Vol.2011-ARC-195 No.22
Vol.2011-OS-117 No.22
2011/4/14

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[†1]　中村俊介[†1]
長尾洋也[†1]　首藤一幸[†1]

非集中型クラウドストレージでは，ノード同士が協調してメンバーシップ管理を行う．この場合，gossip プロトコルをベースとしたメンバーシップ管理を行うことで，効率よく通信を行うことが可能である．

しかし，このような管理方法では，全ノードで定期的に通信を行うため，ノード数が増えるにつれシステム全体の通信量が増加し，システムのスケーラビリティを制約する要因となりうる．しかしながら，非集中型クラウドストレージにおいて，gossip プロトコルをベースとしたメンバーシップ管理がどの程度の通信負荷をもたらすかは知られていない．

そこで本研究では，gossip プロトコルをベースとしたメンバーシップ管理を行う Cassandra を対象として，ノード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．

その結果，システム全体で発生する通信量を定量的に計測することができた．その通信量 $T(bit/s)$ は，ノード数を n とし，$T = 224.6 \times n^2 + 4314.8 \times n$ という関数で表現でき，$O(n^2)$ である．これはクラスタ設計時に有用な知見となる．

A Scalability Study of A Decentralized Cloud Storage

Shohei Okudera,[†1] Shunsuke Nakamura,[†1]
Hiroya Nagao[†1] and Kazuyuki Shudo[†1]

A decentralized cloud storage requires a protocol to share membership information of all nodes. A node finds a responsible node for a datum based on the information. The protocol such as a gossip-based one yields network traffic and greater number of nodes turns out larger amount of traffic. The traffic is a potential candidate of a cause which limits scalability in the number of nodes. It has not been shown quantitatively.

This paper shows a methodology to measure such background traffic produced by a cloud storage. The first target was a cloud storage Cassandra, and measured results outlined its scalability as $O(n^2)$.

1. はじめに

情報爆発時代の到来で，クラウド上に格納されるデータ量が飛躍的に増加している．このような大規模データを高速に処理するために，よりスケーラブルなストレージシステムが求められている．クラウドストレージの中でも Amazon Dynamo[9]，Apache Cassandra[3] といった “非集中型”のアーキテクチャを持つクラウドストレージが重要性を増している．“非集中型” とは各ノードの機能・役割が対等であることを意味していて，スケーラビリティが高いという特徴がある．このような非集中型のクラウドストレージでは，任意のノードで受け取ったデータへのリクエストが，データの保持を担当するノードまで転送される．そのため，各ノードは転送先候補となるノードを把握しておく必要がある．システム内でノード数が増えるに従い，ノードの故障，追加，削除が発生しクラスタの構成変化が日常茶飯事となるため，各ノードはノード同士で情報交換を行い，転送先候補を適切に管理しなければならない．この管理をメンバーシップ管理と呼ぶ．本研究ではメンバーシップ管理に焦点を当てる．

メンバーシップ管理に使用されるネットワーク帯域幅や CPU などのリソース消費量は，最小限に留めなければならない．なぜならば，クラウドストレージでは，データの読み書きのためのリソースが優先されるべきであるからである．高い可用性を実現するために，極力少ないリソースでメンバーシップ管理を行うことが求められる．

Dynamo，Cassandra などの非集中型クラウドストレージでは，gossip プロトコルをベースとしたメンバーシップ管理を行っている．しかし，このような管理方法では，全ノードで定期的に通信を行うため，ノード数が増えるにつれシステム全体の通信量が増加する．その通信量がスケーラビリティを制約する要因となりうる．

しかしながら，非集中型クラウドストレージにおいて，gossip プロトコルをベースとしたメンバーシップ管理がどの程度の通信負荷をもたらすのかは知られていない．そこで本研究では，クラウドストレージに適用されている gossip プロトコルベースのメンバーシップ管理において，その通信量がノード数に応じてどのように増加していくのかを計測し，評価，考察した．

†1 東京工業大学
Tokyo Institute of Technology

Vol.2011-ARC-195 No.22
Vol.2011-OS-117 No.22
2011/4/14

本章の構成は以下のとおりである．2 章で研究背景として，gossip プロトコルを説明し，クラウド環境における gossip プロトコルをベースとしたマルチキャストを提案・評価した関連研究を紹介する．続いて，本研究の実験で使用する非集中型クラウドストレージである Cassandra を概説する．3 章でスケーラビリティの評価手法を説明する．4 章で Cassandra におけるメンバーシップ管理の通信量の測定，評価を行い，5 章で本研究の貢献と今後の課題をまとめる．

