Quality is in the Eye of the Beholder: Towards User-Centric Web-Databases*

Huiming Qu Jie Xu Alexandros Labrinidis
Advanced Data Management Technologies Laboratory
Department of Computer Science, University of Pittsburgh
Pittsburgh, PA 15260, USA
{ huiming, xujie, labrinid} @cs.pitt.edu

ABSTRACT

The proliferation of database-driven web sites (or web-databases) has brought upon a plethora of applications where both Quality of Service (QoS) and Quality of Data (QoD) are of paramount importance to the end users. In our previous work, we have proposed *Quality Contracts*, a comprehensive framework for specifying multiple dimensions of QoS/QoD; we have also developed user-centric admission control and scheduling algorithms in web-databases, whose goal is to maximize overall system performance. In this work, we turn our attention to the user side of the equation. Specifically, we propose to demonstrate how the *adaptation* of Quality Contracts (QCs) by the users can lead to vastly different performance results, both from the user point of view (i.e., user satisfaction) and also from the system point of view. Towards this, we propose to structure our demo in the form of an interactive game, where participants will be playing the role of users continuously adapting their QCs over time, while "playing" against system-generated users, who follow predetermined QC adaptation policies. Finally, we also propose to illustrate the effect of different admission control and scheduling policies.

Categories and Subject Descriptors

H.3.5 [**Online Information Services**]: Web-based services; H.3.4 [**Systems and Software**]: User profiles and alert services, Performance evaluation (efficiency and effectiveness); H.2.4 [**Systems**]: Query processing, Transaction processing

General Terms

Algorithms, Performance, Design, Economics, Experimentation, Human Factors

Keywords

User-centric, web-databases, quality of service, quality of data, quality contracts, query processing, transaction processing

1. INTRODUCTION

Frustration over broken links on web sites from a few years ago is being replaced by frustration over uncharacteristically late response times and stale information served by database-driven web sites today. For example, a stock information web site has to return the users' request on stock prices fast and with fresh data, even when facing floods of workload challenges. To stay competitive, content providers need to empower users to specify their preferences for Quality of Service (QoS) and Quality of Data (QoD). Preference specification inherits the spirit from the utility functions in real-time systems and Service Level Agreements (SLAs) in Grid applications [2, 1]. Knowing user preferences can help the server guide resource allocation, especially during periods of high server load, thus providing graceful service degradation.

Towards this, we proposed *Quality Contracts* (QCs) [4], a framework based on the micro-economic paradigm [7] that provides an intuitive and powerful way for users to specify preferences for QoS and QoD. A QC essentially specifies how much virtual money the user is willing to pay to have his/her query executed (Figure 1). The actual money that the server will get in the end (i.e., the system profit) will depend on *how well* the query was executed.

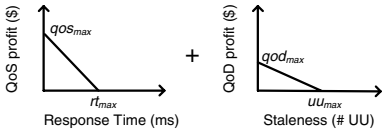


Figure 1: Sample Quality Contract

We have already developed admission control policies [6] and query/update scheduling [5] that consider such user preferences and maximize the overall system profit. In this work, we are focusing on the user side of the equation, by looking into user strategies to select appropriate Quality Contracts over time.

To illustrate the impact of user choices for QCs on query success ratio, we present a simple example where we fix the response time and freshness requirements, and only allow the user to modify the budget per query, Q_{max}. We ran an experiment with 6 users, each of which had 1000 queries and a fixed total budget of $20,000. As we see in Figure 2, slightly higher than the mean Q_{max} ($20) works the best, but as users become more aggressive, the overall success rate will diminish. This is because in a profit-driven Web-DB server, high

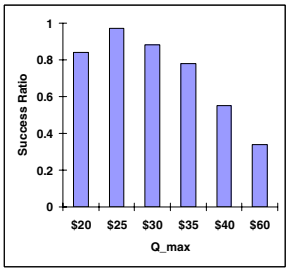


Figure 2: Success Ratio vs. Budget per Query Q_{max}

*Funded in part by NSF ITR award ANI-0325353.

