Achieving class-based QoS for transactional workloads

Bianca Schroeder Mor Harchol-Balter*
Carnegie Mellon University
Department of Computer Science
Pittsburgh, PA USA
<bianca, harchol>@cs.cmu.edu

Arun Iyengar Erich Nahum
IBM T.J. Watson Research Center
Yorktown Heights, NY USA
<aruni,nahum>@us.ibm.com

1. Introduction

Transaction processing systems lie at the core of modern e-commerce applications such as on-line retail stores, banks and airline reservation systems. The economic success of these applications depends on the ability to achieve high user satisfaction, since a single mouse-click is all that it takes a frustrated user to switch to a competitor. Given that system resources are limited and demands are varying, it is difficult to provide optimal performance to *all* users at all times. However, often transactions can be divided into different *classes* based on how important they are to the online retailer. For example, transactions initiated by a "big spending" client are more important than transactions from a client that only browses the site. A natural goal then is to ensure short delays for the class of important transactions, while for the less important transactions longer delays are acceptable.

It is in the financial interest of an online retailer to be able to ensure that certain classes of transactions (financially lucrative ones) are completed within some *target mean response time*. It is also financially desirable for the online retailer to be able to offer a Service Level Agreement (SLA) to certain customers, guaranteeing them some target mean response time that they desire (with possible deteriorated performance for customers without SLAs). This paper proposes and implements algorithms for providing such performance targets on a per-class basis.

A guaranteed mean response time for some class of transactions is one form of a *Quality of Service (QoS) target*. In many situations it is useful to provide more general QoS targets such as *percentile targets*, where $x\%$ of response times for a class are guaranteed to be below some value y. Percentile targets are often demanded by clients as part of a Service Level Agreement (SLA), for example to ensure that at least 90% of the client's transactions see a response time below a specified threshold. In addition to per-class response time and percentile targets, another common QoS target is to provide low *variability* in response times. The reason is that users may judge a relatively fast service still unacceptable unless it is also predictable [1, 4, 10].

Because the dominant time associated with serving an e-commerce transaction is often the time spent at the back-end database (rather than the front-end web/app server), it is important that the QoS be applied to the backend database system to control the time spent there. Yet, commercial database management systems (DBMS) do not provide effective service differentiation between different classes of transactions.

In designing a framework for providing class-based QoS targets one strives for the following high-level design goals:

Diverse per-class QoS target metrics The system should allow for an arbitrary number of different classes, where the classes can differ in their arrival rates, transaction types, etc. Each class is associated with one or more QoS targets for (per-class) mean response time, percentiles of response time, variability in response time, best effort, or any combination thereof.

Portability and ease of implementation Ideally the system should be portable across DBMS, and easy to implement.

Self-tuning and adaptive The system should ideally have few parameters, all of which are determined by the system, as a function of the QoS targets, without intervention of the database administrator. The system should also automatically adapt to changes in the workloads and QoS targets.

Effective across workloads Database workloads are diverse with respect to their resource utilization characteristics (CPU, I/O, etc.). We aim for a solution which is effective across a large range of workloads.

No sacrifice in throughput & overall mean response time Achieving per-class targets should not come at the

*Supported by NSF grants CCR-0133077, CCR-0311383, 0313148, and a 2005 Pittsburgh Digital Greenhouse Grant.

cost of an increase in the overall (over all classes) mean response time or a drop in overall throughput.

With respect to the above design goals, the prior work is limited. Commercial DBMS provide tools to assign priorities to transactions; however these are not associated with any specific response time targets. Research on real-time databases does not consider mean per-class response time goals, but rather looks only at how an individual transaction can be made to either meet a deadline or be dropped (in our system, transactions are not dropped). The only existing work on per-class mean response time guarantees for databases is based on modified buffer pool management algorithms [2, 3, 5, 7]. These techniques are not effective across workloads, since they focus only on one resource; For example, tuning the buffer pool will have little effect on CPU-bound or lock-bound workloads. Moreover, they don't cover more diverse QoS goals such as percentile or variability goals. A major limitation of all the above approaches is that they rely on changes to DBMS internals. Their implementation depends on complex DBMS specifics and is neither simple nor portable across different systems.

