Adaptive Concurrency Control for Heterogeneous Workloads in Main Memory Databases

Fan Gao
University of Wisconsin - Madison
gaof@cs.wisc.edu

ABSTRACT

As memory capacity and CPU core numbers keep increasing, transactional in-memory database is becoming a very popular option in many scenarios. However, many currency control protocols failed to fully exploit these resources due to different bottlenecks caused by high contention. One partial reason is that they are designed and optimized for specific workloads. And recent studies show that adaptive concurrency control protocols have the potential to outperform a single protocol when dealing with heterogeneous and complicated workloads. We may adopt different concurrency protocols according to the information we collected from system monitoring and features we predicted from online machine learning models. We are also trying to explore the possibilities of new concurrency protocols adaptivity in different levels and potential optimization techniques. Adaptive concurrency control combined with workload forecasting will help develop autonomous database management systems.

PVLDB Reference Format:
Fan Gao. Adaptive Concurrency Control for Heterogeneous Workloads in Main Memory Databases. *PVLDB*, 12(N): 1234-5678, 2020.
DOI: https://doi.org/10.14778/url/

1. INTRODUCTION

The advancement of single-threaded CPU has been slowed down since 2010. Physical limitation of CPUs is reached and it's being more and more difficult for modern CPUs to have higher clock frequencies and single-threaded performance. As a alternative, architects are trying to replace previous single-core CPUs with multi-core CPUs, in which each core is smaller and low-power compared to previous single core. We are entering the era of multi-core chips and many researchers to trying to maximize the potential computation power of multi-core CPUs and increase the performance. Nowadays, it's no longer rare to have hundreds of CPU cores and hundreds of gigabytes of main memory integrated in a single modern server, which makes high-throughput main memory database management systems (DBMS) becoming more and more popular [20].

Low data transfer speed between disks and main memory is the major bottleneck for traditional databases. And main memory databases store all data in memory instead of disks. Without slow disk IO and complex buffer management between main memory and disks, the throughput of main memory databases can be significantly increased as long as the data can fit into the memory.

However, new bottleneck has raised due to competing accesses to data from hundreds of cores at the same time, which makes it very difficult to coordinate and avoid potential errors. People are trying to develop different concurrency control protocols to help increase transaction throughput of main memory databases which enforce some basic property (atomicity, consistency, isolation and durability, also known as ACID) at the same time.

Concurrency control is a fundamental and difficult topic in the database area and has been studied intensively for decades. A great number of papers focused on developing lightweight concurrency control protocols with low overhead [5, 11]. Some other studies considered more about minimizing data contention of transactions executed across multiple cores [16, 21].

However, many of these protocols are designed for specific workloads with fixed optimization under a particular scenario. Generally, Optimistic Concurrency Control (OCC) [6] performances better under low-contention workloads (e.g. read-intensive, low-skewed) while Two Phase Locking (2PL) [2] has higher overall throughput when the workload has higher higher contention (e.g. write-intensive, high-skewed). And protocols with partition-level locking usually perform better for partitionable workloads [5].

However, it's difficult for developers to know the workload patterns before actual deployment. And in case the workload shift dramatically, previous default concurrency control protocols may no longer maintain the advantage. For instance, H-Store's [5] great performance is based on the observation that each transaction is more likely access only one partition. And its overall throughput will decrease when the workload contains more cross-partitioned transactions [20]. It would be nice for the database itself to switch to different concurrency control protocols automatically without the interference of database administrators (DBAs).

The idea of self-tuning DBMS is introduced in 1970s. Since then, people are trying to develop tools that can give advice to DBAs on how to choose the best design like index [17, 4], partitioning [1] for a database. And this idea

Proceedings of the VLDB Endowment, Vol. 12, No. N
ISSN 2150-8097.
DOI: https://doi.org/10.14778/url/

is becoming a trend again in recent years called self-driving DBMS [9], specifying different levels of autonomous databases. For a fully autonomous DBMS, it shall be able to automatically choose the best action to optimize itself in the appropriate time and learn from the results from monitoring. In order to avoid sub-optimal actions, the database not only to consider the past and present characteristics of the workloads, but also need to forecast future workloads. Only in this way can the database be able to adaptive to shifting workloads and choose the best action accordingly. To achieve this goal, machine learning and reinforcement learning are powerful tools, especially in workload forecasting [8] and auto-tuning [7, 22].