SIGMOD'07, June 12–14, 2007, Beijing , China.
ACM 978-1-59593-686-8/07/0006.

Q_{max} naturally has more chance to obtain better performance, but having Q_{max} too high will also drain the users' budget more quickly. How users choose the QCs should be based not only on their own budget and workload, but also on the current environment. Our demo, QuiX, will provide a test bed as well as a playground for various users to try their own schemes.

Demo Outline We propose to demonstrate the following:

- a new user-interface for specifying different kinds of QCs,
- automated user-level strategies for adapting QCs over time,
- impact of QC adaptation strategies to user satisfaction and system profit,
- impact of admission control and scheduling strategies to user satisfaction and system profit.

To make the demo highly interactive and induce audience participation, we propose to structure it in the form of a *video-game*, where participants are asked to adapt QCs while "playing" against automated user-level strategies. At the same time, the demo will illustrate the system behavior over time and highlight the impact different choices have on user satisfaction and system profit.

2. CORE TECHNOLOGIES

The QuiX system consists of two parts: the user control module and the web-database server, as shown in Figure 3(b). In the presence of Quality Contracts (QCs), the users and the server have distinct objectives. Next, we elaborate on the distinct optimization problems from each point of view, as well as provide a brief explanation of the supporting algorithms.

2.1 User View

The User Control Module provides an interface for specifying QCs and visualizes execution of QC-enabled queries over time, while keeping track of the current budget each user has. The assumption is that each user is "paying" the server the full budget of the query at query submission time, but the server "refunds" back money because the query was not executed to the full level of service that the user specified.

User Objective: The presence of Quality Contracts serves as a guidance and an incentive for the server to return the queries according to users' preferences on QoS and QoD; at the same time, it leaves to the users the responsibility of choosing the QC appropriately. The ultimate question for the user is how to select Quality Contracts to get as many of his/her queries executed as possible within a certain budget.

We adopt QCs with linearly decreasing positive functions [5] as in Figure 1. For simplicity, we only allow users to modify the total budget for the entire query, Q_{max}. We define the *success ratio* to be the percentage of the returned queries. The user optimization problem is how to choose Q_{max} for each query (i.e., Q^i_{max}) in order to maximize the overall success ratio for all his/her queries.

User Algorithms: Assuming each user has a certain budget B and wants to execute N queries, we introduce five algorithms to choose Q_{max} in order to maximize the user success ratio.

- **Fixed (FIX) :** $Q^i_{max} = \frac{B}{N}$. FIX is a simple static policy that assumes that the budget decreases evenly, and gives each query equal amount of money. However, since QCs use linear functions (e.g., the actual money paid decreases when the response time increases), the accumulated unspent money (or refunds) from previous queries is ignored by FIX. FIX can be made aggressive by adding a certain percentage of over-bidding. However, it is not clear how to determine this percentage (as illustrated in Figure 2).
- **Random (RAN) :** $Q^i_{max} = \textit{uniform}[1,\ 2 * \frac{B}{N} - 1]$. RAN uses $\frac{B}{N}$ as the mean to pick Q_{max} uniformly. Similar to FIX, it also ignores the previous refunds.
- **Adaptive (AD) :** $Q^i_{max} = \frac{B^i}{N^i}$. AD keeps watching the current budget B^i and the number of queries left N^i before query i is issued. However, AD does not consider the competitors and decides based only on its own budget.
- **AdaptiveRandom (ADR) :** $Q^i_{max} = \textit{uniform}[1,\ 2 * \frac{B^i}{N^i} - 1]$. ADR uses $\frac{B^i}{N^i}$ as the mean to pick Q_{max} uniformly. Similarly to AD, it still ignores the surrounding environment.
- **Hybrid (HYB) :** HYB considers both the remaining budget and competition. It saves money by decreasing Q_{max} gradually when there are no "tough" competitors (indicated by a consecutive successful history), and uses the savings by increasing Q_{max} to $\frac{B^i}{N^i}$ when encountering "tough" competitors (indicated by failures).

