RaftによるSiloロギング分散化の実装

田中 昌宏[1,a)]　川島 英之[2,b)]

データベースの障害耐性のため，トランザクションのログを複数サーバに分散させて冗長化させる手法が用いられる．本研究では，信頼性を高めるため，Raftによる合意に基づくSiloロギング分散化の初期実装を行った．性能測定の結果，一部のケースでネットワーク性能から決まる最大スループットを達成したが，全体的に予想より低い性能となった．

1. はじめに

金融システムに用いられるデータベースには，高い信頼性が求められる．システム障害からの迅速な復旧のため，システムを分散させて冗長化する研究が注目されている．その手法の1つとして，ログの分散化による耐障害性がある．Raft2PL [1] では，S2PLのログ複製にRaftプロトコルを導入し，トランザクションやログをグループ化する集約ログ転送法により高速化を実現している．

インメモリ・メニーコアを活用した高性能トランザクション技術として，Silo [2] が提案されている．CCBench [3] により，Siloは現在でも最高性能のプロトコルの1つであると確認されている．しかし，ロギングを含めた性能では，ログの永続化がボトルネックとなり，ログの分散化にはログを転送するネットワークの影響を受ける．

本研究では，Siloのログを Raftプロトコルにより複数のサーバに複製し合意を取る RaftSiloを提案する．本稿では，RaftSiloの初期実装における性能について述べる．

2. RaftSiloの実装

Silo[2]，SiloR [4] で定義されているSiloロギングの特徴は，エポックベースのグループコミットである．512 KiBのログバッファがフルになるか，エポックが更新するタイミングでロガーに転送され，永続化ストレージに書き込みを行う．ロガースレッドでは，担当ワーカーのエポック及び未永続化ログのエポック e_l をチェックし，$\min(e_l) - 1$ 以下のトランザクションについては永続化が完了しているとして，クライアントに完了通知を行う．

RaftSiloの概要を図1に示す．CCBench [3] をベースとするSiloロギング実装 [5] に加えて，新たに実装したRaftサーバと連携させることにより実現した．Raftサーバは，

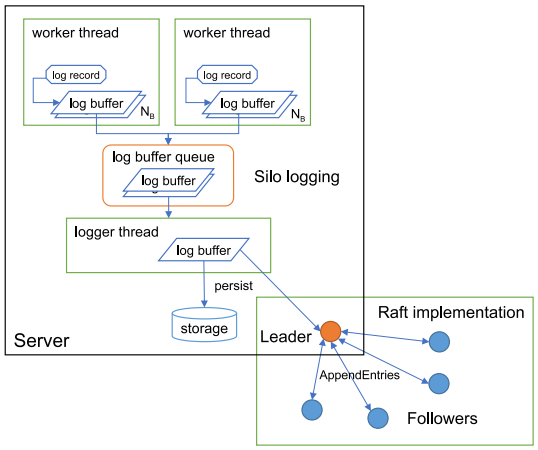

図 1 RaftSiloの概要．

1台のマシンに1スレッドが起動する．今回の実装では，ロガーがログをストレージに書き込むタイミングで，ログバッファを Raftサーバに送る．

Raftサーバは既存のライブラリを用いて実装した．Raftステートマシンの実装にはRedisLabのRaftモジュール [6] を用いた．このモジュールはRaftのステートマシンの動作のみが実装されており，通信やログ永続化の実装はコールバック関数として別に実装する必要がある．また，タイムアウト処理に必要なタイマーも外部から与える必要がある．通信ライブラリにはZeroMQ [7] を用いた．通信やタイマーを扱う非同期処理のフレームワークとして，ZeroMQの高機能版であるCZMQ [8] に含まれるzloopを用いた．その他，メッセージのシリアライズにはMessagePack [9] を用いた．今回は初期実装であり，AppendEntryとRequestVoteのみ実装し，リーダー選挙とログ転送の動作を確認した．

当初は，ZeroMQの通信方式として1対N通信であるREQ-ROUTERパターンを用い，ログはAppendEntryに含めて送っていた．しかしこれでは性能が出なかったため，ログ本体の送信のみ，ストリームのようにデータを配信するPUB-SUBパターンを用いるようにした（図2）．

1 慶應義塾大学大学院 政策・メディア研究科
2 慶應義塾大学 環境情報学部
Keio University
a) masa16.tanaka@keio.jp
b) river@sfc.keio.ac.jp

© 2022 Information Processing Society of Japan

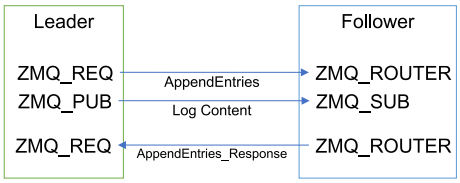

図 2 PUB-SUB パターンによるログ転送.

