Vol. 41 No. 6 情報処理学会論文誌 June 2000

非集中型クラウドストレージのスケーラビリティ評価

奥寺昇平† 中村俊介†
長尾洋也† 首藤一幸†

非集中なクラウドストレージにおいて、任意のノードからデー タを保持する担当ノードにリクエストを到達させるためには、各ノード が他のノードを把握する必要がある。特に、クライアントが接続したノー ドから担当ノードに直接、リクエストを送るクラウドストレージでは、全 ノードがシステム全体の最新の状態を保持する必要があり、整合性を保つことが難しい。そこで、GossipProtocol をベースとしたメンバーシップ管理を行うプロ トコルが取り入れられ、効率よく通信を行うことが可能である。一方、このようなメンバーシップ管理もスケーラビリティを制約する要 因の一つとなりうる。この管理方法では、すべてのノードで定期的に通 信が発生するので、ノード台数が増えるにつれ、総通信量が増えるので ある。よって、フロントエンド (例えばストレージであれば、データの読 み書き処理。) の処理効率つまり、アベイラビリティを下げると考えられ る。しかしながら、非集中型のクラウドストレージにおいて、この管理を 行う処理がどれくらいの通信負荷をもたらすのかといったことは知られて いない。そこで本研究では、GossipProtocol を用いる Cassandra を対象として、 ノード台数に応じてシステム全体の通信負荷がどのように変化するのかを計測・考察する。計測の結果、システム全体で発生する通信量は、ノード 台数を n としたとき、$O(n^2)$ でスケールすることを確認した。定量的にクラウドストレージの *gossip* 通信量を計測できた。

A Scalability Study of A Decentralized Cloud Storage

Shohei Okudera,† Shunsuke Nakamura,† Hiroya Nagao†
and Kazuyuki Shudo†

Abstract in English

1. はじめに

分散システムのアベイラビリティとは、システムの可動性、つまり正常 なサービスが提供をどれくらい継続できるかの指標である。クラウド型のサービスが流行すするなか、現在止まっては泊まることないシステムが求 められている。

分散システムシステムのアベイラビリティを確保する戦略として大き くスケールアップ、スケールアウトの二つの戦略がある。スケールアップ とはシステムを構成する各マシンの能力 (CPU, メモリなど) の高機能化、 高性能化を図ることで、システム全体の性能をスケールさせようという戦 略である。従来はこの戦略でシステムの性能向上が図られてきたが、コス トに見合った性能向上が伴わなかった。そこで、google をはじめとしてシ ステムをスケールアウトさせる戦略をとるようになった。スケールアウト とは、システムのアベイラビリティを稼ぐために一台あたりのノードの性 能を挙げるのではなく、安価な商用ノードを並べ、アベイラビリティを確 保するというと戦略である。スケールアップでは、一台あたりのマシンが 安いため、スケールアップ戦略をとった場合と比べて、総合的にコストパ フォーマンスがよい。その一方で、システム全体のノードのどれかが常に 故障が起きるという問題がある。スケールアウトによるスケーラビリティ 確保の戦略では、マシンの数が増加するほど、システムでマシンの故障が 発生する確率は増加するのである。例えば、各マシンの 1 秒あたりの故障 する確率が p =0 .001、マシンの台数を 1000 とする。この時、1 秒あたり システムで故障が起きる確率は p=0.63 に跳ね上がる。そこて?、ハードは 故障するということを前提としたシステムの仕組みづくりが必要になる。

故障を前提とした分散システムで特に重要な必要な機能は、故障を起こ したマシンを外部のマシンから判断できる仕掛けである。これを故障検知 と呼ぶ。故障検知には、もう一つ重要な役割がある。それはマシン

† 東京工業大学
Toyohashi Institute of Technology

が故障 している事実を直接検出したマシン以外のマシンにも伝えることである。 その機能がなければ、個々のマシンで故障検知を行うことになり効率が悪 く、システムのアベイラビリティが下がってしまう。

