	157	158	159
🟧 ICBSDH	11.63	11.51	1.19	1.04
🟦 ICBS	12.40	12.15	1.56	1.47
🟥 BEVALUE2	14.73	14.24	10.35	10.26
🟥 EDF	24.66	4.88	3.84	0.06
🟩 AED	24.77	5.94	4.91	0.14
🟦 FCFS	24.37	24.47	0.24	0.08
🟪 SJF	28.30	29.37	13.91	13.15
🟩 FREWARD	82.35	84.16	65.43	65.61

(a) Deadline Violation



	156	157	158	159
🟧 ICBSDH	1.09	1.09	1.09	1.09
🟦 ICBS	1.10	1.10	1.10	1.10
🟥 EDF	1.23	1.09	1.31	1.20
🟩 AED	1.29	1.08	1.29	1.29
🟦 FCFS	1.29	1.29	1.29	1.23
🟥 BEVALUE2	1.29	1.28	1.29	1.30
🟪 SJF	1.31	1.34	1.33	1.32
🟩 FREWARD	3.61	2.63	3.18	3.16

(b) SLA Cost

Figure 10: Performance of Scheduling Policies under Varying SLA Cost Function and Deadline, using short-long query mix (Q5 and Q6)

rency level, also known as multi-programming level (MPL) between 1 and 32, and observe the performance behavior of scheduling policies. We choose different arrival rates under different MPL, so that we achieve about 85% system load in all MPL cases.

Figure 9 shows the experiment results. For deadline performance, overall violation generally increases with higher MPL, as average query execution time increases from 0.340 msec (MPL=1) to 1.49 msec (MPL=32), while the deadlines remain unchanged. iCBS consistently performs well in terms of deadline violation under varying MPL. SJF also minimizes deadline violation by running short queries first and keeping the violation low with MPL equal to or less than 16, where deadline is rather far, compared to the execution time. With MPL=32, the deadline is relatively close and simple SJF performs worse than iCBS-DH that considers deadline in scheduling decisions as well. For cost performance, overall cost increases with higher MPL as well, while iCBS performs the best generally.

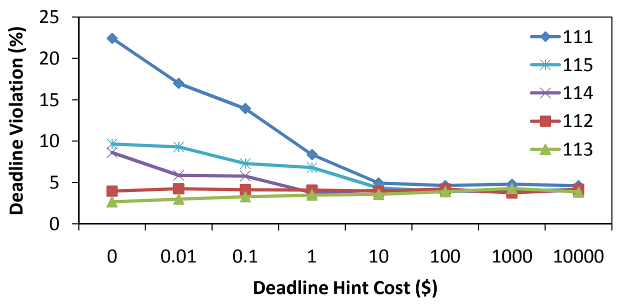
4.2.5 *Varying Query Mix*

In previous experiments, we have used Q1 through Q5, where execution times are in the range of 0.23 to 0.54 msec. This resembles the OLTP database workloads, where each query is short, and many such queries are run at high-throughput with high-MPL. In this subsection, we consider OLAP query workload, where some analytical queries may have long execution times (e.g. minutes to hours) while some are relatively short (e.g. seconds to minutes). We simulate such a query mix with varying execution time, using Q5 and Q6. These queries take 0.54 msec and 158 msec, and may be somewhat small scale, but their 1-to-300 execution time ratio is close to that of typical OLAP query mix. We create the mix using 99.62% of Q5 and 0.38% of Q6, so that each query type contribute about 50% of the total workload. We design additional DTH code and values for Q5 and Q6, as shown in Table 2, since Q6 takes 150 msec or more, making the response time of both Q5 and Q6 slower than before.

