MapReduceによる大規模分散システムのシミュレーション

杉野　好宏†　　華井　雅俊†　　首藤　一幸†

†東京工業大学

概要　インターネット上では様々な分散システムが動作している．これらの開発において，その挙動を把握することが重要であり，そのためにはシミュレーションが必要である．既存のシミュレータは100万ノード程度までの分散システムを扱うことができる．しかし現在，1,000万ノード規模の分散システムがインターネット上で動作している．また，今後，より大規模な分散システムの出現が予想され，シミュレータで実験可能なノード数の増大が急がれる．この問題の解決をするために，本研究では汎用分散処理システムを用いることを提案する．今回，汎用分散処理フレームワークであるMapReduceを用いた手法を提案する．この提案に基づいてシミュレータを開発した．そして，それを用いてP2Pファイル共有のプロトコルであるGnutellaを実装し，その通信シミュレーションを行った．シミュレーション結果の妥当性，および，スケーラビリティを評価し，本提案手法の妥当性と今後の課題を議論する．

Simulation of Large-Scale Distributed Systems with MapReduce

Takahiro SUGINO†, Masatoshi HANAI†, and Kazuyuki SHUDO†

† Tokyo Institute of Technology

Abstract　Distributed systems research requires a supporting simulator. Today, there are distributed systems with a scale of from millions to ten millions working on Internet. To research these existing and emerging systems we need experimental means supporting the same scale. But today's simulators can simulate only a hundred thousands nodes. We propose a method to simulate such a large-scale with MapReduce. General-purpose distributed processing systems such as MapReduce provide high scalability and maturity due to their active research and development. This paper shows the method and the experimental results that an implemented simulator produced by simulating Gnutella protocol. The results are preliminary but they show that the proposed method achieves simulation of a distributed system.

1. はじめに

インターネット上では様々な分散システムが動作している．その開発においては，実験を欠かすことができない．なぜなら，実験をせずに運用を開始すれば，運用先の環境を意図せず破壊してしまう可能性があるからである．実験には，実環境に近い環境を用意して行う方法と，シミュレータを用いて行う方法の2つがある．

既存の分散システムシミュレータは100万ノード程度までの分散システムを扱うことができている[1]．しかし，現在動作している分散システムであるBitTorrent DHT[2]の規模は，それよりもはるかに大きい1,000万ノードに到達している．このとおり，すでに既存のシミュレータでは実験を行うことのできない規模の分散システムが登場しているのである．さらにInternet of Thingsの拡大に伴い，インターネットに接続される機器の数は現在の125億から2020年には500億まで増えると予測されている[3]．この通り，分散システムは今後，より大きくなっていくことが予想される．それにしたがって，実環境に近い環境を用意して実験を行うことは困難になっていく．ゆえに，シミュレータによる実験がこれまで以上に重要なものとなっていくと考えられる．シミュレータで扱うことのできるノード数の増大が急務である．

1,000万ノード以上の規模を目指す上で，シミュレータを分散システムとして実装することは避けることができない．そこで本研究では，汎用分散システムを用いたシミュレーション手法を提案する．様々な場面の運用実績がある汎用分散処理システムをベースにすることにより，耐故障性を得ることができる．また，汎用であるために研究や開発が活発であり，システムの

改良による恩恵を，直接的に享受することができる．

今回，我々は，汎用分散処理システムとして Hadoop MapReduce [4] を用いる．Hadoop は数多くの運用実績を持ち，高い耐故障性やスケーラビリティが期待できる．

本稿は次の構成をとる．第 2 章で関連研究を説明し，第 3 章では提案シミュレータが分散システム上のノードとその間の通信をどのようにモデル化し，MapReduce 上で扱うのか述べる．第 4 章でシミュレーションを行うアルゴリズムの記述の仕方，第 5 章では実験および考察，第 6 章では本提案手法の妥当性を論じ，今後の課題を述べる．