2. 研究背景

この章では，非集中型クラウドストレージのメンバーシップ管理に用いられる gossip プロトコルを説明し，次に本研究の実験で使用する非集中型クラウドストレージである Cassandra について説明する．gossip プロトコルの節では，まず gossip プロトコルの分類を行い，次にクラウド環境に適した gossip プロトコルの応用である Scuttlebutt を採り上げる．また関連研究として，クラウド環境における gossip プロトコルをベースとしたマルチキャストを提案・評価した論文を紹介する．

2.1 gossip プロトコル

分散システムにおける情報の伝搬方法は，伝搬先となるノードの選択方針によって分類することができる．具体的には，すべてのノードに直接伝搬する全対全型，ネットワーク的に近いノードに伝搬する近傍型，ランダムで選択したノードに伝搬するランダム型などであり，それぞれに利点と欠点がある．全対全型は，システム全体への伝搬に要する時間は短いが，すべてのノードとの通信が必要となるので，スケーラビリティが確保できない．また近傍型では，ネットワークの局所性を意識するため，通信コストは改善されるものの，特に近傍ノードについて，ネットワーク情報を常時管理していなければならない．ランダム型は，ノードの激しい出入りに耐性があるといった利点がある．ランダム型の代表的な伝搬プロトコルとして，gossip プロトコルが挙げられる．

gossip プロトコルとは，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プロトコルである．gossip プロトコルを利用して, 情報を他のノードに効率よく伝達することが可能である．ノード P がランダムでノード Q を選択し，情報交換を行うとき，ノード間の伝搬方法によって三つに分類することができる[4]．図 1 は，左から A. pull 型，B. push 型，C. pull&push 型の伝搬方法を表している．

- A. pull 型
  ノード P が保持する情報をノード Q に送信し，ノード Q が保持する情報を更新する（図 1 左）．
- B. push 型
  ノード P が保持する情報のダイジェストをノード Q に送信する．ダイジェストを受け取ったノード Q は，更新が必要な箇所だけをノード P に送り返し，ノード P が保持する情報が更新される（図 1 中）．
- C. pull&push 型
  pull&push 型は，pull 型の gossip プロトコルと似ている．違いは，ノード Q がノード P に更新箇所を送る際に，ノード Q が保持する情報のダイジェストを添付することである．ノード P が保持する情報を更新すると同時に，受け取ったノード Q のダイジェストをもとに，ノード Q が保持する情報を更新させる（図 1 右）．

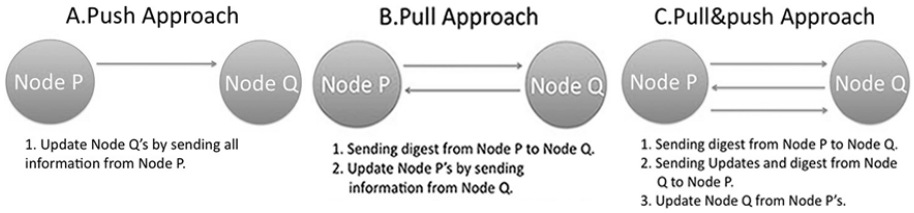

図 1 ノード間の伝搬方法による gossip プロトコルの分類

Scuttlebutt

ここでは，pull&push 型の gossip プロトコルで，クラウドストレージに用いられる Scuttlebutt[5] を詳しく説明する．

Scuttlebutt は，分散システムで各メンバーが保持する情報を共有をするためのプロトコルの一つである．特に，使用できるリソース (ネットワーク帯域幅，CPU など) が制限されている環境で，情報共有を行うことを想定している．想定されている環境に，クラウドストレージにおけるメンバーシップ管理は該当する．なぜならクラウドストレージにおいて，データの読み書き処理がメインの処理であり，メインの処理に最小限の影響で，最新のノード情報の共有することが求められるからである．

Scuttlebutt ではリソース消費を抑えて情報共有を行うために，あるメッセージの最大長に収まる範囲で優先順位が高い更新情報だけを他のノードに伝搬させる．この方法により，システムの整合性を保ちつつ，ネットワーク帯域幅の消費を抑えた情報の共有が可能である．非集中型クラウドストレージの Cassandra はこれを応用したメンバーシップ管理を行っ

Vol.2011-ARC-195 No.22
Vol.2011-OS-117 No.22
2011/4/14

ている．

2.2 関 連 研 究

関連研究として，クラウド環境で gossip プロトコルをベースにしたマルチキャストを提案，評価している研究を挙げる．クラウドのような複数のデータセンタ上で構築される分散システムでは，コストが高くリソース制約があるリンクが存在する．例えば，データセンターを結ぶリンクである．シンプルな gossip プロトコルをベースとしたマルチキャストは，データの伝搬に信頼性があり、スケーラブルかつ堅牢なアプローチである．一方で、その冗長性から，リンクとノードにおけるリソース消費が大きい．そこで Matos ら[2] は，局所的なリンクを優先するオーバレイを構築し，局所性を考慮して伝搬を行うことで，制約のあるリンク間でのトラフィックを減少させた．