Being able to adapt to heterogeneous workloads is the key to arm current self-driving databases with adaptive concurrency control. Several papers have studied mixed concurrency control protocols at different levels. Adaptive Concurrency Control (ACC) [15] is proposed to assign different cores with different concurrency control schemes to execute transactions. Its follow-up paper CormCC [14] provided a general mixed concurrency control framework which support switching protocols online with minimal overhead. Callas [19] and Tebaldi [13] introduced the idea of modular concurrency control, partitioning transactions into groups and chose optimal concurrency control protocols according to offline workload analysis. This extra locking mechanism may limit the scalability of the databases and become the performance bottleneck with high core counts. In addition, some paper focused more on helping optimize OCC itself adaptively. For example, MOCC [18] and HSync [12] tried to integrate 2PL-like methods into OCC to minimizing the cost of conflicts of accessing hot-spot data. AOCC [3] valued the impact of validation phase in OCC and incorporated different validation methods (LRV, GWV) and assign one of them to each transactions accordingly.

Above papers all trying to switch to different concurrency control protocols adaptively by analyzing workload characteristics and system resources. But few of them put the focus on future workloads. As stated above, to be fully autonomous, the database must be able to forecast future workloads and choose the optimal protocol adaptation. Combining adaptive concurrency control protocols with workload forecasting in a general framework as an external or internal agents [10] for the DBMS would be a possibly ideal solution.

The rest of the paper is main composed of two sections. Section 2 will analyzed one possible adaptive concurrency control mechanism. Section 3 will talk about how to forecast future workloads. Some similarity of these two parts including:

- Workload analysis: modeling workloads to help extract features and analyzing patterns to choose appropriate forecasting models.

- Clustering: cluster data to fit into different cores to assign different protocols and cluster queries into templates to reduce the cost of prediction.

2. ADAPTIVE CONCURRENCY CONTROL

2.1 Workload Modeling

In order to make the concurrency control adaptive, we need to build a model to help choose the optimal workload for a specific workload. The model shall engineer key features of the workloads, collect their information at runtime and build the model from existing workloads.

Many protocols are designed for workloads with specific characteristics. For example, OCC performs better when the workload is read-intensive and low-skewed, 2PL performs better when the workload is write-intensive and high-skewed, H-Store performs better when the workload is partitionable and each transaction is highly likely to access only one partition. Based on similar observations, we can summarize some potential key features that determine the characteristics of a specific workload.

- Read/write ratio: the ratio of read to write operations per transaction. Write-intensive workloads are more likely to have higher contention.

- Cross-cluster ratio: the cost of cross-cluster transactions decides the applicability of H-Store. As the number of cross-partition transaction increases, coarse-locking of H-Store will no longer hold the advantage and will degrade the throughput reversely.

- Transaction length: the average number of operations inside a transaction. The cost of abortion and restart of a transaction will increase significantly if the database need to redo large number of long transactions.

- Record contention probability: the possibility of concurrent transactions reading/writing the same record. Contention probability models the skew degree of transactions.

To do collect information of these features at runtime, one simple idea is to maintain a separate process and uses it to sample transactions and collect information in real time. Furthermore, the job of collecting information can be completed by either a centralized coordinator or pushed down to decentralized workers. For example, each worker can set a flag on the records accessed by the sampled operations, and then check the intersection of other workers' flags after the actual execution phase to estimate the record contention probability.

To build the final model, we can use different parameters (e.g. theta in Zipf's Law) to generate vast sets of workloads and train models like decision trees based on these workloads. It shall be better if we can make the model more robust by using workload forecasting to train incrementally. But now let's just leave it to Section 3 and skip it for a while.

2.2 Data Clustering

For chips with multiple cores, a natural idea would be partition both the data and CPU cores into clusters and assign one concurrency protocol for each cluster. Initially, the records is partitioned into N cluster, where N equals the number of CPU cores. Afterwards, we compute the utilization (percentage of transactions accessing the same cluster) and cross-cluster transaction ratio for each cluster. Then we can keep merging clusters with lowest utilization and most cross-cluster transaction until the threshold is met. In the end, we can assign each cluster with proportional CPU cores.

2.3 Mixing Protocols

After data clustering, we can now assign each cluster with the most appropriate concurrency control protocols. Naturally, there would be cases such that one record can be assessed by different protocols in a single transaction. This will introduce new overhead in coordinating conflicts. One solution is to enforce each protocol to execute all operations to a cluster in a single transaction. For instance, if a transaction assigned to a cluster running OCC trying to access a record in a cluster running 2PL, we will still use 2PL to access this record, as it's located in a cluster running 2PL.