2. 関 連 研 究

ns-2 [5]，ns-3 [6]，PeerSim [7]，Overlay Weaver [1, 8] など，さまざまなシミュレータが開発されてきた [9]．

ns-2 [5] はネットワーク研究を目的とした，離散事象シミュレーションを行うことができるオープンソース・ソフトウェアである．有線や無線ネットワークにおける TCP や，ルーティングおよびマルチキャストプロトコルをサポートしており，数千ノード規模のシミュレーションが可能である．

ns-3 [6] は，ns-2 のアーキテクチャおよびモデルを改善すべく開発されたオープンソース・ソフトウェアである．数千ノード規模の離散事象シミュレーションを行うことが可能であり，ネットワーク・プロトコルやアドホック・ネットワークをシミュレートできる．

上記 2 つのシミュレータは，アンダーレイネットワークをシミュレーション対象としており，その延長として分散システムのシミュレーションを行うことが可能である．アンダーレイネットワークをシミュレートすることに計算機のリソースを大きく要するため，これがシミュレーション規模拡大のボトルネックとなりうる．本研究では，シミュレーション規模の追求を優先に取り組んでいる．アンダーレイネットワークの詳細なシミュレートは，今後の課題である．

PeerSim [7] はクエリ・サイクルおよび離散事象シミュレーションを行うことができるオープンソース・ソフトウェアである．P2P システムを対象としており，アンダーレイネットワークはシミュレート対象としていない．オーバーレイネットワーク上での P2P システムをシミュレートすることができ，クエリ・サイクル・シミュレーションでは 100 万ノードでのシミュレーションを行うことが可能である．

Overlay Weaver [1, 8] はオーバレイネットワーク構築キットツールであり，シミュレーションのために実装したプログラムを変更なしで実ネットワーク上で動作させることが可能である．オーバーレイネットワークにおける分散システムをシミュレート対象としており，今現在，最大で 100 万ノード程度のシミュレートが可能である．また，計算機数台での分散シミュレーションも可能となっている．

これら既存手法では，計算機を数十台から数百台といった規模の分散シミュレーションは想定していない．この規模では，ただ複数台での分散シミュレーションが可能であればよいというわけではなく，耐故障性が問題となってくる．1 回のシミュレーションには数日かかることも普通なので，1 台でも停止するとそれまでのシミュレーションが無駄になるのでは使いものにはならない．したがって，シミュレータ自体が耐故障性を備える必要がある．耐故障性なしには台数を増やせない，つまりスケールしない，ということである．

3. 提 案 手 法

本研究では汎用分散処理システムを用いたシミュレーション手法を提案する．

シミュレータ自体を分散システムとするのは，大規模なシミュレーションを行うためである．そのためには，計算機の台数に応じて規模を拡大できること，つまりスケーラビリティが求められる．また，シミュレータがスケールするためには，耐故障性が必要となる．例えば，平均故障間隔 15,000 時間の計算機を用いたとして，故障なしで 24 時間動作し続ける確率は，1 台であれば 99.8%であるが，これが 100 台となると 85.3%，500 台であれば 44.9%となってしまう．

これらの要件を満たすために，本研究では汎用分散システムを利用する．汎用分散処理システムは，システム設計の時点から複数台の計算機による並列処理を想定して設計されている．そのため元来スケーラビリティの高い設計となっており，フェイルオーバー機能が実装されているなど，耐故障性も備えている．さらに，汎用であるため，シミュレーション以外にも様々な用途で利用されている．広く用いられていることから，ソフトウェアとして成熟度の高さが期待できる．そして，処理システムが改良されたとき，本シミュレータはそのままその恩恵を享受することができる．汎用分散処理システムとして，本研究では Hadoop MapReduce を利用する．オープンソースソフトウェアであり，かつ，広く用いられているため，ソフトウェアとしての成熟が期待できる．大規模な実用例として，4,500 台での運用実績がある [10]．以下ではまず，本研究で利用した Hadoop MapReduce の説明し，その後，具体的な提案手法を述べる．