2.2 Server View

The server is responsible for processing both updates and queries in order to match the service requirements specified in the QCs attached to queries. The server combines our previous work on admission control [6] and on scheduling queries and updates [5].

Server Objective: The server optimization problem is how to handle admission control and scheduling to maximize the server's profit from user-submitted queries (with QCs).

Scheduling Scheme: We use QUTS [5] to schedule the queries and updates. QUTS is a meta-scheduling scheme: it keeps a separate query priority queue and a separate update priority queue (each with its own scheduling mechanism), and decides on the priority between the two queues according to the submitted QCs.

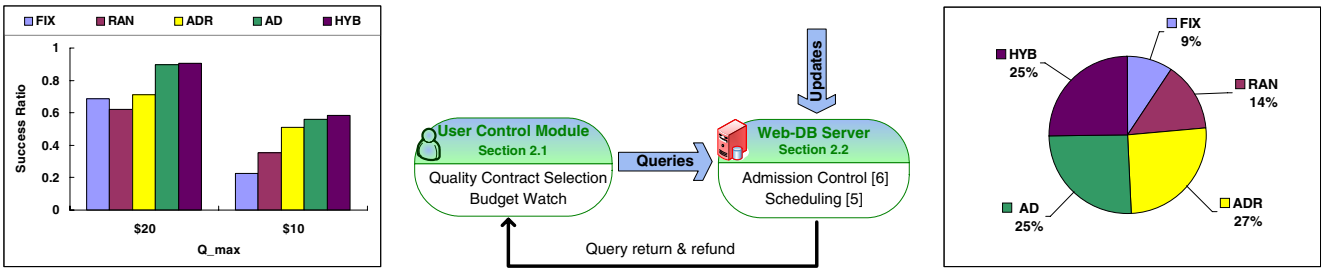
Admission Control Scheme: UNIT [6] was originally designed to be used with non profit-driven scheduling schemes. UNIT eliminates those updates that have the least interests from queries and rejects those queries that threaten the overall user satisfaction. Given QUTS scheduling, which is profit-driven, we do not need shedding capabilities for updates, since this is automatically achieved with QUTS postponing those least interesting updates. However, we still need to keep the query admission control component, because the QCs could have negative values (penalties) and the server needs a "shield" to prevent unnecessary losses.

2.3 Sample Results

We test the different user algorithms and show results from both the user's point of view (Figure 3(a)) and the server's point of view (Figure 3(c)). We have two classes of users for each experiment, with the same total budget. Class \$20 has a mean Q_{max} of \$20 and 4000 queries; class \$10 has a mean Q_{max} of \$10 and 8000 queries.

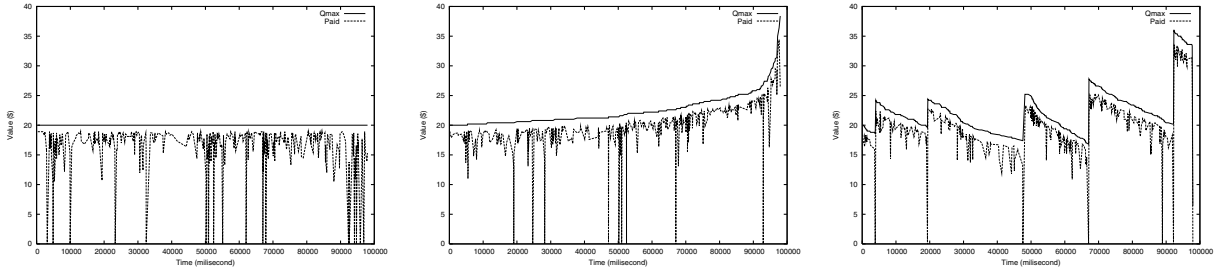
Figure 3(a) shows the query success ratio for the different classes of users. For each algorithm, class \$20 has higher success ratio than class \$10, whereas among different algorithms, Hybrid gives the best performance. Figure 3(c) summarizes the profit gained by the server from each user algorithm in class \$20. Notice that server gains much more profit from algorithms which consider the budget left than those do not.