To mix OCC, 2PL and H-Store, we can divide the general execution process into four phases, where each protocols may go through some of them:

Phases	Pre-process	Execution	Validation	Commit
2PL		√		√
OCC		√	√	√
H-Store	√	√		√

And we will use different protocols to access data in different clusters. In the pre-process phase, the mixed protocol will acquire the locks of all clusters running H-Store that need be accessed. Afterwards, if we need to access the record in a cluster running OCC, we will use the OCC logic (execution-validation-commit) to deal with that record; if we are trying to access a cluster running 2PL, we can use 2PL logic (execution-process) to deal with that record. In the final commit phase, we will releases partition-level locks for clusters running H-Store, record locks for clusters running 2PL and write locks for clusters running OCC. And that's the end of a complete transaction execution process.

2.4 Online Reconfiguration

To make the mixed concurrency control protocols more adaptive, one single cluster that already assigned a protocol may need to switch to another protocol. Naive protocol switching will require to stop all current transactions or even the restart of DBMS. To minimize the overhead of protocol switching, we can use mediated switching for online reconfiguration.

A mediated protocol is compatible with both the old and new protocol. In the execution process, a central coordinator will inform all workers that running old protocol switch to the mediated protocol asynchronously. After all workers finish executing transactions running the mediated protocol, then the coordinator will notify them to switch to the new protocol.

To make the mediated protocol compatible with both old and new protocol, the key is to enforce the execution phases of both old and new protocol at the same time. That is, the mediate protocol will go through the join set of execution phases of both old and new protocol. Take the example of mediate protocol from OCC to 2PL:

1. Executing phase: acquire the record lock (2PL), read/write the record value, and updated the corresponding timestamp and read/write sets (OCC)

2. Validation phase: locks all records in the write set and validate the read set (OCC)

3. Commit phase: commit values and release locks acquires by both 2PL and OCC

By enforcing the logic of both old and new protocol in the mediated process, we can enable a cluster switching to another protocol with minimal overhead.

3. FORECASTING WORKLOADS

3.1 Workload Patterns

Databases built for real world applications will never meet with static workloads. To make our workload forecasting model more robust, it's necessary to first dive into the common patterns of workloads in reality. Briefly speaking, workload forecasting is just a special case of time series analysis. Some common workload patterns include:

- Cycles: It's clear that vast and vast human activities display cycle patterns. The length of cycles also varies from single hour and day to multiple weeks and months. For example, for online book ordering, people are less likely to create a new order in working hours and busy workdays compared to relaxed evenings and weekends.

- Peaks: Peaks are also common in daily life. Procrastination makes people leave their work until the last minute. As the deadline is approaching, websites will receive higher and higher visits and will form a steep peaks (or valley). Evolution: Evolution describes the pattern of sudden shifting. New releases of mobile application, popular songs, etc. are all possible applications of these workload pattern.

3.2 Forecasting Models

To predict future time series, there are different models that are appropriate under certain kinds of scenarios. Common methods include:

- Linear Regression: Simplest linear model that has been widely used in many different areas. Its simplicity helps avoid over fitting and it's especially useful for short-term workload forecasting. But its simple architecture also make it unable to reveal complex and long-term patterns. In addition, LR's fast training speed is a huge advantage compared to KR and RNN.

- Kernel Regression: KR is the non-linear version of LR. It uses kernel function to help achieve the non-linearity. Inputs in the training are weighted, i.e., recent data will have higher priority while older data are more likely to be neglected.

- Recurrent Neural Network: Famous deep learning model that becomes very popular in recent years. Generally, it has the most powerful ability to reveal the patterns even in a black box. But its drawbacks are also very obvious: 1) great cost for training and inference. 2) Complex model makes it more like to over fit the data and significantly reduce the prediction accuracy.

Different forecasting models can better reveal the specific hidden pattern behind time series. LR performs better at revealing short-term patterns. KR is good at revealing long-term patterns. It's the only model that can predict yearly peaks. RNN outperforms other models when the workload has complex patterns, as long as we're careful to avoid over fitting.

A common technique that is widely used in machine learning competitions is called Ensemble. It calculates the predict results of multiple models. The combination of different models or models training on different data that reveals the same pattern will help reduce the variance and bias. In most cases it can help improve the prediction accuracy, but typically not to a large extent. Additionally, Ensemble also means the higher training and inference cost, so we need to trade off between accuracy and cost.

Forecasting future workloads is still an inaccurate description. Some studies choose to predict indirect future system resource demand for different workloads, while some other people prefer to predict using the query execution logic. Among them, QueryBot 5000 [8] is a splendid framework.

QueryBot 5000 predict future workloads based on logical composition of queries instead of indirect physical resources. Thus the model is not dependent on specific hardware and can better adapt to dynamical environment. The framework grouped SQL queries into different templates, and then user clustering to pick out most common templates (add up to cover more than 95% of the queries) that are representative for workload patterns. And the last step is to predict future workloads of the clustered most-common templates.