本研究との関連は，提案・評価を行った通信手法が gossip プロトコルをベースにしていることである．本研究との違いの 1 つ目は，Matos らは一般のクラウドを対象としているのに対して，本研究は非集中型クラウドストレージに焦点を当てることである．一般のクラウド環境が対象であるので，その言及するところは汎用的であるが，クラウドストレージといった特定の領域に対して深い言及，考察はなされていない．2 つ目は，本研究と Matos らの研究は，ともに gossip プロトコルを応用しているが，その通信目的が異なることである．Matos らは通信目的がマルチキャストであるのに対し，本研究はメンバーシップ管理を対象としている．マルチキャストであれば，情報の伝達は一回であり，定期的に行われるものではない．一方，メンバーシップ管理においては，一定間隔でノードの情報を監視，他のノードへの伝搬を行わなければならない．そのため，必要となる通信量，CPU 負荷といったものが全く異なり，当然，その評価も異なる．

実際，gossip プロトコルをベースしたメンバーシップ管理を用いるクラウドストレージに対するスケーラビリティ評価は未だなされていないといった現状がある．

2.3 Apache Cassandra

Apache Cassandra[3),6)] は，非集中型のクラウドストレージである．米 Facebook 社がオープンソースソフトウェアとして公開し，現在 Apache トップレベルプロジェクトとして活発に開発が進められている．複数のデータセンタ上に分散して配置された多数のノードでクラスタを構成することを想定しており，高い可用性と単一故障点を持たない非集中な分散モデルが大きな特徴である．

Cassandra のメンバーシップ管理について説明する．クラスタを構成する各ノードは，リング状の ID 空間に配置される．また各ノードは，クラスタ上のすべてのノードの状態を把握する．クラスタに新規追加されるノードは，まず，seed ノードと呼ばれるノードと通信を行い，クラスタに加わる．各ノードが保持する情報の伝搬は，2.1 で述べた gossip プロトコルの Scuttlebutt を用いて行う．具体的には，以下の手順に沿って情報交換を毎秒行う．

STEP1: ランダムに生存ノードを一つ選択し，情報交換を行う．

STEP2: 生存ノードと通信不能ノードの数に応じたある確率で，通信不能ノードからランダムに一つを選択し，情報交換を行う．

STEP3: STEP1 で情報交換を行ったノードが seed ノードではないときに適用する．seed ノードが生存ノードよりも少ないとき，生存ノード，seed ノード，通信不能ノードの数に応じたある確率で，seed ノードからランダムに一つ選択し，情報交換を行う．

Cassandra の各ノードは，以上の手順に従った情報交換を毎秒 1~3 ノードと行う．

3. スケーラビリティの評価手法

スケーラビリティを評価するためには，Cassandra ノードを実際に用意できるマシン台数以上に多数起動し，計測を行う必要がある．具体的には，Cassandra が狙う規模である数百ノードあるいはそれ以上を起動しなければならない．そこで，マシン 1 台あたりに多数の Cassandra ノードを起動することが必要になる．

この章では, このような条件のもとでスケーラビリティの評価を行うための手法を説明する．“1 マシンで複数ノードの起動”，“Cassandra の軽量化”，“通信量の測定”の順に本研究での評価手法を述べる．

3.1 1 マシンで複数ノードの起動

Cassandra のように IP アドレスでノードを識別するソフトウェアでは，IP エイリアシングを利用することで 1 マシンで複数ノードの起動が可能になる．Cassandra を 1 マシンで複数ノード起動させるためには，これに加え，通信ポートやデータの保存場所などに関する設定を各ノードで行う必要がある． そこで，このような個別の設定を各ノードで行い，多数のノードを一括して起動させるスクリプトを作成した．

IP エイリアシングの利用とプライベートネットワークを構築

Cassandra は IP アドレスをもとにノードを識別する．Cassandra のように IP アドレスでノードを識別するソフトウェアでは，1 マシンで複数ノードを立ち上げるためには，各ノードに対して IP アドレスを付与しなければならない．