故障検知を行う代表的な方法に Gossip Protocol がある。Gossip Protocol とは、ソーシャルネットワーク で見られる噂 (ゴシップ) の伝搬をモデ ルとしたアルゴリズムである。Gossip Protocol を利用して、検知した故 障の情報を他のマシンに効率よく伝達することが可能である。しかしなが ら、gossip を利用した故障検知はメインタスクではない。特にクラウドストレージにおいてのメインタスクは、クライアントからのデータの読み書 きリクエストを処理することであり、故障検知はバックグラウンド、サブ タスクとして扱われなければならない。すなわち、CPU サイクルや使用 するネットワークバンド幅が限られている環境で故障検知が実現されるこ とが求められているのである。これらの問題を解決するために gossip を応用したさまざまなプロトコルが提案されている。

しかしながら、クラウドストレージにて gossip が及ぼす影響については、はっきりしていない。このような故障検知に求められる指標には、一貫性、 パフォーマンス、ダイナミックな環境、スケーラビリティなど様々がある。我々は、パフォーマンスの一つである通信量について研究した。

2. 関 連 研 究

何をのせよう?クラウドストレージでネットワーク測定を行った論文か？

3. gossip protocol と Cassandra

3.1 gossip protocol

Gossip Protocol とは、ソーシャルネットワーク で見られる噂 (ゴシップ) の伝搬をモデ ルとしたアルゴリズムである。Gossip Protocol を利用して、検知した故障の情報を他のマシンに効率よく伝達することが可能である。ノード p,q が存在するとする。ノード p とノード q で情報交換を行うとき、Gossip Protocol をその上方伝搬方法によって三つのカテゴリーに分類することができる。

- Pull 型
  ノード p が持っている情報をノード q に送信し、ノード q の情報を更新する方法
- Push 型
  まず、ノード p はノード q に状態についてダイジェストを送信する。ダイジェススストを受け取った q は必要な更新箇所だけをノード p にバックする。最後に、ノード p から必要なノード情報をおくる。
- Pull&Push 型
  pull 型の gossip と似ている。加えて、ノード q もまた、ノード p と比較して古くなった参加者キーのリストを送信る。

ここでは、Pull&Push 型を応用したプロトコル Scuttlebutt について詳しく説明する。

Scuttlebutt とは、分散システムにてノード同士がイベンチュアリーコンシステンシーに情報の融合 (交換) を行うためのプロトコルの一つである。Pull&Push 型ゴシップをベースとしている。特に、使用できるネットワーク帯域幅、CPU サイクルが限定されているときを想定している。具体的には、使用するネットワーク帯域幅を抑えるたｍに、更新したい情報を gossip する相手に渡すことをやめる。ある MTU(Message Transfer Unit) を決めて、そのサイズ内で送信できる情報を送信する。どの更新情報を優先するかで議論があり、Scuttlebutt では、ノード p からノード q へゴシップ通信を行うとき、以下のように通信が行われる。(TODO1:mapping(nodename-key-value-version),deltas(key-value-version) の説明)
(TODO2:mapping 内のどのデルタスも異なるバージョンを持つように更新することをいう。)

(1) 1:p がもつ mapping の中で一番新しいバージョンを q に送信する
(2) 2:q がもつ mapping の中で一番新しいバージョンを p に送信し返す
(3) 3:q から受信したバージョンよりも新しいバージョンをもつ p のデルタスのみ (p のマップのサブセット) を q に送信する。
(4) 4:3. と同様に、p から受信したバージョンよりも新しいバージョンをもつ q のデルタスのみ (q のマップのサブセット) を p に送信する。

最大のバージョンをダイジェストで通信することで、無駄な通信を行うことを控えている。さらに MTU が存在するときには、3.4. ですべての更新情報を送信することはできない。このとき、デルタス同士の優先順位は、バージョンによって決定することにする。

この方法で、○○の結果がでました。(TODO:この欄いる？論文をよむ、まとめる)

3.2 Cassandra

Cassndra は大量の構造化データを管理する分散ストレージシステムである。地理的に離れて分散された

数百ノードからなる商用サーバー上で運用されることが想定されていて、可用性の高いサービスを提供し、単一故障点を持たない。このようなスケールでは大小の故障はひっきりなしに起こる。このように故障がひっきりなしに起きる状況においても、カサンドラは一貫した状態を保つのよう設計されているので、信頼性、スケーラビリティも高い。Cassandra は、非集中型のクラウドストレージである。メンバーシップ管理について詳しく見ていく。

