3 Achieving QoS for Database Systems (DBMS)

We now turn our attention to Web servers serving *dynamic* rather than static content. Here response times can be an order of magnitude slower than in the case of static content, due to delays incurred at the backend database.

The problem Consider an online store with an inventory management system, whose customers incur these delays every time they shop for an item. Given that a small fraction of customers, the 10% "biggest spenders", contribute more than half the total store revenue [5, 8], it is plausible that one could significantly reduce delays for the few big spenders, without significantly penalizing the remaining 90% of the customers, by giving *high priority* to big spenders. The goal of our research is to provide prioritization and differentiated performance classes within a traditional (general-purpose) relational database system (we use IBM DB2[11], Shore[15], and PostgreSQL[16]) running representative benchmark workloads for online transaction processing: TPC-C [20] and TPC-W [21].

Related prior work While the above problem sounds very natural, implementations of class-based prioritization in general-purpose traditional DBMS do not exist. While both IBM and Oracle claim to provide such prioritization tools (IBM DB2gov and QueryPatroller [11, 6] and Oracle DRM [17]), all of these are limited to CPU scheduling, and prove largely ineffective in our experiments. On the research front, the work on class-based prioritization for general-purpose DBMS is simulation-based, e.g., [4, 3]. There is a large body of literature on real-time database systems (RTDBMS) [1, 2, 10], which differ from the general-purpose DBMS we study in implementation and goals.

Implementation results In [13] we investigate the bottleneck resource for IBM DB2 and Shore under the TPC-C workload and find that transactions spend more time *waiting in lock queues* than they do waiting for or using CPU or I/O. Figure 1(left) shows the resource breakdown of a transaction's life for the Shore DBMS. The graph is almost identical for IBM DB2. The long lock wait times stem from the fact that a database transaction can hold a lock resource, while *simultaneously* waiting in another lock queue on a different resource. This result motivates us to implement scheduling algorithms for *prioritizing the lock queues*, rather than prioritizing CPU, as was tried in prior work above. After a detailed statistical characterization of what causes large waiting times, we propose and implement a new preemptive lock-scheduling algorithm called POW *(Preempt on Wait)* [14] which only preempts those transactions that are simultaneously holding one lock while waiting on a second lock. As shown in Figure 1(right), for low think time (high load), POW improves high-priority response times by almost a *factor of ten*, while low-priority transactions are only penalized by 10%.

Where systems meets theory Our implementation in [14] involves scheduling lock queues *internal* to the DBMS. In [18, 19], we ask, "*Wouldn't it be great to achieve this same performance differentiation without ever touching the DBMS?*" Our solution is to install a front-end box which limits the number of transactions simultaneously in the DBMS, by temporarily delaying low-priority transactions in an *external queue*, and allowing high-priority transactions through, thus reducing the lock queue contention that high-priority transactions experience. Queueing analysis is used in dynamically deriving the proper level of multiprogramming (the number of transactions to allow in) [19]. Surprisingly, we find that *external scheduling is as effective as internal scheduling*. The advantages of external scheduling are simplicity and portability across any DBMS, plus the

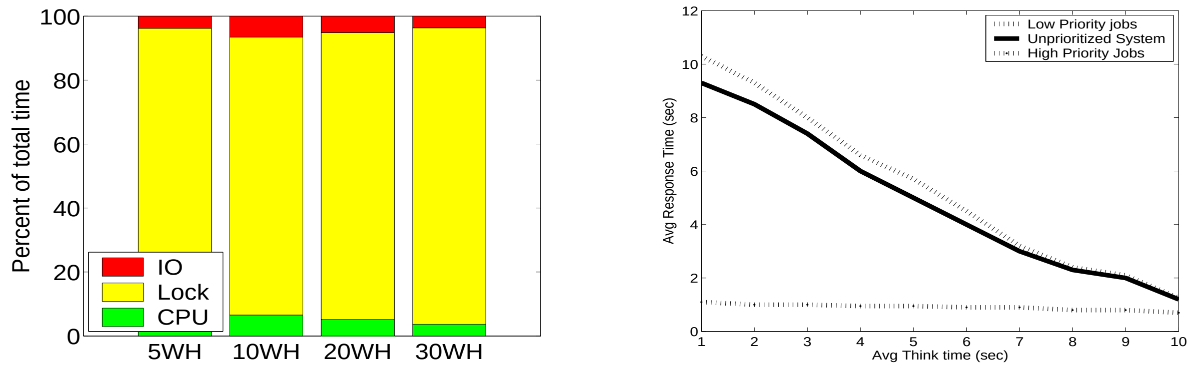

Figure 1: *Results for Shore with TPC-C: (Left) A transaction's life is dominated on average by lock wait times. (Right) POW lock prioritization improves high-priority response times dramatically under high load (low think time).*

fact that many different complex types of QoS are achievable via external scheduling, including variance and tail-delay guarantees [18].

