分散合意アルゴリズム Raft の調査

梶原　顕伍† 　川島　英之†† 　建部　修見††

† 筑波大学情報学群情報科学類　〒 305–8571 茨城県つくば市天王台 1–1–1
†† 筑波大学計算科学研究センター　〒 305–8571 茨城県つくば市天王台 1–1–1
E-mail: †kajiwara@hpcs.cs.tsukuba.ac.jp, ††{kawasima,tatebe}@cs.tsukuba.ac.jp

あらまし　本論文では Raft を用いた分散 KVS について，同期処理が従来のデータベースシステムと同様に主たる性能劣化要因であるという観察結果を示す．また，同分散 KVS の高性能化手法としてサーバの分散化という手法を提案する．

キーワード　分散合意，Raft，KVS

1. 序　　論

データを管理するシステムにおいて高信頼化は必須事項である．高信頼化を達成する一手法として，複数のノードを利用して分散データシステムを構成し，その上でデータを複製させる方法がある．複製を用いることで，分散データベースシステムにおいてマスターデータを保持しているノードが故障した場合でも，複製データを保持しているノードがサービスを提供することが可能になる．これにより，システム内部の障害をクライアントに気づかれることなく，データ管理サービスは提供され続けることが可能である．

複製を管理するには分散して存在する複数のデータオブジェクトの状態について，複数のノードの間で合意を得る必要がある．分散システムで合意を得る手段の一つに，2014 年に Usenix ATC 会議で報告された Raft [5] がある．Raft は Paxos [4] と同等の性能を有しながらも Paxos よりも格段の理解しやすさを提供する分散合意プロトコルである．

Raft を用いれば分散キューバリューストア (KVS) を構築可能である．Raft を用いて作成された高信頼な分散 KVS に etcd [6] がある．etcd は CoreOS においてデータの信頼性を保証するという重要な役割を担う．一方，Raft を用いた分散 KVS の設計をゼロから考えたとき，最高性能を発揮する分散 KVS の構成は筆者が知る限り論じられていない．Raft を用いた分散 KVS では複数のモジュールが存在するが，それらのモジュールの性能に与える影響が調査されていない．また，KVS の性能を極大化するにはデータ分散が有効だと考えられるが，それに関する考察は行われていない．

本論文では Raft を用いた分散 KVS の高性能化について論ずる．まず，分散 KVS について微視的性能を観察した結果を報告する．微視的観察を行うために，我々は Raft に基づいて分散 KVS をゼロから構築した．これを RKVS と表記する．RKVS の特性を評価するために，ログレコードをストレージに同期する処理を省く設定を行った結果，大幅に性能が向上するという結果が得られた．これより RKVS では集中方トランザクション処理システムと同様に，同期処理に関するコストが高いことが示唆されるという知見を得た．次に，データ分散方式によりサーバ (注1) を分散化する方式を検討する．

本論文の構成は以下の通りである．2 章では Raft の構成概要と提案手法を述べる．3 章では RKVS の評価結果を述べる．4 章では分散サーバ方式の検討を行う．最後に 5 章では論文をまとめる．

2. 分散合意手法 Raft

2.1 アーキテクチャ

Raft のアーキテクチャ概要を図 1 に示す．Raft クラスタのノードには，LEADER, CANDIDATE, FOLLOWER という 3 種類の状態がある．LEADER はクラスタ内に 1 台しか存在しない．クライアントは LEADER ノードとインタラクションする．クライアントが FOLLOWER にアクセスしてきた場合は LEADER にリダイレクトされる．FOLLOWER ノードは LEADER ノードに従う．LEADER ノードはクライアントから更新要求を受信すると，ログレコードを生成する．そして LEADER ノードはログレコードをストレージに保存した後，ログレコードを FOLLOWER ノードへ転送する．FOLLOWER ノードは受信したログレコードをストレージに保存すると，ACK を LEADER へ送信する．LEADER を含め，Raft クラスタの過半数が命令を受け取ったことを LEADER が確認した後，クライアントに COMMIT を通知する．最終的には全てのノードが同じ命令を同じ順番で実行するため，全てのノードでデータが複製される．即ち Raft は複製化状態機械 (replicated state machine) に基づいて設計されている．一般に利活用されている複製化状態機械の例として，Chubby [2] や ZooKeeper [3] などが挙げられる．

LEADER ノードが故障したり，LEADER ノードへのネットワークコネクションが切断されると，Raft クラスタ内に LEADER が存在しなくなる．このような場合，FOLLOWER ノードは状態を CANDIDATE へと変更する．クラスタ内のノードは各 CANDIDATE への投票を行い，新しい LEADER を選出する．この選出にはクラスタを構成するノード数の過半数（以後過半数と表現する）の選任が必要である．過半数を得たノードが存在しなければ，LEADER 選出作業を再実行する．

(注1)：Raft では LEADER という用語で呼称される．

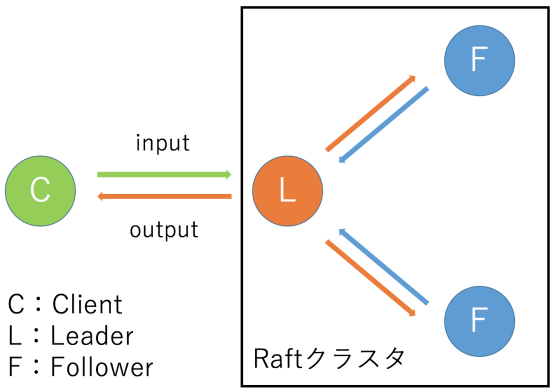

図 1　Raft クラスタのアーキテクチャ

表 1　ノードの仕様

RAM	64 GB
CPU コア数	8 × 2
OS	CentOS release 6.8 (Final)