2 The EQMS

Our approach aims at achieving the above high-level design goals through an external frontend scheduler called the *EQMS (External Queue Management System)*.

The core idea behind the EQMS is to maintain an upper limit on the number of transactions executing simultaneously within the DBMS called the Multi-Programming Limit, or "MPL" (see illustration in Figure 1). If a transaction arrives and finds MPL number of transactions already in the DBMS, the arriving transaction is held back in an external queue. Response time for a transaction includes both waiting time in the external queue (queueing time) and time spent within the DBMS (execution time).

The immediately apparent attribute of our approach is that it lends itself to portability and ease of implementation as there is no dependence on DBMS internals. Also moving the scheduling outside the DBMS, rather than scheduling individual DBMS resources (such as the bufferpool or lock queues), makes it effective across different workloads, independent of the resource utilization.

With respect to obtaining diverse QoS targets, the core idea is that by maintaining a low MPL, we obtain a better estimate of a transaction's execution time within the DBMS, and hence we are able to maintain accurate estimates of the per-class mean execution times. This in turn gives us an upper bound on the queueing time for a transaction, which can be used by the scheduler in order to ensure that QoS targets are met. The actual algorithms that we use are more complex and rely on queueing analysis in order to meet a more diverse set of QoS targets, and behave in an adaptive manner.

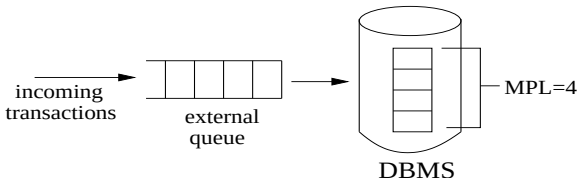

Figure 1. *Simplified view of mechanism used to achieve QoS targets. A fixed limited number of transactions (MPL=4) are allowed into the DBMS simultaneously. The remaining transactions are held in an unlimited external queue. Response time is the time from when a transaction arrives until it completes, including queueing time.*

The external scheduler achieves class differentiation by providing short queueing times for classes with very stringent QoS targets, at the expense of longer queueing times for classes with more relaxed QoS targets. There are no transactions dropped. One inherent difficulty in this approach is that not every set of targets is feasible, e.g., not every class can be guaranteed a really low response time. An external scheduler therefore also needs to include methods for determining whether a set of QoS targets is feasible.

The effectiveness of external scheduling and whether it requires sacrifices in overall performance (e.g. throughput or mean response time) depends on the choice of the MPL. For scheduling to be most effective a very low MPL is desirable, since then at any time only a small number of transactions will be executing inside the DBMS (outside the control of the external scheduler), while a large number are queued under the control of the external scheduler. On the other hand, too low an MPL can hurt the overall performance of the DBMS, e.g., by underutilizing the DBMS resources resulting in a drop in system throughput. Therefore, another core problem an external scheduler needs to solve is that of choosing the MPL.

The EQMS presents a unified external scheduling framework that addresses all of the above problems. Figure 2 gives an overview of the EQMS architecture. The EQMS takes as input a set of classes with one or several QoS targets for each class. These are specified by the online retailer and are not part of the EQMS. A QoS target can be a target on the mean response time, the x-th percentile of response time, variability in response time or a combination thereof. The core component of the EQMS is the *Scheduler* which decides on the order in which transactions are dispatched to the DBMS such that the associated QoS targets are met. The scheduler relies on the *MPL Advisor* to determine an MPL that provides sufficient scheduling control, while keeping