Impact IBM has been very interested in productizing our work on externally-provided QoS guarantees for DBMS, and has filed a joint patent on this work with CMU. Several researchers have followed up on this work. Some apply similar ideas to web requests [24], or propose to improve upon our solution through the use of query progress indicators [12], or by making the scheduling more adaptive [22, 23]. Others have provided lock scheduling policies with theoretical guarantees [7] or I/O scheduling that can be used in combination with our priority mechanisms [9].

Funding This research was supported by (i) Pittsburgh Digital Greenhouse grant "QoS for Online Shopping" (2003-2005); (ii) Pittsburgh Digital Greenhouse grant "External QoS Management System for Backend Database Servers" (2005-2006); (iii) IBM graduate student fellowship.

References

[1] R. K. Abbott and H. Garcia-Molina. Scheduling real-time transactions: A performance evaluation. In *Proceedings of Very Large Database Conference*, pages 1–12, 1988.

[2] R. K. Abbott and H. Garcia-Molina. Scheduling real-time transactions: A performance evaluation. *ACM Transactions on Database Systems*, 17(3):513 – 560, 1992.

[3] 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.

[4] M. J. Carey, R. Jauhari, and M. Livny. Priority in DBMS resource scheduling. In *Proceedings of Very Large Database Conference*, pages 397–410, 1989.

[5] D. Champernowne. A model of income distribution. *Economic Journal*, 63:318–351, 1953.

[6] I. Corporation. IBM DB2 query patroller administration guide version 7.0, 2000.

[7] R. Guerraoui, M. Herlihy, and B. Pochon. Toward a theory of transactional contention managers. In *Proceedings of ACM PODC*, 2005.

[8] C. D. Guilmi, E. Gaffeo, and M. Gallegatti. Power law scaling in the world income distribution. *Economics Bulletin*, 15(6):1–7, 2003.

[9] C. Hall and P. Bonnet. Getting priorities straight: improving linux support for database I/O. In *Proceedings of VLDB*, 2005.

[10] J. Huang, J. Stankovic, K. Ramamritham, and D. F. Towsley. On using priority inheritance in real-time databases. In *IEEE Real-Time Systems Symposium*, pages 210–221, 1991.

[11] I. T. Lab. IBM DB2 universal database administration guide version 5. Document Number S10J-8157-00, 1997.

[12] G. Luo and J. Naughton. Multi-query SQL progress indicators. In *Proc. of EDBT'06*, 2006.

[13] D. McWherter, B. Schroeder, N. Ailamaki, and M. Harchol-Balter. Priority mechanisms for OLTP and transactional web applications. In *Proceedings of the 20th International Conference on Data Engineering*, Boston, MA, April 2004.

[14] D. McWherter, B. Schroeder, N. Ailamaki, and M. Harchol-Balter. Improving preemptive prioritization via statistical characterization of OLTP locking. In *Proceedings of the 21st International Conference on Data Engineering*, San Francisco, CA, April 2005.

[15] U. of Wisconsin. Shore - a high-performance, scalable, persistent object repository. `http://www.cs.wisc.edu/shore/`.

[16] PostgreSQL. `http://www.postgresql.org`.

[17] A. Rhee, S. Chatterjee, and T. Lahiri. The Oracle database resource manager: Scheduling CPU resources at the application. High Performance Transaction Systems Workshop, 2001.

[18] B. Schroeder, M. Harchol-Balter, A. Iyengar, and E. Nahum. Achieving Class-Based QoS for Transactional Workloads. In *Proceedings of the 22nd International Conference on Data Engineering Poster Paper*, Atlanta, GA, April 2006.

[19] B. Schroeder, M. Harchol-Balter, A. Iyengar, E. Nahum, and A. Wierman. How to determine a good multi-programming level for external scheduling. In *Proceedings of the 22nd International Conference on Data Engineering*, Atlanta, GA, April 2006.

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

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

[22] Q. Zhang, A. Riska, E. Riedel, and E. Smirni. Bottlenecks and their performance implications in e-commerce systems. In *Proceedings of WCW '04*, 2004.

[23] Q. Zhang, A. Riska, W. Sun, E. Smirni, and G. Ciardo. Workload-aware load balancing for clustered web servers. *IEEE Transactions Parallel Distributed Systems*, 16(3), 2005.

[24] J. Zhou and T. Yang. Selective early request termination for busy internet services. In *Proc. of WWW'06*, 2006.