細粒度予約制御とバッチ処理の並列化による Shirakami の高性能化

田辺 敬之[1]　川島 英之[2]

1. はじめに

長いバッチと短いオンラインのトランザクション（tx）を同時に流すと，既存の並行性制御（CC）ではバッチがほぼ確定しない設定がある（**図 1**）．Shirakami [1] はバッチが書く表に書き予約を置いて確定させるが，バッチが書かない行を読む短い tx までアボートさせる．本稿の Shirakami′ は，在庫補充のバッチを足した TPC-C [2] で，バッチの最中の注文を Shirakami の 1.26〜約 90 倍通した．その代わりにバッチが減ったのは，倉庫 8・同時に走るバッチ 4 件の 2 つの設定だけで，そこでも Shirakami の 0.72・0.84 倍にとどまる．

Shirakami は，長い tx の方式（文献 [1] の S-LTX．以下 LONG）と短い tx の方式（同 S-OCC．以下 SHORT）を直列化可能に共存させる．LONG は書くと宣言した表に確定まで書き予約を置き，Silo [3] 型の楽観的な CC である SHORT は，予約のある表の行を読んで予約より後に確定しようとするとアボートする．Shirakami は，バッチを 5 つの設定のどれでも，ロックが取れないとすぐアボートする二相ロック（2PL） [4] の 7.6 倍以上確定させた．代わりに，バッチが書かない在庫の行を読む注文もアボートし，倉庫 8・同時に走るバッチ 4 件・バッチ 1 件が 2 倉庫の設定では，注文は書き予約が無いときの見積もりの 0.6% しか通らない．

文献 [1] は，書く行が始めに決まらない tx のために予約の単位を表とし，1 つの LONG を 1 本のスレッドで進める使い方だけを論じる．TuskFlow [5] は長い tx を並列に進めるが，tx を一括で決まった順に流す．本稿の貢献は次の 2 つである．

- 書き予約の粒度の動的な制御：アプリの宣言に応じて書き予約を表より細かい単位に置き，約束があれば走査に沿って外していく．
- strand：1 つの LONG を複数のスレッドで進めながら，ほかの tx には 1 本の LONG に見せる API.

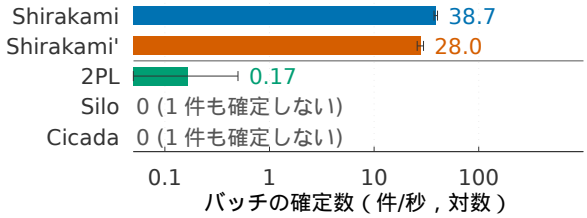


図 1　バッチ（補充）の確定数（倉庫 8・同時に走るバッチ 4 件・バッチ 1 件が 2 倉庫，対数目盛り）．Shirakami′ は同時に走るバッチが 1 件で Shirakami より少なく，2PL・Silo・Cicada はほぼ確定しない

2. Shirakami′

Shirakami′ は Shirakami に，次の 2 つの柱を足す．

書き予約の粒度の制御．アプリは LONG を始めるとき，表ごとに予約の単位を表・キーの範囲・書く行のキーのどれかで宣言できる．エンジンは宣言のうち最も細かい単位を使い，守れない宣言だけを表の単位に落とす．書く行を前もって言えないバッチなどでは，アプリが「走査で通り過ぎた行に後から書かない」と約束したときに限り，その行を予約から外す．

strand．strand は，1 つの LONG に複数のスレッドから読み・書き・走査を出す API である．分け方と本数はアプリが決める．予約・直列化の位置・読み書きの記録は tx に 1 つなので，ほかの tx からは 1 本の LONG に見える．倉庫 8・同時に走るバッチ 4 件・バッチ 1 件が 2 倉庫の設定では，同時に走るバッチを 4 件から 1 件に，予約の置かれる倉庫を 8 から 2 に減らす．エンジンは次の 3 つを守る．同じキーへの同時の操作は，どちらかの順に 1 本ずつ流したのと同じ結果になる．確定とアボートは各

[1] 慶應義塾大学 政策・メディア研究科
[2] 慶應義塾大学 環境情報学部

© 1959 Information Processing Society of Japan

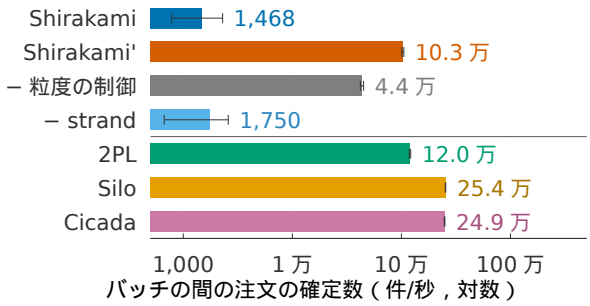


図 2 バッチの間の注文の確定数（図 1 と同じ設定，対数目盛り）．「−」は Shirakami′ からその部品を抜いた形，2PL・Silo・Cicada はバッチをほぼ確定しない