3.1 Hadoop MapReduce

MapReduce [11] は 2004 年に発表された，巨大なデータの処理を目的としたプログラミングモデルであり，汎用分散処理フレームワークである．Hadoop MapReduce [4] は，これを元にしたオープンソースのクローン実装である．map 関数と reduce 関数の 2 つを記述することにより，容易に分散処理プログラムを書くことができる．数百 TB から数 PB という膨大なデータを数多くの計算機に分散して並列に処理する，入力と出力は Hadoop 分散ファイルシステムに記録される．また，フェイルオーバーの機能も実装されており高い耐故障性を備える．行われる処理は図 1 のように Map 処理, Shuffle 処理, Reduce 処理の 3 つのフェーズに分けられる．これらの一連の処理を 1 回行うことを iteration と呼ぶ．1 回の iteration では不可能なデータの処理を，複数回の iteration により処理することも可能である．以下では，Map 処理, Shuffle 処理, Reduce 処理の説明をする．

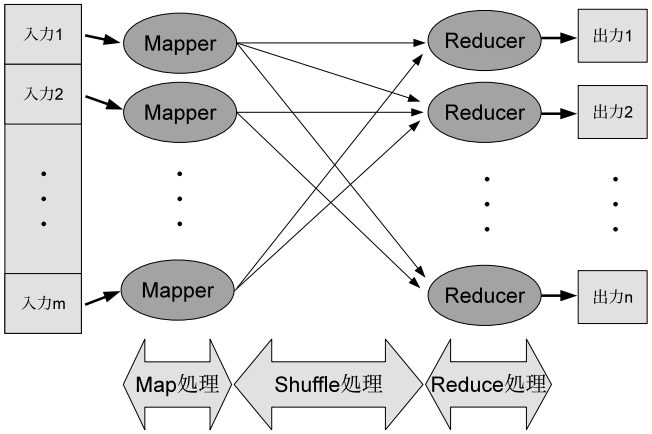

図 1　MapReduce の処理フロー
Fig. 1　Prosecc flow of MapReduce

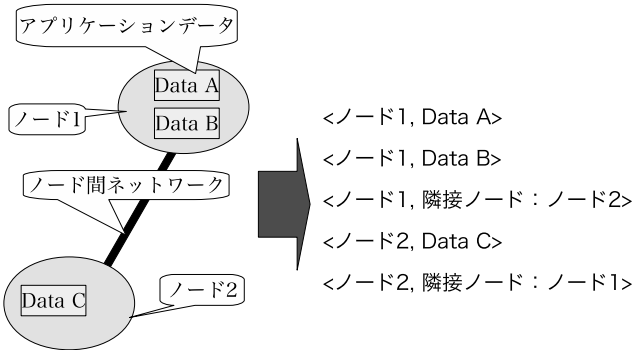

図 2　key-value ペアによる分散システムのモデル化
Fig. 2　Modelling of a distributed system with key-value pairs

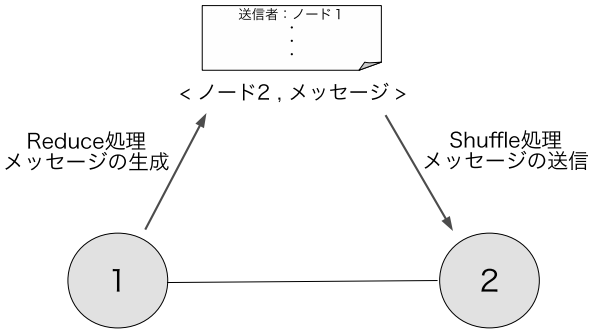

図 3　メッセージ送信の例
Fig. 3　An example of sending a message

Map 処理　Map 処理を行う Mapper は 64MB 程度に分割された入力データを受け取り，map 関数に従って key-value のペアを生成する．それぞれの Mapper は独立に動き，分割されたデータは並列に処理される．扱われるデータには局所性はない．

Shuffle 処理　Mapper により生成された key-value ペアを，Reduce 処理を行う Reducer に分配する．その際に，key-value ペアは同じ key ごとに集められ，特定の Reducer に渡される．また，処理される key-value ペアは任意の順序にソートすることができる．