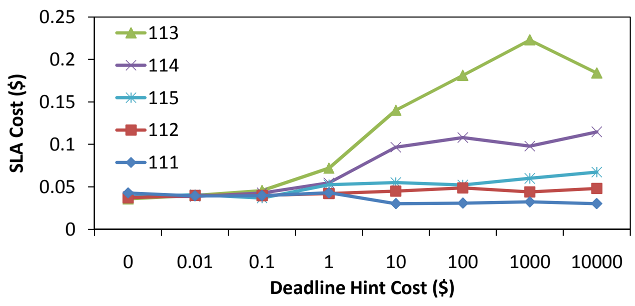
Figure 10 shows the results under DTH=15x, where HardDeadlineTime code varies between 6 and 9. iCBS-DH and iCBS gives good deadline performance in general. They are, however, outperformed by EDF and AED under the DTH=157 and 159: under these codes, long-running query Q6 has a cost step at 250 msec and a deadline at 500 msec. iCBS-DH and iCBS schedules some of Q6 for SLA cost reduction at around 250 msec, and causes increase in deadline violation, while EDF and AED do not do this, and just focus on 500 msec deadline, which is a relaxed time budget that leads to lower deadline violation overall. Cost performance displays similar results across various scheduling policies, except FirstReward that suffers from high scheduling overhead mentioned above.

4.2.6 *Varying Deadline Hint Cost*

Lastly, we study the performance behavior of iCBS-DH that we propose in this paper. We consider the impact of hint cost value that suggests how importantly the deadline should be regarded for scheduling decisions. If hint cost value is high, deadline observation will be more stressed than regular SLA cost steps. If hint cost value is low, the opposite will happen. Short queries are used, i.e. Q1 through Q5.



(a) Deadline Violation



(b) SLA Cost

Figure 11: Performance of iCBS-DH under varying Deadline Hint Cost

Figure 11 shows the experiment result. We vary hint cost value between \$0.01 and \$10000. We also show hint=0 case, where it works the same as the regular iCBS. For deadline performance, the high hint cost reduces violations and hint=\$10 or higher performs similarly. Hint=\$1 slightly increases the violation, while hint value less than that gives a big jump in violation. Hint=\$1 forms a critical point because regular cost steps in this experiment is in the range of \$0.69 and \$1.62. Note that some hint cost has different impacts on deadline performance with different SLA cost function shape and deadline: given DTH=113 and 112, deadline violation does not change much with low hint cost value, while the violation increases dramatically with low hint cost given 114,115, and 111. This is because under the DTH=112 and 113, deadline is the same as or later than the SLA cost step, so the iCBSh-DH's cost minimization based on iCBS keeps the deadline violation low even without strong deadline hint.

As expected, cost performance gets worse with higher hint cost value. With hint cost value \$0.1 or \$0.01, iCBS-DH performs very similar to iCBS, and hint=\$1 gives a slight rise in the cost. Hint of \$10 or higher makes the cost much higher compared to that of iCBS. Again, different DTH shows different curve with high hint cost. This is because, with DTH=113, emphasis on the deadline (i.e. 30 msec after arrival) means the less attention on the cost step (i.e. 20 msec after arrival), leading to high SLA cost. In other cases, e.g. DTH=111, 112, 115, emphasis on the deadline does not hurt the cost given the SLA cost function and deadline.

It seems that iCBS-DH creates an effective tradeoff between deadline violation and cost minimization with the hint value. The following is a general rule of thumb we observe from our experiments. In order to obtain low deadline violations, we suggest setting the hint cost value as the 10 to 100 times greater than the maximum cost step heights, or jumps. If one needs to keep both deadline and cost performance in balance, we suggest setting the hint value at the average cost step height, \$1 in our experiment above.

5. RELATED WORK

In addition to the previous works mentioned earlier, we discuss the following related work.

Cost-based Scheduling Recently, in the context of data warehouse, [13] has proposed a scheduling policy that considers multiple objectives of minimized slowdowns, fairness, and differentiation. It seems to be an effective scheduling policy for in-house infrastructure or private clouds where multiple users belong to a single party. However, it does not support SLAs, which are crucial in public cloud environment where multiple parties share the resources. Also, SLA cost-aware scheduling policies [17, 26] have been proposed recently. These works assume continuous SLAs and therefore have very high time complexity. In comparison, we use discrete SLAs, which are easier to express in plain English [34], and achieve logarithmic time complexity.