スレッドの操作の終わりを，アボートの片付けは全部のスレッドが抜けるのを待つ．この 3 つは，小さな規模のモデル検査で破れず，テストにも通った．

3. 評価

環境と負荷． 計算機は Intel Xeon Platinum 8468（1 ソケット, 物理 48 コア, SMT なし), DRAM 128 GB, Ubuntu 22.04.5（Linux 5.15.0）の 1 台である．どの系列もスレッドは 48 本，メモリ確保は mimalloc 2.1.7，エポックは 3 ミリ秒とした．各点は tx を 10 秒流し，3 回の平均を取った．バッチ 1 件は受け持つ倉庫の在庫を全部読み，しきい値を下回った品目を書き戻す LONG で，注文は同じ行を減らし，アボートすると出し直す．設定は，倉庫の数・同時に走るバッチの件数・バッチ 1 件が受け持つ倉庫の数を変えた 5 つである．比べる相手は 2PL，Silo [3]，Cicada [6] である．Shirakami′ は 2 節の 2 つの柱を組む．スレッドはどれも 48 本だが，Shirakami′ は注文のスレッドから 4〜7 本を strand に回す．

結果． Shirakami′ の注文は 5 つの設定のどれでも Shirakami の 1.26〜約 90 倍である．倉庫 8・同時に走るバッチ 4 件・バッチ 1 件が 2 倉庫の設定で部品を 1 つ抜くと（**図 2**），Shirakami′ の注文は strand を抜いた形の 59 倍，書き予約の粒度の制御を抜いた形の 2.35 倍で，2 つの柱は組むとどちらか 1 つより大きく伸びる．伸びの大半は strand と，それに回したスレッドによる．バッチが Shirakami より減ったのは倉庫 8・同時に走るバッチ 4 件の 2 つの設定だけで（0.72・0.84 倍），ほかの 3 つの設定では Shirakami と同じかそれより多かった．

4. まとめ

本稿は，Shirakami に書き予約の粒度の動的な制御と strand を足した Shirakami′ を示した．在庫補充のバッチを足した TPC-C で，倉庫 8・同時に走るバッチ 4 件の設定では，バッチが Shirakami より下がる（バッチ 1 件が 2 倉庫のとき 0.72 倍，1 倉庫のとき 0.84 倍）が，注文は Shirakami の約 70 倍・11.6 倍通る．ほかの 3 つの設定では，バッチを Shirakami と同じかそれより多く確定しながら，注文を Shirakami の 1.26〜約 90 倍通す．

謝辞 本論文は，NEDO（国立研究開発法人新エネルギー・産業技術総合開発機構）の委託事業「ポスト 5G 情報通信システム基盤強化研究開発事業」（JPNP20017）ならびに JPNP25014，JSPS 科研費 25H00446，JST CREST JPMJCR24R4，セコム科学技術財団，JST COI-NEXT SQAI（JPMJPF2221），JST ムーンショット型研究開発事業 JPMJMS2215 から得られた成果に基づいています．

参考文献

[1] Tanabe, T., Umegane, S., Arakawa, S., Kurosawa, R., Hoshino, T., Kawashima, H., Tanaka, M. and Kambayashi, T.: Shirakami: A Hybrid Concurrency Control Protocol for Tsurugi Relational Database System, arXiv:2303.18142v3 (2026).
[2] Transaction Processing Performance Council: *TPC Benchmark C Standard Specification, Revision 5.11* (2010).
[3] Tu, S., Zheng, W., Kohler, E., Liskov, B. and Madden, S.: Speedy Transactions in Multicore In-Memory Databases, *Proc. SOSP '13*, pp. 18–32 (2013).
[4] Eswaran, K. P., Gray, J. N., Lorie, R. A. and Traiger, I. L.: The Notions of Consistency and Predicate Locks in a Database System, *Commun. ACM*, Vol. 19, No. 11, pp. 624–633 (1976).
[5] Theodorakis, G., Firth, H., Clarkson, J., Crooks, N. and Webber, J.: TuskFlow: An Efficient Graph Database for Long-Running Transactions, *Proc. VLDB Endow.*, Vol. 18, No. 12, pp. 4777–4790 (2025).
[6] Lim, H., Kaminsky, M. and Andersen, D. G.: Cicada: Dependably Fast Multi-Core In-Memory Transactions, *Proc. SIGMOD '17*, pp. 21–35 (2017).

© 1959 Information Processing Society of Japan