3.2.1 Cassandra のメンバーシップ管理

Cassandra を構成する各ノードは、リング上の ID 空間上に配置される。各マシンは、システムに参加しているすべてのメンバーを把握している。また、新規ノード追加時には、指揮するノードに予めシステム内のメンバーの一部のアドレス (これを seed と呼ぶ) を設定しておき、そのメンバーに新規に参加することを伝えた後で、実査にクラスタに加わる。メンバー情報を上記で上げた gossip プロトコルをベースにノード情報を交換している。具体的には、以下のロジックに沿って情報交換は行われている

ゴシップタイマータスクは、毎秒動作する。各タスクが動作する間以下のルールによってゴシップの交換を行う。

(1) もし他の生存しているノードが存在したら、ランダムで生存しているノードを選択してゴシップ通信を行う。
(2) 生存ノードと到達できないノードの数に応じたある確率によって、ランダムで到達できないノードにゴシップ通信を行う。
(3) 1. でゴシップ通信を行ったノードが seed ではなかったとき、あるいは seed ノードの数より生存しているノードが少ないときは、生存ノード、seed, 到達不可のノードの数に応じたある確率で、ランダムで seed に gossip 通信を行う。

このルールは、Cassandra ノードは、毎秒 1-3 回のゴシップ通信を行うことを示している。(ただし、クラスタを構成していないならば 0 回。)

一回の gossip 通信では、三章で説明した gossip 通信が用いられ、メンバーの生存情報を交換している。

4. Cassandra の軽量化と測定手法

4.1 Cassandra ノードの軽量化

(TODO1:A. クラスタ構成するか？B. 起動直後か？でメモリー、スレッドは変化する。このことに木をつけて各必要がある。)
(TODO2:環境のことを記す必要があるのでは？)
(TODO3:3 つの工夫を Cassandra ノードに施しました。その結果○○となりました。のほうが綺麗か！)
(TODO4: readrepair のせってい OFF とか、replication の話とか盛り込む？) 実験を行うにあたって物理リソースの都合上、1 台あたり複数の Cassandra ノードを起動する必要があった。デフォルトの設定では、1 ノードの Cassandra 起動するためににデータを全く保持していない状態でさえ、スレッド数が 130、メモリー使用領域が 120M 程度とリソースを多く消費する。複数ノードでクラスタを構成した時にさらにリソースが消費される。リソースの消費を抑えるために、Cassandra データ保持部分のプログラムの改変と、設定パラメータのチューニングを行った。

4.1.1 プログラムの改変

Cassandra は、データを保持していない状態であってもシステム管理のためのテーブルを保持する必要があり、メモリー使用量域がかさむ。この点を踏まえメモリー使用領域を減らすためにプログラムに改変を加える。

Gossip Protocol の通信量を測定する実験では、実データがどのようなものかは関係がない。そこで、実際のデータを保存するのではなくデータサイズだけを保管するように変更した。

4.1.2 パラメータの調整

- JVM 最大ヒープサイズの制限を変更
  Cassandra は JVM 上で動作する。多数台起動するために、JVM 最大ヒープサイズの制限を 1G から 160M に変更した。
- 設定ファイルのパラメータ調整
  設定ファイルでにて、同時読み込みを許す最大値、同時書き込みを許す最大値を制限することでスレッド数を減らした。

また、これらのパラメータの調整は、Gossip Protocol の通信量を測定する実験では、直接関わらないパラメータであることに留意する。この調整により、Cassandra を○○ノード起動したときに起動直後で、1 ノードあたりの平均で、メモリが●● M ⇒●● M, スレッド数が●● M ⇒●● M と削減できた。