表 1 評価環境

CPU	Intel Xeon Platinum 8276 2.20GHz
物理/論理コア数	28/56
L1/L2/L3 キャッシュ	32 KiB/1 MiB/38.5 MiB
DRAM	512 GiB
InfiniBand	EDR 1 レーン (25 Gbps)
Ethernet	10 Gbps

3. 測定環境

今回の Raft サーバの構成は，リーダー 1 台とフォロワー 4 台である．測定に用いたマシンは表 1 の通りである．使用したマシンには 4 ソケットに CPU が装着されているが，今回は 1 ソケットのみで測定する．ネットワークは InfiniBand EDR 1 レーンと 10G Ethernet を用いる．使用した ZeroMQ は InfibiBand 用には設計されていないため，IPoIB の使用となる．

ワークロードは YCSB の派生で read : write = 0 : 100, Skew は 0（一様分布）である．この設定により，アボート率が 0 に近く，オペレーションがすべてログになり，ログの生成量が最も多くなる．トランザクション中命令数は 10（命令はすべて update），レコード数は 100 万である．今回は，ストレージ性能の影響を受けず，ネットワークのみの影響を調査するため，ログのストレージへの書き込みを省略して測定した．

今回のログサイズは，1 オペレーションあたり 16 bytes, 1 トランザクションあたり 160 bytes である．ログを 4 フォロワーそれぞれに送るとするとその 4 倍の転送量となる．ログの転送速度から推定されるトランザクションのスループットは，25 Gbps の InfiniBand では約 5 Mtps, 10G Ethernet では約 2 Mtps となる．

4. 測定結果

図 3 にトランザクションのスループットを InfiniBand と Ethernet で測定した結果を示す．Ethernet では，8 スレッドのとき，ログ転送速度から決まる 2.0 Mtps の性能が得られている．しかし，そこからスレッド数を増やすと 1.5 Mtps 程度まで性能が低下している．ログ生成量が増えたときに何らかの干渉が起こっていることが予想されるが，その原因は不明である．

一方，InfiniBand では，16 スレッドのときに 3.0 Mtps で頭打ちとなっており，これは転送速度から予想される

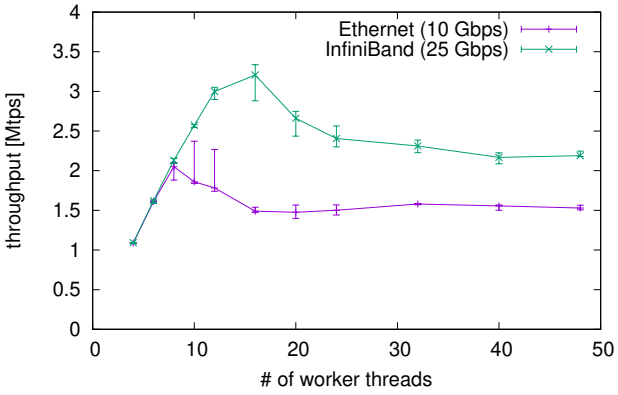

図 3 RaftSilo のトランザクション性能.

5 Mtps より低い．この原因も現時点ではわからない．スレッド数が増えると性能が低下している現象も Ethernet と同様に起こっている．

5. まとめ

Silo ロギングを Raft プロトコルによって分散させる実装を行い，初期性能評価を行った．その結果，ネットワーク性能から決まる最大トランザクション性能は，Ethernet で 8 ワーカースレッドのときに達成したが，ワーカースレッドが多い場合や InfiniBand では達成できなかった．この原因を究明して性能を向上させることが今後の課題である．

謝辞 本研究の成果は，国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務（JPNP16007）の結果得られたものである．

参考文献

[1] Ogura, T., Akita, Y., Miyazawa, Y. and Kawashima, H.: Accelerating Geo-Distributed Transaction Processing with Fast Logging, *IEEE Big Data 2021*, pp. 2390–2399 (2021).
[2] Tu, S., Zheng, W., Kohler, E., Liskov, B. and Madden, S.: Speedy transactions in multicore in-memory databases, *SOSP* (2013).
[3] Tanabe, T., Hoshino, T., Kawashima, H. and Tatebe, O.: An Analysis of Concurrency Control Protocols for In-Memory Databases with CCBench, *PVLDB* (2020).
[4] Zheng, W., Tu, S., Kohler, E. and Liskov, B.: Fast databases with fast durability and recovery through multicore parallelism, *OSDI* (2014).
[5] 田中昌宏，川島英之：エポック同期法による Silo の応答高速化，情報処理学会論文誌データベース（TOD），Vol. 15, No. 3, pp. 63–74 (2022).
[6] RedisLab: A complete implementation of the Raft Consensus Algorithm, https://github.com/RedisLabs/raft.
[7] ZeroMQ: ZeroMQ - An open-source universal messaging library, https://zeromq.org/.
[8] CZMQ: CZMQ - High-level C Binding for ZeroMQ, http://czmq.zeromq.org/.
[9] Furuhashi, S.: MessagePack: It's like JSON. but fast and small., https://msgpack.org/.

© 2022 Information Processing Society of Japan