Reduce 処理　Shuffle 処理により分配された key-value ペアを reduce 関数に従って処理を行い，その結果をデータとしてディスクに出力する．Shuffle 処理により，扱われる key-value ペアには key に関して局所性がある．

3.2　分散システムの表現

MapReduce のプログラミングモデルにおいて，どのように分散システムを表現するのか述べる．現実の分散システムは，各ノードがそれぞれの持つ情報に基づき自律的に動き，インターネットを介してノード間で通信を行い，相互に作用しながら動作するものである．したがってシミュレーションでは，各ノードの保持する情報とそれに基づいた動作，およびノード間通信を表現することが必要である．これらを 1 台の計算機の上で表現可能であるのは明らかであるが，複数の計算機によるシミュレーションにおいても，同様の表現の実現が求められる．分散システムにおいてノードが持つ情報，およびメッセージ送受信をどのように表現するのか，次節以降で議論する．

3.2.1　ノードの表現

分散システムにおいてノードが持つ情報を，MapReduce を利用したシミュレータの中でどのようにモデル化したか述べる．次節で詳しく述べるが，メッセージ送信は Shuffle 処理により表現することが可能である．また，シミュレートする分散システムの中で，各ノードが実行する操作は，データに局所性のある Reduce 処理において操作を行うことで表現することができる．それに応じてシミュレータ上における分散システムを key-value ペアにより表現した．また，モデル化とは関係はないが，今回の実装において Map 処理では，入力ファイルから key-value ペアを生成する処理を行う．表 1 に，分散システムの各要素の，key-value ペアを用いた表現を示す．key-value ペアの key には，そのノードの IP アドレスを，value には表 1 で示した情報を 1 つ割り当てる．これにより， key-value ペアが Shuffle 処理を経て Reducer に割り当てられる際に，同じ IP アドレスを持つ key-value ペアを同一の Reducer に集めることができる．こうして，Reduce 処理において，そのノードの持つ全ての情報を利用して処理を行うことが可能になった．図 2 左に示したノードおよびノード間ネットワークは，key-value ペアを用いて図 2 右のように表現される．

3.2.2　メッセージ送受信の表現

メッセージ送受信をどのようにモデル化したのかを説明する．メッセージ送受信を表現するためには，送信されたメッセージが，宛先ノードの保持する全ての情報と一緒にまとめられる必要がある．これを実現するために，メッセージを送信するノードは Reduce 処理の際に，送信の宛先ノードの IP アドレスを key に，メッセージボディを value に割り当てた key-value ペアを生成する．これにより，Shuffle 処理を経て，この key-value ペアは宛先と同じ IP アドレスを持つ key-value ペア毎に，同一の Reducer に割り振られる．Reducer では，割り当てられた key-value ペアを解釈し，その種類ごとに応じた処理が行われる．その中で，メッセージの受信処理を行うことが可能である．例えば，ノード 1 からノード 2 へのメッセージ送受信は表 3 のように表現される．

しかし，これだけでは正確にメッセージ送受信を表現するこ

表 1　key-value ペアでの分散システムの表現
Table 1　Expression of a distributed system with key-value pairs

key	value	表現対象
IP アドレス	隣接ノード	ノードが通信可能なノードの宛先
IP アドレス	アプリケーションデータ	アプリケーションが保持するデータ
IP アドレス	スケジュール	ノードが実行すべき処理とそのタイミング
IP アドレス	メッセージボディ	メッセージ

とができない．key-value ペアを処理する順序を考慮する必要がある．例えば，アプリケーションデータ A を保持するノード 1 が，メッセージ B を受け取る場合を考える．ノード 1 は A の有無により B の処理が異なるものとする．A を表す key-value ペアが B を表すものよりも先に処理が行われた場合，ノード 1 は A の保持を確認した上で B の処理を行うことができる．しかし，処理される順番が逆であった場合，ノード 1 は B の処理を行う際に A の保持を確認することができず，期待した振る舞いをしないということが起こってしまう．これを防ぐために key に priority という属性を追加し，この値に基づいてソートを行い，メッセージを表す key-value ペアはノード情報よりも後で処理を行うようにした．これにより，送信されたメッセージを宛先ノードの受信メッセージとして正確に処理することができるようになった．