4.2 測 定 手 法

以下の実験を行った。

- 実験 1:gossip の通信量の測定
- 実験 2:YCSB を用いてデータの読み書きを行ったときの通信量の測定

4.2.1 実験 1:gossip の通信量の測定

実験 1 では gossip の通信量の測定を行った。実験シナリオは、マスターとなるマシンを 1 台とワーカー

となるマシンを１０台を用意した。ワーカーマシンを Node1,Node2,...Node20 と名付ける。マスターの役割は、通信量計測の開始・終了、Cassandra ノードの起動, 計測した記録の解析をワーカーに指示すること、最終的な通信量の推定を行うである。一方、ワーカーの役割は、通信量の計測、Cassandra ノードを起動すること、通信量の解析である。また、1 台あたり複数の Cassandra ノードを立ち上げる必要がある。Cassandra ノードの立ち上げ方は、30 秒ごとに、1 台あたり 10 ノードの Cassandra を一度に起動し、これを目指す台数に達成するまで続ける。最初の Cassandra ノードを起動した瞬間から各マシンで 10 分間の通信量を計測した。

計測後に各マシンで通信量を解析し、マスターとなるマシンに解析結果を送信する。マスターは、送られて通信量から合計値を出し、Cassandra で発生する通信量の推定を行う。

マスター、ワーカーで実行するプログラムは、シェルスクリプトでプログラムを書き、各ワーカーへの指示は、GXP を使用して制御した。GXP とは、同じコマンドを多数のホストで並列に実行するためのジョブスケジューラのことである。また、パケット情報の解析には、java,R, シェルスクリプトを使った。

4.2.2 計測方法について

実験 l 、実験 2 の計測にあたっていくつか工夫した共通の点を紹介する。

- ユーザー, プロセスのシステムリソース制約を外す
  通常のオペレーションでは、 l ユーザに共有のシステムリソースを占有されないように管理されている。具体的には、 l ユーザが同時に実行出来るプロセス数、ファイル・ディスクリプタの数や、ユーザーが実行するプロセスにおいて、仮想メモリーの使用領域、物理メモリーの使用領域などが制限される。
  Cassandra ノードの実行には、メモリー使用量域が大きく、使用するスレッド数が多い。我々の実験環境では、スレッド数が 130 程度であった。linux のデフォルトの設定では、 l ユーザが同時に実行出来るプロセス数は 1024 であるので、Cassandra ノードを 7 台までしか起動できなかった。そこで、linux のユーザーリソースを決める設定ファイルを編集し、 l ユーザーのリソース制限、 l プロセスのリソース制限を緩和した。その結果、多数台の起動が可能になった。
- NFS を利用した書き込みをできるだけ控える
  概して一度に多数ノードを起動するときは、実行ファイルを共通にして起動することが多い。すなわち、NFS 上に共通の実行スクリプトを保存し、実行時に読み込むことが多い。しかしながら、一度に非常に多数の NFS を通じた読み込みが発生すると、上手くアクセス出来ないことがある。そこで、各マシンのローカル上に実行ファイルを保管し、起動時にそのファイルを読み込むようにした。
- IP エイリアシングを利用し、プライベートネットワークを構築
  Cassandra のメンバー管理では、IP アドレスでメンバーを認識する。つまり、今回の実験ようにｌマシンあたり複数ノードの Cassandra を立ち上げようとすると、不都合が生じる。
  そこで、IP エイリアシングを使用して仮想アドレスを作成し、Cassandra ノードごとに割り振ることにした。その結果、同じマシン上に立ち上がった Cassandra マシン同士の差異が明らかになり、不整合が起きなくなった。
  さらに、通信量の測定の際にはノイズを防がないといけない。ノイズとは、Cassandra ノード以外から要求されるリクエストのことである。具体的には、ARP や ssh などのパケットのことである。これらのパケットを誤って計測してしまうことを避けるために、IP エイリアスリングを行うと同時に、プライベートネットワークを構築した。このネットワークに参加しているのは、Cassandra ノードだけである。この作業で、プライベートネットワーク内で飛び交うパケットのみを取得すればよく、ノイズが減らすことができる。
  具体的には、10.20.0.0/16 のネットワークを構築した。さらに Cassandra ノードが物理的にどのマシン上で起動しているかを判別しているために、実マシン (Node11,Node12,...,Node20) 上の番号 n を使用し、Node[n] 上で起動するマシンは、仮想的に 10.20.n.0/24 なるサブネットを設けた。n=19 のとき、10.20.19.1 〜10.20.19.254 までの仮想アドレスを作成し Cassandra ノードに割り当てた。