Figure 4 shows Q_{max} and the actual payment over time for algorithms Fixed, Adaptive, and Hybrid. Fixed has the most failures (shown by actual payment of 0); Adaptive increases Q_{max} because it utilizes earlier refunds; Hybrid saves money actively and is able to rebound to a higher Q_{max} than Adaptive when failures are detected, thus achieving the fewest failures.



(a) User View: Success Ratio (b) System Architecture (c) Server View: Profit

Figure 3: System Architecture and Sample Results



(a) Fixed Strategy (b) Adaptive Strategy (c) Hybrid Strategy

Figure 4: Budget vs. Money Paid per Query over Time

3. DEMONSTRATION HIGHLIGHTS

We named our framework *QuiX*, which is short for QUality-aware Integrated admission Control and Scheduling.

QuiX: the video-game We propose to illustrate QuiX in the form of a video-game, to make the demo highly interactive and induce audience participation. We will allow up to two users to "play" concurrently, and also against system-defined opponents. The concept is similar to that of a flight simulator, where different environment scenarios will be illustrated and users need to adapt accordingly. The goal of the game is to maximize user satisfaction (i.e., query success ratio) given a certain overall budget and query workload. Towards this, users will be able to modify the properties of the submitted QCs themselves or choose a suggestion from one of the QC adaptation "wizards". Whoever adapts best to the environment changes will eventually win.

Visualization QuiX will provide per-user and aggregate statistics. Users need to monitor the behavior of queries over time (in a plot much like the ones in Figure 4) and also keep track of the remaining budget. Additionally, aggregate statistics will be available, showing the user-view across all users and also the system view (like the ones in Figure 3(a) and (c)).

QC Adaptation One simple way to adapt QCs is to change the Q_{max} value (i.e., budget) for each query. Another option is to change the *shape* of the QCs, by utilizing different alternatives such as linear function, step function, and the combination of the two. We plan to support all of these alternatives with an intuitive user interface.

Algorithms We expect to illustrate the different strategies for adapting QCs from the user perspective, as indicated by our sample results in Figure 3 and Figure 4. We also plan to show the effect of different admission control [6] and query/update scheduling strategies [5] both on the user satisfaction and the overall system profit.

4. CONCLUSIONS

We are advocating the use of Quality Contracts as a unified framework to support user QoS/QoD preferences for web-databases. Towards this, we built QuiX, our framework for QUality-aware Integrated admission Control and Scheduling. We propose to demonstrate QuiX in the form of a video-game, where participants will be asked to adapt QCs, "playing" against other users and automated QC adaptation strategies. This will allow us to evaluate the impact of QC adaptation, admission control and scheduling strategies on user satisfaction and system profit. As part of our future work, we plan to extend QuiX to consider Quality of Information metrics (e.g., [3]), and also consider the issue of content provider selection preliminary work in [8]).

5. REFERENCES

[1] A. AuYoung, L. Grit, J. Wiener, and J. Wilkes. Service contracts and aggregate utility functions. In *HPDC*, 2006.

[2] R. Buyya, D. Abramson, and S. Venugopal. The Grid Economy. *Proceedings of the IEEE*, 93(3):698–714, 2005.

[3] B. T. Dai, N. Koudas, B. C. Ooi, D. Srivastava, and S. Venkatasubramanian. Column heterogeneity as a measure of data quality. In *First Intl. VLDB Workshop on Clean Databases*, 2006.

[4] A. Labrinidis, H. Qu, and J. Xu. Quality contracts for real-time enterprises. In *Proc. of First Intl. Workshop on Business Intelligence for the Real Time Enterprise (BIRTE)*, 2006.

[5] H. Qu and A. Labrinidis. Preference-aware query and update scheduling in web-databases. In *ICDE*, 2007.

[6] H. Qu, A. Labrinidis, and D. Mosse. Unit: User-centric transaction management in web-database systems. In *ICDE*, 2006.

[7] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. "Mariposa: a wide-area distributed database system". *The VLDB Journal*, 5(1):048–063, 1996.

[8] J. Xu and A. Labrinidis. Replication-aware query processing in large-scale distributed information systems. In *WebDB*, 2006.