このようにして，1 回のメッセージ送信を 1 回の iteration により実現することができた．シミュレーションを行う際には，複数回の iteration によりメッセージ送信を表現する．

4. シミュレート対象の記述

シミュレート対象の記述方法は容易であることが望ましい．また，MapReduce を意識することなく記述できることが望ましい．そこで，シミュレーション対象のひな形となる抽象クラスを用意し，いくつかのメソッドを実装することでシミュレート対象を記述できるようにした．また，記述を容易にするために，メッセージ送信等を行うメソッドを用意している．これによって，key-value ペアといった MapReduce の仕組みを意識することなく記述ができる．

シミュレート対象を記述する際には，各処理のハンドラである次の 2 つのメソッドを実装する．

messageHandler(message)
ノードが新たに受信したメッセージに応じた処理を行う．引数 message は受信したメッセージを表す．

scheduleHandler(schedule)
ノードに対し，実行される時刻が指定された処理を行う．引数 schedule はノードが実行すべき処理のスケジュールを表す．

ネットワークのトポロジや，メッセージ送信などのノードの動作は，入力ファイルにより与える．この入力ファイルはテキスト形式で，出力ファイルも同様の形式である．本研究では，これを用いて Gnutella プロトコル [12]（付録 A）に基づくアルゴリズムを実装し，`Query` メッセージ, `QueryHit` メッセージの送受信をシミュレートした．この擬似コードは図 4 に示す．

```
Gnutella extends AbstractAlgorithm {
  messageHandler(message, node){
    if( messageがQueryで、同じQueryを受信していない ){
      if( 検索されているデータをnodeが持っていたら ){
        messageの送信ノードにQueryHitを送信
      }
      nodeの隣接ノードにQueryを転送
    }
    else if( messageがQueryHitである ){
      if( nodeがQueryHitに対応するQueryを受信している ){
        そのQueryの送信ノードに対しQueryHitを転送
      }
      else if( nodeが最初のQueryの送信ノードである ){
        データを発見
      }
    }
  }

  scheduleHandler(schedule, node) {
    if( scheduleがデータ検索を表す ) {
      全隣接ノードにQueryを送信
    }
  }
}
```

図 4　Gnutella の擬似コード
Fig. 4　Pseudocode of Gnutella

5. 実　　験

本シミュレータを用いて Gnutella の実装を行い，ファイル検索時のメッセージ送受信をシミュレートした．実験では，各ノードに IP アドレスおよび各ノード固有のアプリケーションデータを与える．`Query` メッセージの TTL の初期値は 7 なので，検索を開始したノードに `QueryHit` メッセージによる返答があるまでに，最大で 14 回メッセージの転送が行われる．Reduce 処理時にメッセージを生成し，次の Map 処理および Shuffle 処理においてメッセージの送信が行われるので，今回の実験では iteration を 15 回繰り返した．

5.1　実験 1：2 次元メッシュ上でのデータ検索

本シミュレータを用いたシミュレーションで得られた結果の妥当性を評価した．ネットワークのトポロジは，正方形の 2 次元メッシュとした．2 次元メッシュの 1 辺のノード数を 10 から 1,000 の間で変化させ，アプリケーションデータの検索シミュレーションを行った．10 から 100 の間では 10 ずつ，100 から 1,000 の間では 100 ずつ変化をさせた．シミュレーションを行ったネットワークのノード数は 100 から 1,000,000 である．本実

表 2　5.1 節の実験環境
Table 2　Experimental settings for Section 5.1

OS	Mac OS 10.7.4
Java 仮想マシン	Java SE 6 update 33
CPU	Intel(R) Core i7 2.4 GHz
MapReduce 実装	Apache Hadoop 0.20.2
mapred.child.java.opts	-Xmx200m
io.sort.mb	100MB
Hadoop 実行モード	擬似分散モード