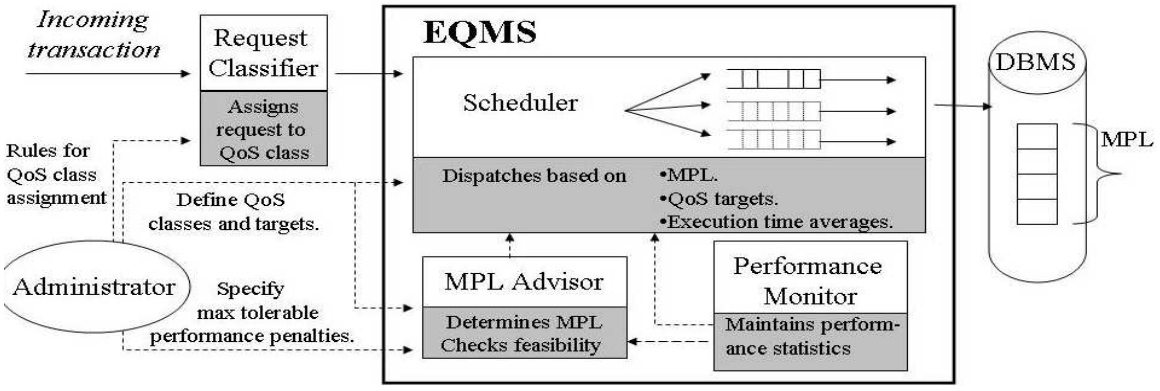

Figure 2. *Overview of the EQMS system.*

performance penalties, such as loss in throughput, below a threshold defined by the DBA (database administrator). The MPL Advisor also checks for the feasibility of a given set of targets.

An important feature of the EQMS is that its two main components, the Scheduler and the MPL Advisor, operate in a self-tuning and adaptive fashion making the EQMS robust to dynamic changes in the system load or workload characteristics. For example, the MPL Advisor combines feedback control (based on information collected by the *Performance Monitor*) with queueing theory to automatically tune the MPL parameter and update it when necessary.

A detailed description of the EQMS is presented in [6].

3 Summary of results

We experimented with IBM DB2 and PostgreSQL under a range of workloads, including CPU-bound, I/O-bound, and high vs. low lock contention workloads, based on different configurations of TPC-C [8] and TPC-W [9]. We created between 2 and 5 different transaction classes. Each class was assigned a mean response time target or a target on the x-th percentile of response time. In addition, we tried to minimize the variability of response time for each class as much as possible. Lastly, we evaluated the self-tuning features of the EQMS by varying the overall system load.

In short, we find that per-class mean response time targets are typically met with an error of less than 5%. Furthermore, the percentile targets are usually met with an error of less than 1%. We also find that the EQMS is able to reduce the squared coefficient of variation (variability) for each class from 2.3 to 0.11 for TPC-C type workloads, and from 15 to 0.19 for TPC-W type workloads. Lastly, we find that the EQMS responds quickly to changes in load.

Moreover, we observe that the EQMS works well across all workloads studied. The reason is that the core idea of limiting the MPL reduces contention within the DBMS at the bottleneck resource, independently of what the particular bottleneck resource is.

A detailed description of the results is presented in [6].

References

[1] A. Bouch and M. Sasse. It ain't what you charge it's the way that you do it: A user perspective of network QoS and pricing. In *Proceedings of IM'99*, 1999.

[2] K. P. Brown, M. J. Carey, and M. Livny. Managing memory to meet multiclass workload response time goals. In *Proceedings of Very Large Database Conference*, pages 328–341, 1993.

[3] K. P. Brown, M. J. Carey, and M. Livny. Goal-oriented buffer management revisited. In *Proceedings of the 1994 ACM SIGMOD Conference on Management of Data*, pages 353–346, 1996.

[4] B. Dellart. How tolerable is delay? Consumers evaluation of internet web sites after waiting. *Journal of Interactive Marketing*, 13:41–54, 1999.

[5] K. D. Kang, S. H. Son, and J. A. Stankovic. Service differentiation in real-time main memory databases. In *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '02)*, 2002.

[6] B. Schroeder, M. Harchol-Balter, A. Iyengar, and E. Nahum. Achieving class-based QoS for transactional workloads. Technical Report CMU-CS-05-186, Carnegie Mellon University, 11, 2005.

[7] M. Sinnwell and A. Koenig. Managing distributed memory to meet multiclass workload response time goals. In *15th IEEE Conference on Data Engineering (ICDE'99)*, 1997.

[8] Transaction Processing Performance Council. TPC benchmark C. Number Revision 5.1.0, December 2002.

[9] Transaction Processing Performance Council. TPC benchmark W (web commerce). Number Revision 1.8, February 2002.

[10] M. Zhou and L. Zhou. How does waiting duration information influcence customers' reactions to waiting for services. *Journal of Applied Social Psychology*, 26:1702–1717, 1996.