Constraint-conscious Scheduling [16, 32, 33] propose scheduling policies for both cost optimization and deadline enforcement. They focus on workflow scheduling, rather than individual job scheduling that we address in this paper. [12] uses a dual-queue approach with EDF (Earliest Deadline First) and SRPT (Shortest Remaining Processing Time). Their purpose, however, is different from ours in that they focus on tardiness minimization while we aim at both cost minimization *and* deadline enforcement.

[20] presents a scheduling algorithm for systems consisting of certifiable mixed-criticality sporadic tasks. In their work, a job is either non safety-critical or safety-critical, where the latter one has more conservative estimation of worst-case execution time (WCET). The algorithm is designed to assign an ordering among jobs in a greedy fashion in order to find a feasible schedule. The problem in [20] is fundamentally different from our problem: in their problem setting, there is no concept of utility; instead, they focus on finding a feasible schedule (the algorithm may fail to find such a schedule), assuming each job takes its worst-case execution time.

[11] treats the scheduling problem as a sequential decision problem in the form of Markov decision process, and solves the scheduling problem using reinforcement learning techniques. [29] uses similar ideas to address periodic workloads. Although these approaches can handle arbitrary utility functions as well as hard deadlines, these models are limited in that: there have to be a number of job classes (tasks), for each class the number of outstanding jobs cannot be too large, and jobs within the same class must share the same utility function.

6. CONCLUSIONS

In this paper, we presented workload scheduling under two different types of SLAs, soft and hard SLA. We proposed a deadline- and cost-aware scheduler called iCBS-DH. We also evaluated deadline and cost performance of various scheduling policies under a large range of SLA cost function and deadline types.

7. ACKNOWLEDGEMENT

We thank Michael Carey, Hector Garcia-Molina, and Jeffrey Naughton for the insightful discussions and comments.

8. REFERENCES

[1] Amazon Relational Database Service. http://aws.amazon.com/rds/.

[2] M. Brantner, D. Florescu, D. A. Graf, D. Kossmann, and T. Kraska. Building a database on s3. In *SIGMOD*, 2008.

[3] D. G. Campbell, G. Kakivaya, and N. Ellis. Extreme scale with full sql language support in microsoft sql azure. In *SIGMOD Conference*, pages 1021–1024, 2010.

[4] M. J. Carey, M. Livny, and H. Lu. Dynamic task allocation in a distributed database system. In *ICDCS*, 1985.

[5] Y. Chi, H. J. Moon, and H. Hacigumus. icbs: Incremental cost-based scheduling under piecewise linear slas. In *PVLDB*, 2011.

[6] Y. Chi, H. J. Moon, H. Hacigumus, and J. Tatemura. Sla-tree: A framework for efficiently supporting sla-based decisions in cloud computing. In *EDBT*, 2011.

[7] C. Curino, E. Jones, Y. Zhang, E. Wu, and S. Madden. Relational cloud: The case for a database service. *MIT CSAIL Technical Report 2010*.

[8] J. Duggan, U. Çetintemel, O. Papaemmanouil, and E. Upfal. Performance prediction for concurrent database workloads. In *SIGMOD Conference*, pages 337–348, 2011.

[9] D. Florescu and D. Kossmann. Rethinking cost and performance of database systems. *SIGMOD Record*, 38(1):43–48, 2009.

[10] A. Ganapathi, H. A. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. I. Jordan, and D. A. Patterson. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *ICDE*, pages 592–603, 2009.

[11] R. Glaubius, T. Tidwell, B. Sidoti, D. Pilla, J. Meden, C. Gill, and W. D. Smart. Scalable scheduling policy design for open soft real-time systems. *Real-Time and Embedded Technology and Applications Symposium, IEEE*, 0:237–246, 2010.

[12] S. Guirguis, M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, and K. Pruhs. Adaptive scheduling of web transactions. In *ICDE*, 2009.