- tcpdump の使用
  通信量の測定は、tcpdump を使用した。上述したように、Cassandra ノード同士のやりとりはプライベートネットワーク上で行われるので、この

ネットワークをまたぐすべての TCP パケットのサイズを記録すればよい。具体的には、tcpdump に以下のオプションをつけて実行した。取得するパケットは二つの条件でフィルターをかけた。

- src net 10.20.0.0/16
  このフィルターにより、指定したデバイスを経由したパケットのうち、送信元が 10.20.0.0/16 のネットワークであるパケットのみが取得する。つまり、これで、このネットワーク以外のパケットを取得しないことになる。この条件で、余計な ARP クエリなどを弾ける。
- dst net 10.20.(マシン番号).0/24
  このフィルターにより、指定したデバイスを経由したパケットのうち、受信元が 10.20.(マシン番号).0/16 のネットワークであるパケットのみが取得できる。つまり、tcpdump を実行するマシンで実行する Cassandra ノード宛のパケットを取得することになるのである。

この 2 つの条件により、他のマシンで起動している Cassandra ノードからこのマシンで起動している Cassandra に送られてくるパケットだけを取得できるのである。

- 同じマシン上で動作している Cassandra ノード同士の通信は取得できない。
  一方、tcpdump で測定する方法では同じマシン上で動作している Cassandra ノード同士の通信は取得できないことに留意したい。

4.3 通信量の測定について

tcpdump を使用した上のような計測方法では、同じマシン上での CassandraNode 同士の通信を計測することはできない。そこで我々は下の仮定をもとに、計測した総通信量から Cassandra ノードで発生する総通信量を推測することにした。

- 仮定:任意の Cassandra Node 同士の通信量は平均すると同じである。

n 台の各マシンで m 台の cassandra ノードを起動したとする。つまり、システム全体で合計 n × m 台の Cassandra ノードが起動されている。各マシンで tcpdump を使用して計測した得られたトラフィックの合計を T とし、Cassandra ノードで発生する通信量 TT とく。このとき、

$$T = ((n-1)*m \text{ 台の } cassandra \text{ ノードの通信量}) \quad (1)$$

である。上の仮定を用いると、n * m 台の cassandra ノードで発生する通信量 TT は、

$$TT = T*(n*m)/((n-1)*m) = T*n/(n-1) \quad (2)$$

と Cassandra ノードで発生する総通信量を推測することができた。

4.4 実 験 環 境

以下に実験環境を示す。

- Cassandra 0.6.6
- OS Linux 2.6.35.10 74.fc14.x86_64
- CPU: 2.40 GHz Xeon E5620 × 2
- Java 仮想マシン: Java SE 6 Update 21
- メモリー: 32GB RAM
- ネットワーク: 1000BASE-T

5. 実験・評価

5.1 本実験 1:gossip の通信量

図??は、10 秒あたりのマシン間の総通 信量の変化をノード数別に表したグラフであ る。(ただし、1M $=10^6$,1K$=10^3$ とする。) ノードの台数によらず、100 秒以降は通信量 が安定していることがわかる。図? 2 は、ノード数と通信量が安定して いる時の (ここでは、実験開始から 200-300 秒 後とした)1 秒あたりの通信量の平均をプロッ トしたものである。

5.2 評　　価

5.2.1 総通信量の見積もり

図中の曲線は、プロット した点から二次関数でフィッティングしたも のである。n をノードの台数として得られた関数は、

$$[\text{通信量 } (bit)] = 224.6 \times n^2 + 4314.8 \times n \quad (3)$$

である。よって、通信量は $O(n^2)$ でスケール することがわかった。 この関数から、ノード台数をパラメータとし て Cassandra の Gossip Protocol で発生しうる全体の通信量を推測することができる。例えば、n = 1000 のとき、[通信量] = 229Mbps と なる。このように、この関数を使って総通信量 が見積もることができる。また、クラスタの設 計時にも活かすことができる。