2.2　RKVS

Raft は複製化状態機械であるから，Raft に基づいて KVS を設計可能である．Raft に基づく KVS で著名なものには etcd [6] がある．本研究では Raft に基づく KVS の微視的評価を行うため，新しい KVS をゼロから設計・実装した．RKVS の構成は図 1 とほぼ同様であり，同図に KVS モジュールが追加されているのみである．クライアントからの接続要求に伴ってスレッドを作成し，マルチスレッドで複数コネクションを並行的に処理する．Raft の原論文 [5] では FOLLOWER 数を 4 として合計ノード数を 5 としており，本研究でも同様の設定を行った．ある LEADER とそれに関する FOLLOWER のノード関係は同一である．従って LEADER に関するスレッドが増加しても FOLLOWER スレッドは増加しない．実装は Java で行い，実行ステップ数は 1911 行となった．

3.　RKVS の微視的評価

本章では RKVS の微視的評価を行った結果を報告する．原論文同様に 5 台のノードを用い，1 台を LEADER とした．ノードの仕様を表 1 に示す．

RKVS のスループットを測定した結果を図 2 に示す．図中の Without Sync は本来実行すべきログのストレージへの同期処理を省いた場合の結果である．この同期処理は集中型データベースシステムにおいて主要なボトルネックであることが知られ，グループコミットなどの諸手法により高性能化が行われる．結果として，RKVS でも同様に同期処理が主要なボトルネックであることが観察された．

4.　複数リーダの利用による RKVS の高性能化の検討

2.2 節で述べた RKVS においては，LEADER ノード 1 台のみが全てのデータを管理する．Raft や Paxos は設定情報 (注2)

(注2)：所謂/etc の情報

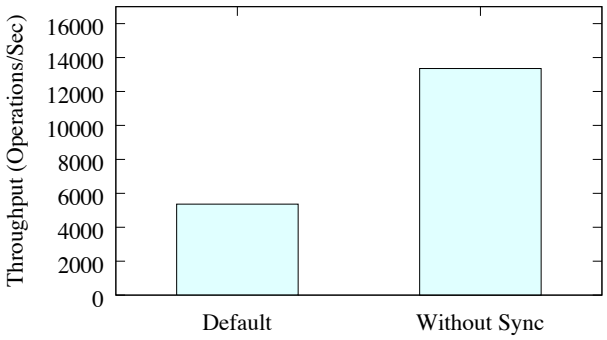

図 2　スループット

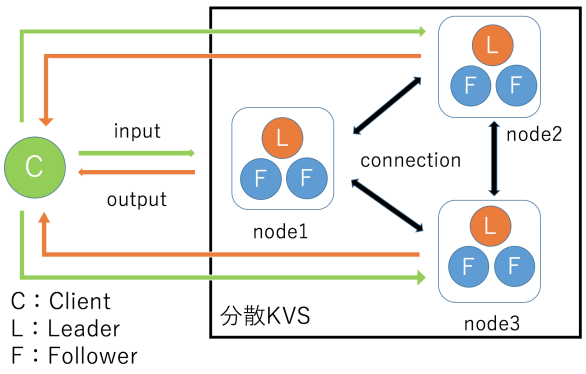

図 3　複数の LEADER が存在する RKVS

を管理するために使われることが多いが，Spanner [1] のように巨大なデータ本体を扱う場合にも使われる．

このような場合，LEADER のコストが増大するため，RKVS の性能が劣化することは容易に推定される．この問題を解決するために，本研究では Raft の LEADER を複数ノードで起動させてデータ分散を行う方針を提案する．この場合，N 個の LEADER のそれぞれはデータの一部のみを担当する．従って各リーダが処理するデータ量は，理想的には従来方式の $1/N$ まで削減される．これにより LEADER のコストが削減することが予見される．図 3 に LEADER が複数存在するような RKVS の概要を示す．3 つのノードが示されているが，各ノードに LEADER が存在し，それぞれがクライアントと通信を行う様子が示されている．

5.　結　　論

本論文では Raft を用いた分散 KVS である RKVS について，同期処理が従来のデータベースシステムと同様に，主たる性能劣化要因であるという観察結果を述べた．また，RKVS の高性能化手法として LEADER の分散化という手法を検討した．

謝　　辞

本研究の一部は,JST CREST「ポストペタスケールデー タインテンシブサイエンスのためのシステムソフトウェ ア」,JST CREST「EBD:次世代の年ヨッタバイト処理に向けたエクストリームビッグデータの基盤技術」,JST CREST「広域撮像探査

観測のビッグデータ分析による統 計計算宇宙物理学」, 科研費「#16K00150」による.

文　　献

[1] *Spanner: Google's Globally-Distributed Database*, 2012.
[2] Mike Burrows. The chubby lock service for loosely-coupled distributed systems. pp. 335–350, 2006.
[3] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the USENIX Annual Technical Conference*, pp. 11–11, 2010.
[4] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, Vol. 21, No. 7, pp. 558–565, 1978.
[5] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference*, pp. 305–320, 2014.
[6] Brandon Philips. etcd 2.3.7 documentation, 2016. [Accessed 2016-10-25].