Gato: Optimizing Multi-Version Deterministic Concurrency Control with Dynamic Partitioning and Split-on-Demand

Jiyeon Lim[1,a)] Haowen Li[2,b)] Hideyuki Kawashima[3,c)]

Abstract: In this paper, we introduce Gato, an innovative concurrency control protocol designed to tackle two significant challenges prevalent in multi-version deterministic systems, particularly those based on Bohm. The first challenge pertains to the constraints imposed by static segmentation during the concurrency control stage, while the second involves performance bottlenecks arising from read spinning in highly contended environments. Gato effectively addresses these issues through dynamic partitioning that alleviates imbalances and enhances throughput by redistributing workloads in real time. Additionally, Gato employs a split-on-demand strategy that reduces delays and optimizes transaction flow during the execution phase. These enhancements enable Gato to emerge as an efficient and scalable alternative in multi-version deterministic concurrency control, significantly lowering latency and enhancing scalability across a diverse range of highly contended workloads.

Keywords: Concurrency control, Multi-versioned database, Dynamic partitioning, Deterministic systems, High-contention

1. Background

Deterministic Concurrency Control aims to minimize conflicts between transactions by eliminating traditional locking and rollback mechanisms, instead of pre-setting the execution order of database tasks. Bohm [1] has introduced a multi-version deterministic concurrency control protocol that specifies the order of transactions during the concurrency control phase, allowing for execution in the execution phase without the need for complex mechanisms. However, Bohm's reliance on static partitioning during the concurrency control phase may result in load imbalances. Additionally, in the execution phase, high-contented scenarios can cause delays in read operations until write operations are completed. To address these issues, Gato incorporates dynamic partitioning and split-on-demand techniques in Caracal [2] to enhance performance in multi-version deterministic concurrency control.

2. Research Questions

Bohm employs a system known as multi-version deterministic concurrency control, which offers several significant advantages. Transaction processing within Bohm occurs in two phases. First, it establishes a fixed order for all transactions during the concurrency control phase. Next, in the execution phase, transactions are completed in this predetermined order. Consequently, Bohm functions in a predictable manner, as the transaction sequence remains consistent. However, Bohm faces two primary challenges when there is high contention for resources:

2.1 Static Partitioning in the Concurrency Control Phase

Bohm employs a static partitioning method where records are divided into fixed partitions and assigned to each thread during the concurrency control phase. This approach enables each thread to process transactions independently, as records are allocated to specific static partitions. Consequently, this minimizes data access conflicts between threads and eliminates the necessity for partition synchronization. However, a high volume of requests for particular records can create imbalances, potentially resulting in longer transaction processing times for the threads handling those records. This can lead to delays and performance bottlenecks, with other threads remaining idle until the responsible thread completes its tasks. Additionally, static partitioning complicates the system's ability to adjust load dynamically, which restricts overall scalability.

2.2 Read Spinning in the Execution Phase Due to Write Dependencies

Bohm is structured in the execution phase to prevent reads from locking writes. However, this design has several drawbacks. When a read operation depends on a specific record version that has not yet been written, it must enter a ”spinning” state and wait for the correct version to become available. While this read spinning is crucial for maintaining consistency, it can lead to

1 Faculty of Policy Management, Keio University, Japan
2 Graduate School of Media and Governance, Keio University, Japan
3 Faculty of Environment and Information Studies, Keio University, Japan
a) limjy19106@keio.jp
b) tony_li.haowen@keio.jp
c) river@sfc.keio.ac.jp

significant delays in high-contention environments where multiple transactions concurrently access and modify the same record. Additionally, it increases the processing time of subsequent transactions, ultimately reducing the overall throughput of the system.

Bohm's limitations highlight the necessity for a flexible control method capable of adapting to variable situations and reducing delays associated with read dependency. Addressing these challenges is crucial to ensure that contemporary data-intensive requirements are effectively met.