この場合，IP エイリアシングを利用して複数の仮想アドレスを作成し，それを各ノードに割り振ればよい． その結果，同じマシン上で動作するノードの識別が可能になる．

Vol.2011-ARC-195 No.22
Vol.2011-OS-117 No.22
2011/4/14

さらに，通信量の測定の際にはノイズを防がなければならない．ここでのノイズとは，測定対象のノード以外から要求されるリクエストを指している．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結果に加えてしまうことを避けるために，IP エイリアシングを行うと同時に，測定対象のノードのみが参加するプライベートネットワークを構築する．これにより，プライベートネットワーク上のパケットを取得することで，これにより，測定対象ではないパケットの受信する可能性を低くできる．

本研究では，具体的に 10.20.0.0/16 の Cassandra ノードのネットワークを構築した．さらに，各 Cassandra ノードが動作しているマシンを容易に特定するために，マシン (Machine1, Machine2，...，Machine10) 番号 n を使用し，各 Machine[n] に仮想的な 10.20.n.0/24 なるサブネットを設けた．例えば，Machine9 で起動する Cassandra ノードに割り当てられる仮想アドレスは，10.20.9.x ($1 < x < 254$) である．

ユーザ・プロセスあたりのシステムリソース制約の緩和

ここでは，多数ノード起動への障壁となる OS の制約について述べる．Linux などのオペレーティングシステムでは，1 ユーザ，1 プロセスに，共有のシステムリソースが占有されないように管理されている．具体的には，1 ユーザが作成可能なプロセス数，ファイル・ディスクリプタの数や，1 プロセスあたりの仮想メモリーの使用量，物理メモリーの使用量などが制限される．

我々の実験環境では，数ノードでクラスタを構成するだけで 1 ノードあたりのスレッド数が 130 程度必要であった．さらに，クラスタを構成するノード数を増加させると，必要なスレッド数は増加する．Linux のデフォルトの設定では，1 ユーザが作成可能なスレッドの数は 1024 であるので，Cassandra を 7 ノードまで起動可能であった．

そこで，Linux のユーザーリソースを制限する設定ファイルを編集し，1 ユーザー，1 プロセスあたりのリソース制限を緩和した．その結果, さらに多数のノードが起動可能になった．

3.2 Cassandra の軽量化

3.1 節で 1 マシンで複数ノードを起動させる方法について述べた．しかしながら，これだけでは “多数” のノードを 1 マシンで起動することは難しい．これは，1 ノードの Cassandra は，データを保持していない状態にも関わらず，スレッド数が 130，メモリー使用量が 120 Mbyte 程度とリソースを多く消費するからである．以後，$G = 2^{30}$ $M = 2^{20}$，$K = 2^{10}$ とするからである．またクラスタを構成した際には，1 ノードあたりのリソース消費量がさらに増加する．多数の Cassandra ノードを起動するには，1 ノードを起動するためのリソースの消費量を抑える必要があった．そこで，データ保持部分のプログラムの改変と，設定パラメータの調整を行った．

プログラムの改変

メモリー使用量を削減するためにプログラムを改変した．

メンバーシップ管理の通信量を測定するためには，実データの保存は必要ない．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存するように変更し，メモリーの使用量を削減した．注意すべき点は，このプログラムの改変よるメンバーシップ管理への影響は全くないことである．

パラメータの調整

- JVM 最大ヒープサイズの制限
  Cassandra は Java 仮想マシン (JVM) 上で動作する．多数ノードを起動するために，JVM 最大ヒープサイズをデフォルトの 1 Gbyte から 160 Mbyte に変更した．160 Mbyte という数字は，データを保持していない状態でのメモリ使用量 120 Mbyte 程度に余裕をもたせた値である．
- Cassandra のリソース使用量の調整
  設定ファイルにて，並行して読み出しを行うスレッド数，書き込みを行うスレッド数それぞれの最大値を制限した．

またこれらのパラメータは，メンバーシップ管理の挙動に影響を与えない．

これらの調整により，1 マシンで Cassandra を最大 65 ノードまで起動することが可能となった．

3.3 通信量の測定

3.1 節のように多数ノードを 1 マシン上で起動した場合, 以下のように `tcpdump` を用いると，通信量の測定が可能である．

tcpdump の使用

通信量の測定は `tcpdump` を使用し，測定対象のノードで構築されるプライベートネットワーク上のパケットを記録する．具体的には TCP パケットのパケットサイズを記録した．また，パケットを重複して計測することを避けなければならない．実験では，`tcpdump` に以下の 2 つのフィルターを指定して計測を行った．具体的なネットワークの名前は，3.1 節で説明したものを用いる，

- `src net 10.20.0.0/16`
  このフィルターにより，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を取得する．これにより，このネットワーク以外からのパケットを計測しないことにな