4. CONCLUSIONS

There are great number of concurrency control studies in recent years. But most of them can only display best performance under a specific workloads. It's hard to make one single concurrency control protocol fit with all dynamic workloads. Thus in this article we introduced the adaptive concurrency control. It can switch to different workloads according to the characteristics of shifting workloads. Combined with workload forecasting, adaptive concurrency control will become more robust and be a potential component of self-driving database management systems.

5. REFERENCES

[1] S. Agrawal, V. R. Narasayya, and B. Yang. Integrating vertical and horizontal partitioning into automated physical database design. In *SIGMOD '04*, 2004.

[2] P. A. Bernstein and N. Goodman. Concurrency control in distributed database systems. *ACM Comput. Surv.*, 13:185–221, 1981.

[3] J. Guo, P. Cai, J. Wang, W. Qian, and A. Zhou. Adaptive optimistic concurrency control for heterogeneous workloads. *Proc. VLDB Endow.*, 12:584–596, 2019.

[4] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index selection for olap. *Proceedings 13th International Conference on Data Engineering*, pages 208–219, 1997.

[5] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. B. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi. H-store: a high-performance, distributed main memory transaction processing system. *Proc. VLDB Endow.*, 1:1496–1499, 2008.

[6] H. T. Kung and J. T. Robinson. On optimistic methods for concurrency control. In *VLDB*, 1979.

[7] G. Li, X. Zhou, S. Li, and B. Gao. Qtune: A query-aware database tuning system with deep reinforcement learning. *Proc. VLDB Endow.*, 12:2118–2130, 2019.

[8] L. Ma, D. V. Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. J. Gordon. Query-based workload forecasting for self-driving database management systems. *Proceedings of the 2018 International Conference on Management of Data*, 2018.

[9] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. C. Mowry, M. Perron, I. Quah, S. Santurkar, A. Tomasic, S. Toor, D. V. Aken, Z. Wang, Y. Wu, R. Xian, and T. Zhang. Self-driving database management systems. In *CIDR*, 2017.

[10] A. Pavlo, M. Butrovich, A. Joshi, L. Ma, P. Menon, D. V. Aken, L. J. Lee, and R. Salakhutdinov. External vs. internal: An essay on machine learning agents for autonomous database management systems. *IEEE Data Eng. Bull.*, 42:32–46, 2019.

[11] Renkun, T. Alexander, and J. A. Daniel. Lightweight locking for main memory database systems. In *VLDB 2012*, 2012.

[12] Z. Shang, F. Li, J. X. Yu, Z. Zhang, and H. Cheng. Graph analytics through fine-grained parallelism. In *SIGMOD '16*, 2016.

[13] C. Su, N. Crooks, C. Ding, L. Alvisi, and C. Xie. Bringing modular concurrency control to the next level. *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017.

[14] D. Tang and A. J. Elmore. Toward coordination-free and reconfigurable mixed concurrency control. In *USENIX Annual Technical Conference*, 2018.

[15] D. Tang, H. Jiang, and A. J. Elmore. Adaptive concurrency control: Despite the looking glass, one concurrency control does not fit all. In *CIDR*, 2017.

[16] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden. Speedy transactions in multicore in-memory databases. In *SOSP '13*, 2013.

[17] G. Valentin, M. Zuliani, D. C. Zilio, G. M. Lohman, and A. Skelley. Db2 advisor: an optimizer smart enough to recommend its own indexes. *Proceedings of 16th International Conference on Data Engineering (Cat. No.00CB37073)*, pages 101–110, 2000.

[18] T. Wang and H. Kimura. Mostly-optimistic concurrency control for highly contended dynamic workloads on a thousand cores. *Proc. VLDB Endow.*, 10:49–60, 2016.

[19] C. Xie, C. Su, C. Littley, L. Alvisi, M. Kapritsos, and Y. Wang. High-performance acid via modular concurrency control. In *SOSP '15*, 2015.

[20] X. Yu, G. Bezerra, A. Pavlo, S. Devadas, and M. Stonebraker. Staring into the abyss: An evaluation of concurrency control with one thousand cores. *Proc. VLDB Endow.*, 8:209–220, 2014.

[21] X. Yu, A. Pavlo, D. Sánchez, and S. Devadas. Tictoc: Time traveling optimistic concurrency control. In *SIGMOD '16*, 2016.

[22] J. Zhang, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, J. Xing, Y. Wang, T. Cheng, L. Liu, M. Ran, and Z. Li. An end-to-end automatic cloud database tuning system using deep reinforcement learning. *Proceedings of the 2019 International Conference on Management of Data*, 2019.