[13] C. Gupta, A. Mehta, S. Wang, and U. Dayal. Fair, effective, efficient and differentiated scheduling in an enterprise data warehouse. In *EDBT*, 2009.

[14] H. Hacigumus, J. Tatemura, W.-P. Hsiung, H. J. Moon, O. Po, A. Sawires, Y. Chi, and H. Jafarpour. Clouddb: One size fits all revived. In *IEEE SERVICES*, 2010.

[15] J. R. Haritsa, M. Livny, and M. J. Carey. Earliest deadline scheduling for real-time database systems. In *IEEE RTSS*, 1991.

[16] D. Hong, T. Johnson, and S. Chakravarthy. Real-time transaction scheduling: a cost conscious approach. In *SIGMOD*, 1993.

[17] D. E. Irwin, L. E. Grit, and J. S. Chase. Balancing risk and reward in a market-based task service. In *HPDC*, 2004.

[18] E. D. Jensen, C. D. Locke, and H. Tokuda. A time-driven scheduling model for real-time operating systems. In *IEEE Real-Time Systems Symposium*, 1985.

[19] D. Kossmann, T. Kraska, and S. Loesing. An evaluation of alternative architectures for transaction processing in the cloud. In *SIGMOD Conference*, pages 579–590, 2010.

[20] H. Li and S. Baruah. An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *Proceedings of the 2010 31st IEEE Real-Time Systems Symposium*, RTSS '10, pages 183–192, Washington, DC, USA, 2010. IEEE Computer Society.

[21] Z. Liu, M. S. Squillante, and J. L. Wolf. On maximizing service-level-agreement profits. In *ACM Conference on Electronic Commerce*, 2001.

[22] H. J. Moon, Y. Chi, and H. Hacigumus. Sla-aware profit optimization in cloud services via resource scheduling. In *IEEE SERVICES*, 2010.

[23] J. M. Peha and F. A. Tobagi. A cost-based scheduling algorithm to support integrated services. In *INFOCOM*, 1991.

[24] J. M. Peha and F. A. Tobagi. Cost-based scheduling and dropping algorithms to support integrated services. *IEEE Transactions on Communications*, 44(2):192–202, 1996.

[25] P. Pirzadeh, J. Tatemura, and H. Hacigumus. Performance evaluation of range queries in key value stores. In *DataCloud*, 2011.

[26] F. I. Popovici and J. Wilkes. Profitable services in an uncertain world. In *Supercomputing*, 2005.

[27] K. Ramamritham, S. H. Son, and L. C. DiPippo. Real-time databases and data services. *Real-Time Systems*, 28(2):179–215, 2004.

[28] B. Schroeder, A. Wierman, and M. Harchol-Balter. Closed versus open system models: a cautionary tale. In *NSDI*, 2006.

[29] T. Tidwell, R. Glaubius, C. D. Gill, and W. D. Smart. Optimizing expected time utility in cyber-physical systems schedulers. In *Proceedings of the 2010 31st IEEE Real-Time Systems Symposium*, RTSS '10, pages 193–201, Washington, DC, USA, 2010. IEEE Computer Society.

[30] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, and H. Hacigumus. Intelligent management of virtualized resources for database management systems in cloud environment. In *ICDE*, 2011.

[31] P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu, and H. Hacıgümüş. ActiveSLA: A profit-oriented admission control framework for database-as-a-service providers. In *SoCC*, 2011.

[32] J. Yu, R. Buyya, and C. K. Tham. Cost-based scheduling of scientific workflow applications on utility grids. In *International Conference on e-Science and Grid Computing*, July 2005.

[33] Y. Yuan, X. Li, Q. Wang, and X. Zhu. Deadline division-based heuristic for cost optimization in workflow scheduling. *Information Sciences*, 179(15):2562 – 2575, 2009.

[34] L. Zhang and D. Ardagna. Sla based profit optimization in autonomic computing systems. In *ICSOC*, 2004.