5.2.2 1 ノードあたりの通信量

また同様に、1 ノードあたりの通信量を見積もることも可能である。総通信量を Cassandra node 台数 n で割った値が、1 ノードあたりの通信量 t' となる。つまり、

$$\begin{aligned} t' &= [\text{ 1 ノードあたりの通信量 } (bit/s)] \\ &= (224.6 \times n^2 + 4314.8 \times n)/n \\ &= 224.6 \times n + 4314.8 \end{aligned} \quad (4)$$

と O(n) でスケールすることがわかる。グラフに示す

[width=15cm]img/time-traffic.eps

図 1 infer-traffic の実行結果

[width=15cm]img/node-traffic.eps

図 2 infer-traffic の実行結果

と以下になる。X 軸がノード台数、Y 軸が通信量である。また、この通信量はメンバーシップ管理で受信、送信する通信量のそれぞれの値である。

5.2.3 通信量が $O(n^2)$ でスケールしていく理由

最後に、メンバーシップ管理の通信量が $O(n^2)$ でスケールしていく理由について考察する。

ノード台数を n として、安定時に発生する通信量が $O(n^2)$ であることをしめす。

$$(\text{ノード台数}) = n \tag{5}$$

$$(\text{総通信量}\,(bit/s)) = (\text{1 ノードあたりの通信量}\,(bit/s)) * (\text{ノード台数}) \tag{6}$$

さらに、(1 ノードあたりの通信量 (bit/s)) である t' を分解すると、

$$\begin{aligned} t' &= (\text{1 ノードあたりの通信量}/s) \\ &= (\text{1 秒毎の } gossip \text{ 通信する回数}) * (\text{一回の } gossip \text{ 通信の通信量}) \end{aligned} \tag{7}$$

メンバー構成が安定した時を考える。Cassandra のメンバー管理シップでは、メンバー構成が安定したときは、毎秒 1-2 回の gossip 通信が行われる。

$$(\text{1 秒毎の } gossip \text{ 通信する回数}) < 2 = Order(Const) \tag{8}$$

一方、

$$(\text{一回あたりの } gossip \text{ 通信の通信量}) = Order(n) \tag{9}$$

(TODO:上の式は、非常に怪しい！)

よって、

$$(\text{1 台あたりの通信量}) = O(n) \tag{10}$$

$$(\text{総通信量}) = Order(n^2) \tag{11}$$

となることが証明できた。また上記に Cassandra 固有の方式はほぼないといえる。よって、メンバー構成が安定した時の gossip base のメンバーシップ管理アルゴリズムの結果といえる。

6. おわりに

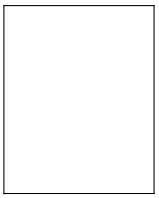
本論では、GossipProtocol を用いる Cassandra を対象として、ノード台数に応じてシステム全体の通信負荷がどのように変化するのかを計測・考察した。計測の結果、システム全体で発生する通信量は、ノード台数を n としたとき、$O(n^2)$ でスケールすることを確認でき、定量的にクラウドストレージの gossip ベースのメンバーシップに要する通信量を計測することができた。

謝辞 ありがとうございます！

参考文献

1) Avinash Lakchman and Prashant Malik: Cassandra ? A Decentralized Structured Storage System.(Proc. LADIS ’09,2009).
2) Giuseppe de Candia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakchman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels: Dynamo: Amazon’s Highly Available Key-value Store (SOSP ’07,2007)).
3) Robbert van Renesse, Dan Dumitriu, Valient Gough, Chris Thomas and Amazon.com, SeattleEf?cient Reconciliation and Flow Control for Anti-Entropy.(LADIS ’08, 2008)
4) Marcia Pasin,St?phane Fontaine,Sara Bouchenak: Failure Detection in Large Scale Systems: a Survey, (NOMS Workshops ’08, 2008).
5) 丸山不二夫, 首藤一幸: クラウドの雲, pp.626–634 (1989).
6) Cassandra Wiki, http://wiki.apache.org/cassandra/

(平成 12 年 2 月 4 日受付)
(平成 12 年 5 月 11 日採録)



奥寺 昇平（正会員）
東京工業大学