Vol.2011-ARC-195 No.22
Vol.2011-OS-117 No.22
2011/4/14

る．つまり，この条件により，余計な ARP クエリなどの計測を避けることができるのである．

- `dst net 10.20.(マシン番号).0/24`
  このフィルターにより，送信先が 10.20.(マシン番号).0/24 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 コマンドを実行するマシン上で動作するノード宛のパケットだけを計測することができる．

この 2 つの条件により，他のマシンで動作するノードから，コマンドを実行したマシン上で動作するノードに送られてくるパケットを計測することができる．

通信量の推定

同一マシンに IP エイリアシングで作成したネットワークインタフェース間のパケットは tcpdump 等では取得できない．よって，同じマシン上で動作するノード間の通信は計測できない．そこで，計測可能な通信量からクラスタ全体の通信量を推定する．ここでは，メンバーシップ管理のようにノード間で発生する通信に対称性があると仮定できる場合の推定方法について説明する．

我々は以下の仮定が正しいものとして，計測した通信量から，クラスタ全体の通信量を推定する．

- 仮定: 同じマシン上で動作するノード間の通信量の平均と，異なるマシン上で動作するノード間の通信量の平均は等しい．

各ノードからは，同じマシン上で動作するノードも異なるマシン上で動作するノードも区別はない．メンバーシップ管理において，この仮定は妥当である．クラスタ全体の通信量の推定方法を，具体的に数式を用いて説明する．m 台の各マシンで n ノードずつ起動する（ただし，$m \geq 2$ とする）．つまり，システム全体で合計 $n \times m$ ノードが起動している．各マシンで計測した通信量の合計を T_t とし，クラスタ全体の通信量 T を推定する．T_t は，各ノードと他のマシン上で起動する $n \times (m-1)$ ノードとの間で発生する通信量の合計である。求める通信量 T は，各ノードとクラスタを構成する残りすべてのノードとの間で発生する通信量の合計であるので，上の仮定により，

$$T = T_t \times ((n \times m) - 1)/(n \times (m-1)) \quad (1)$$

と表せる．このようにノード間で発生する通信に対称性があると仮定できる場合には，通信量の推定が可能である．

またノード間で発生する通信に対称性がないときも，通信量の推定が可能である場合もある．具体的には，データ読み書きの通信量は推定可能である．各マシンで計測した通信量からメンバーシップ管理の通信量の差し引き，計測できない通信量について同様な推定を行うことで，データ読み書きのための通信量の推定が可能である．

4. 実験・評価

クラウドストレージのスケーラビリティ評価を行う．評価対象は，メンバーシップ管理の通信量とデータの読み書きの通信量とする．これらがノード数に応じてどのように変化するかを実験・評価した．データの読み書きの通信量は，ノード数に応じて増加しないことが予想できる，つまり，データの読み書きの通信量はスケーラビリティの制約にならないと予想されるが，これを実験で定量的に確認する．以下に実験環境を示す．

実 験 環 境

- 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
- ネットワーク: 1000BASE-T

4.1 実験 1: メンバーシップ管理の通信量の測定

実験 1 ではノード数に応じたメンバーシップ管理の通信量の変化を測定する．

4.1.1 実験内容および実験結果

まず，実験シナリオを説明する．用意したマシンは，計測の管理を行う “管理マシン” を 1 台と，Cassandra ノードが動作し，通信量の計測を行う “計測マシン” を 1 0 台である．各計測マシンを Machine1, Machine2, ..., Machine10 と名付ける．具体的な管理マシンの役割は，通信量計測の開始・終了を計測マシンに指示することと，計測マシンで測定した通信量を集約することである．計測マシンの役割は，Cassandra ノードの起動と通信量の計測である．このとき，1 台あたり複数の Cassandra ノードを立ち上げる．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，これを目指すノード数に到達するまで続ける．最初の Cassandra ノードを起動した瞬間から，各マシンで 10 分間計測を行った．計測後は，計測マシンにて個別に通信量を解析し管理マシンに解析結果を送信する．管理マシンは，送られてきた通信量を合計し，Cassandra クラスタ全体で発生する通信量の推定を行う．