3. Contribution: Gato

Gato introduces two significant innovations: a method that employs dynamic partitioning during the concurrency control phase and the split-on-demand mechanism of Caracal [2] in the execution phase. These advancements effectively tackle the limitations of Bohm previously identified, positioning Gato as a more flexible and efficient solution in MVDCC. The contributions of Gato are as follows:

3.1 Dynamic Partitioning Mechanism in the CC Phase

Gato introduces dynamic partitioning to address the limitations of Bohm's static partitioning by actively monitoring the workload of partitions in real time and reallocating them as necessary. Each partition is managed in a hash table that records the frequency of access and load status for every record. This enables Gato to detect real-time scenarios where transaction tasks are concentrated in specific threads or where certain records are receiving excessive access. When a thread reaches an overload state, Gato reallocates some transaction tasks from the overloaded thread's partitions to those that are less congested, effectively preventing an overconcentration of particular records. For instance, when a thread receives a transaction operation, it distributes the record across other partitions to ensure a more balanced workload. This reallocation occurs in real-time, maintaining a balanced workload among threads and facilitating efficient resource utilization.

Figure 1 illustrates how Gato's dynamic partitioning mechanism operates. On the left, all transaction tasks are initially concentrated within the CC1 threads, resulting in an overload. On the right, some records from CC1 are reallocated to CC2 and CC3 as a result of Gato's dynamic partitioning, which adjusts the load to prevent concentration on specific threads. Consequently, Gato enhances throughput and reduces latency, offering stable and rapid performance even in high-contention environments. In this way, Gato overcomes the inefficiencies associated with static partitioning and delivers a scalable solution that adapts to varying conditions.

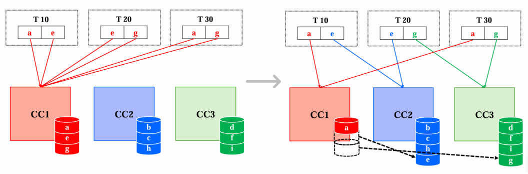

Fig. 1: Dynamic Partitioning in Gato with Intra-Transaction Parallelism for Load Balancing

3.2 Split-on-Demand Technique in the Execution Phase

During the execution phase of Bohm, a read spinning due to dependency on write operations may occur. This happens when a read operation is forced to wait until a write operation completes. The issue becomes more pronounced when multiple transactions attempt to access the same record simultaneously, resulting in delays and diminished system performance. To mitigate this, Gato introduced a technique called Caracal's split-on-demand, which effectively separates read and write operations. For instance, if transaction A is modifying a record while transactions B and C wish to read the same data, Gato can detect this conflict in real-time. In such cases, Gato permits the read operations that can be executed immediately to proceed without waiting for the write operation to finish, thereby reducing delays. Shared memory plays a vital role in this process, allowing Gato to monitor the status of each core in real time and redistribute transactions as necessary. Each core can swiftly access the latest information about a record—such as start and end times, access counts, and potential contention—through a hash table housed in shared memory. This capability aids in maintaining cache consistency, ensuring that each core has the most current data when accessing records. By employing split-on-demand techniques alongside shared memory, Gato significantly diminishes the latency associated with read spinning, resulting in enhanced overall performance. Consequently, we believe Gato outperforms Bohm by delivering faster and more reliable performance, even in high-demand environments, thus supporting efficient transaction processing.

4. Conclusion

Enhancements in Gato's design yield substantial performance improvements in MVDCC systems. The implementation of dynamic partitioning ensures a balanced distribution of transactions, effectively alleviating bottlenecks and boosting throughput, even in high-contention scenarios. Furthermore, split-on-demand techniques minimize transaction latency by reducing delays associated with read spinning. These features position Gato as a highly adaptable protocol suited for systems that demand both flexibility and high performance.

Acknowledgments This paper is based on results obtained from the project "Research and Development Project of the Enhanced Infrastructures for Post-5G Information and Communication Systems (JPNP20017) " and JPNP16007 commissioned by the New Energy and Industrial Technology Development Organization (NEDO), and from JSPS KAKENHI Grant Number 22H03596, and SECOM Science and Technology Foundation.

References

[1] J. M. Faleiro and D. J. Abadi, "Rethinking serializable multiversion concurrency control," *Proc. VLDB Endow.*, no. 11, p. 1190–1201, 2015.

[2] D. Qin, A. D. Brown, and A. Goel, "Caracal: Contention management with deterministic concurrency control," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. ACM, 2021, p. 180–194.

[3] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi, "Calvin: fast distributed transactions for partitioned database systems," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, p. 1–12.