次に実験結果を述べる．図 2 が，Cassandra ノード全体での，1 秒あたりのメンバーシッ

Vol.2011-ARC-195 No.22
Vol.2011-OS-117 No.22
2011/4/14

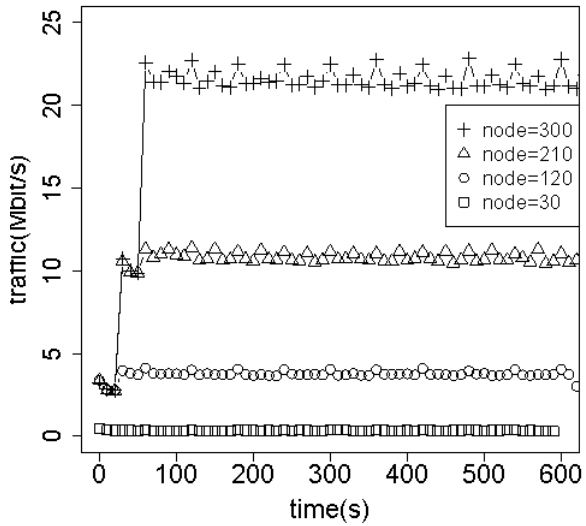

図 2 ノード数別通信量の時間変化

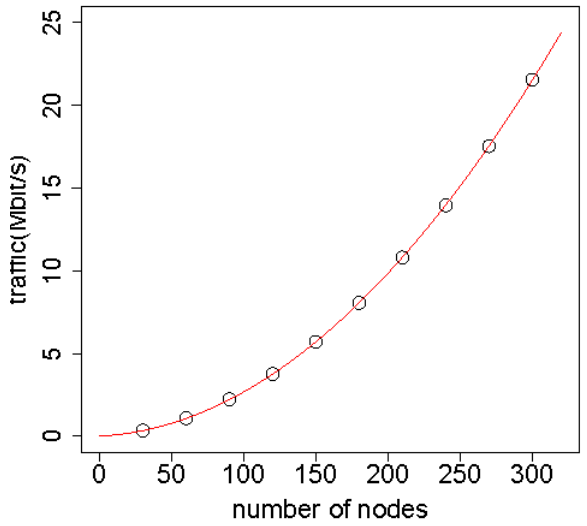

図 3 ノード数に応じた通信量の増加

プ管理の通信量の時間変化をノード数別に表している．ただし，$1M = 10^6$，$1K = 10^3$ である．図から，ノード数によらず，100 秒以降は通信量が安定していることがわかる．そこで，通信量が安定している時間帯、具体的には，実験開始から 200~300 秒後の 1 秒あたりの通信量の平均を算出し，ノード数別にプロットしたのが図 3 である．

4.1.2 評価・考察

実験 1 の評価・考察を述べる．

メンバーシップ管理の総通信量の見積もり

図 3 の曲線は，プロットした点から 2 次関数にてフィッティングしたものである．得られた関数は，非常によく合致していることが図からうかがえる．ノード数を n，通信量 (bit/s) を T とすれば，この関数は，

$$T = 224.6 \times n^2 + 4314.8 \times n \quad (2)$$

である．よって，総通信量は $O(n^2)$ であることがわかる．この関数から，ノード数をパラメータとして，メンバーシップ管理によ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．こうした値は，クラスタの構成を決める際などに役立つ．これについては後述する．

1 ノードあたりの通信量

また同様に，1 ノードあたりのメンバーシップ管理の通信量を見積もることが可能である．総通信量を Cassandra ノード数である n で割った値が，1 ノードあたりの通信量 T_i となる．つまり，

$$T_i = (224.6 \times n^2 + 4314.8 \times n)/n \quad (3)$$

$$= 224.6 \times n + 4314.8 \quad (4)$$

と，O(n) である．1 ノードあたりの通信量も定量的に明らかにできた．

複数のデータセンター上のクラスタにおける gossip プロトコルをベースとしたメンバーシップ管理の問題点

上述した通信量の関数はクラスタの構成を決める際に有用な指針となる．その有用性を表す例として，複数のデータセンターにまたがったクラスタを構成した場合の，ータセンター間の広域ネットワークでの通信負荷を考える．このように複数データセンター上でクラスタを構成することは特別なことではない．データセンターレベルの故障に耐性があり，非常に

Vol.2011-ARC-195 No.22
Vol.2011-OS-117 No.22
2011/4/14

図 4 二つのデータセンターをまたぐクラスタの構成

耐故障性があるクラスタ構成である．実際に Digg や Facebook といった大規模サービスを行う企業において，このようなクラスタ構成で運用が行われている．

ここでは簡単のため，データセンターは 2 つとし，各データセンター A，B に同数の Cassandra ノードが動作しているとする．この時，データセンター A，B 間の広域ネットワークで発生する通信量を考える．データセンター間での通信量 T^{AB} は，システム全体で発生する通信量を T とおくと，n が大きい時，$T/2$ とみなせる．一方，総通信量の関数 (2) から T は $\mathrm{O}(n^2)$ であるので，T^{AB} も $\mathrm{O}(n^2)$ である．具体的に $n = 2000$ とすれば，広域ネットワークで発生する通信量は，$454Mbps$ である．つまり，メンバーシップ管理というバックエンドの通信だけで，数百 Mbps の通信が必要になるのである．これは，さらにノード数が増加したときに通信のボトルネックとなり，性能の低下につながると予想される．より効率の良いプロトコルや，リンクごとのコストの違いを考慮できるメンバシップ管理プロトコルが望まれる．gossip の弾性（resilience）と配信木の効率の良さを両立したい．

メンバーシップ管理の通信量が $\mathbf{O}(n^2)$ である理由

メンバーシップ管理の通信量が $\mathrm{O}(n^2)$ である理由について考察する．総通信量 $T(bit/s)$ は，1 ノードあたりの通信量 (bit/s) を T_i，ノード数を n として，

$$T = T_i \times n \tag{5}$$

である．さらに 1 ノードが毎秒行う情報交換の回数を s，1 回の情報交換にかかる通信量 $t_{gossip}(bit)$ とする．1 ノードあたりの通信量 $T_i(bit/s)$ は，

$$T_i = s \times t_{gossip} \tag{6}$$

である．メンバー構成が安定した時を考える．Cassandra のメンバーシップ管理では，メンバー構成が安定したときは，毎秒 1~2 回のノード間の情報交換が行われる．よってノード間の情報交換の回数 s は，

$$s \leq 2 = O(1) \tag{7}$$

である．一方，1 回の情報交換にかかる通信量 $t_{gossip}(bit)$ は，Scuttlebutt プロトコルの特性により，安定時にはノード数 n に依存するので，

$$t_{gossip} = O(n) \tag{8}$$

よって式 (5) より，1 ノードあたりの通信量 $T_i(bit/s)$， 総通信量 $T(bit/s)$ は，

$$T_i = O(n) \tag{9}$$

$$T = O(n^2) \tag{10}$$

となることがわかる．以上の考察により，メンバーシップ管理の通信量が $\mathrm{O}(n^2)$ であることを示した．これは総通信量の関数 (2) を導出する際に，2 次関数でフィッティングを行ったことは，以上の考察を踏まえてのことである．

4.2 実験 2: データ読み書きの通信量の測定

実験 2 ではノード数に応じたデータ読み書きの通信量の変化を測定する．データの読み書きの通信量は，ノード数に応じて増加しないことが予想できる．つまり，データの読み書きの通信量はスケーラビリティの制約にならないと予想されるが，これを実験で定量的に確認する．

4.2.1 実験内容と実験結果

実験シナリオは，管理マシンを 1 台と計測マシンを 5 台を用意し，実験 1 と同様の手順で Cassandra の起動と通信量の測定を行った．データを読み書きは，Yahoo! Cloud Serving Benchmark (YCSB)[7] で行った．YCSB は Yahoo! Research が開発したクラウドストレージ用のベンチマークである．

実験結果を述べる．図 5 は，YCSB に用意されたワークロードを実行したときに，データ読み書きで発生する通信量 (Mbit/s) をノード数別に表した図である．ここでは，QPS $(queries/s)$ は 1000，更新時のデータサイズは，1 Kbyte で固定した．ただし，メンバーシップ管理の通信量を差し引いた後に，図に表現している．

4.2.2 考察と評価

実験 2 の結果について考察する． ここでは，ノード数が増えるに従い，データ読み書きの通信量は一定になったことを考察する．

Vol.2011-ARC-195 No.22
Vol.2011-OS-117 No.22
2011/4/14

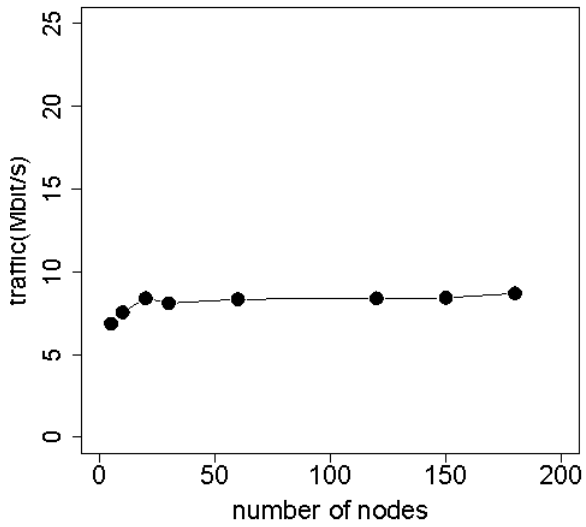

図 5 ノード数に応じたデータ読み書きの通信量の変化

図 5 から QPS を一定にしたとき，Cassandra のノード数 n に応じて，通信量の増加は緩やかになり一定の値に近づいていくことが確認できる．これについて考察する．まず，Cassandra におけるデータ読み書き時の通信手順を説明する．クライアントからデータの読み書きリクエスト要求を受け，データの受け渡しを行う Cassandra ノードを proxy ノードと呼ぶ．書き込み時には，proxy ノードが担当ノードへデータを転送し，担当ノードはデータ書き込みが成功したことを知らせる Ack を proxy ノードに返す．読み出し時には，proxy ノードが担当ノードに読み出しリクエストを転送し，受信した担当ノードは proxy ノードにデータを返す．このように，Cassandra では proxy ノードがクライアントとのデータのやり取りを仲介するのである．

負荷分散が適切に行われデータへのアクセスに偏りがないとすれば，proxy ノードがデータの担当ノードである確率は，ノード数を n として $1/n$ である．つまり，クライアントが要求するデータリクエストに対して，$(n-1)/n$ の確率で通信が発生することになる．よって，n が大きくなるとその確率は 1 に収束するので，ほぼ必ずリクエストに対して通信が発生するようになり，通信量はある一定値に近づいていく．

5. ま と め

本研究では，非集中型クラウドストレージにおいて，通信量の観点からスケーラビリティ評価を行った．その結果，gossip プロトコルをベースとしたメンバーシップ管理による通信量を定量的に計測することができ，ノード数を n としたとき，$O(n^2)$ であることを確認した．具体的に，通信量は $T = 224.6 \times n^2 + 4314.8 \times n$ という関数で表現でき，$n = 1000$ のとき全体で 229Mbps の通信が発生すると推定できる．この関数は，クラスタ設計時に有用な知見となる．特に複数のデータセンター上でクラスタを構成するとき，データセンター間の広域ネットワークで発生する通信量が $O(n^2)$ であり，ノード数が増えることで性能の低下につながることを指摘した．また，実験で適用したスケーラビリティの評価手法も本研究の成果である．本研究では Apache Cassandra を対象として通信量の測定を行ったが，他のソフトウェアについても計測可能な汎用的な測定手法である．

今後の課題を 2 つ述べる．1 つ目は，データセンター間のリンクを意識したメンバーシップ管理プロトコルの提案である．複数のデータセンター上でクラスタを構成することが主流の現在において，データセンターを結ぶ広域ネットワーク上の通信を考慮したメンバーシップ管理が望まれる．例えば gossip プロトコルを応用するのであれば，別のデータセンターにあるノードに対して，重複する情報の通信を減らせば，データセンター間のリンクで発生する通信を削減できると考える．

2 つ目は，gossip プロトコルをベースとしたメンバーシップ管理を通信量以外の観点からスケーラビリティ評価を行うことである．CPU 負荷，およびすべてのノードに情報が伝搬するまでの時間という観点からのスケーラビリティ評価は必要である．これらがどの程度スケーラビリティの制約になるかを評価することで，システムをよりスケールさせていく上で重要な指針となる．

謝辞 本研究は科研費（22680005）の助成を受けたものである．

参 考 文 献

1) 丸山不二夫, 首藤一幸: クラウドの技術, pp.6–13, 株式会社アスキー・メディアワークス (2009).
2) Miguel Matos, António Sousa, Jose Pereira, Rui Oliveira, Eric Deliot and Paul Murray: CLON: Overlay Networks and Gossip Protocols for Cloud Environments, *Proc. 3rd Workshop on Dependable Distributed Data Management (WDDM '09)* (2009).

Vol.2011-ARC-195 No.22
Vol.2011-OS-117 No.22
2011/4/14

3) Avinash Lakchman and Prashant Malik: Cassandra - A Decentralized Structured Storage System, *Proc. 3rd Int'l Workshop Large Scale Distributed Systems and Middleware (LADIS '09)* (2009).
4) Marcia Pasin, St'phane Fontaine and Sara Bouchenak: Failure Detection in Large Scale Systems: a Survey, *Proc. NOMS Workshops 2008* (2008).
5) Robbert van Renesse, Dan Dumitriu, Valient Gough and Chris Thomas: Efficient Reconciliation and Flow Control for Anti-Entropy, *Proc. 2rd Int'l Workshop Large Scale Distributed Systems and Middleware (LADIS '08)* (2008).
6) The Apache Cassandra Project, http://cassandra.apache.org/index.html (2010).
7) Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan and Russell Sears: Benchmarking Cloud Serving Systems with YCSB, *Proc. Symposium on Cloud Computing (SOCC 2010)* (2010).
8) 田浦健次朗: GXP: An Interactive Shell for the Grid Environment, *Proc. Int'l Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA 2004)* (2004).
9) Giuseppe DeCandia, Deniz Hastoruna, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakchman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels: Dynamo: Amazon's Highly Available Key-value Store, *Proc. 21st Symposium on Operating Systems Principles (SOSP 2007)* (2007).