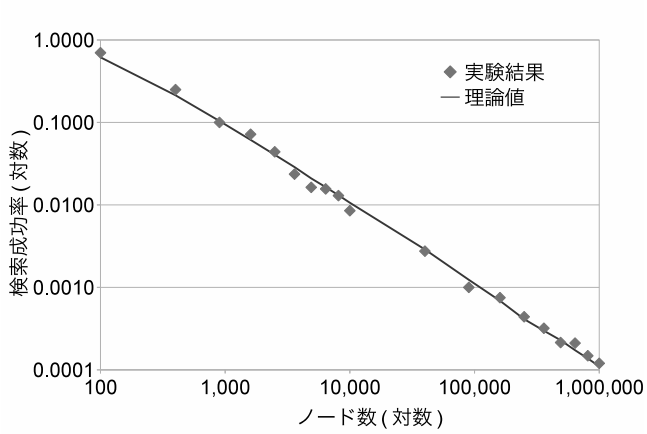

図 5　ノード数と検索成功率
Fig. 5　Number of nodes and a success rate in data search

験では，データ検索を一定の割合のノードが行うものとした．単純なトポロジであるため，Query メッセージによる検索成功率の理論値を求めることが可能である．シミュレーション結果と理論値との比較により，今回のシミュレーションにおいて妥当な結果を得られるのか評価した．本実験の実験環境は図 2 に示す．

図 5 に実験結果を示す．検索成功率は理論値とほぼ等しい値となった．したがって，今回のシミュレーションで得られた結果は妥当なものであると考えている．

5.2　実験 2：BA モデル複雑ネットワーク上でのデータ検索

本シミュレータを用いて，BA モデル [13] に基づくネットワークにおいて Gnutella のシミュレーションを行った．BA モデルとは，次のようなアルゴリズムによりグラフを生成するものである．

（1） m ノードからなる完全グラフを生成する．
（2） (1) で生成した完全グラフに新しいノードを 1 つ追加する．追加されるノードは，既にグラフ上に存在するノードから異なる m 個を選び，それらに対してエッジを張る．グラフ上のノードは，その次数に比例する確率により選ばれる．
（3） (2) を繰り返し行い，指定したノード数に達するまで続ける．

BA モデルにより生成したネットワークは，ノード数 N に対して平均経路長 l が

表 3　5.2 節の実験環境
Table 3　Experimental settings for Section 5.2

OS	Linux 3.3.4-3
Java 仮想マシン	Java SE 7 Update 4
CPU	Intel(R) Xeon(R) E5620(2.40 GHz) × 2
MapReduce 実装	Apache Hadoop 0.20.2
mapred.child.java.opts	-Xmx200m
io.sort.mb	100MB
Hadoop 実行モード	擬似分散モード

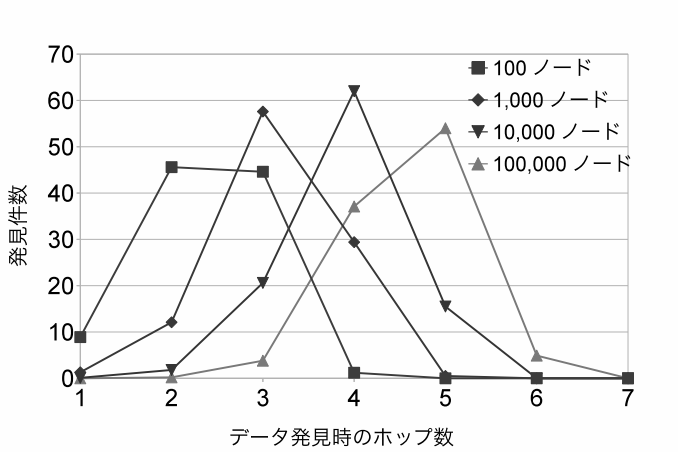

図 6　データ発見時の Query メッセージのホップ数とデータ発見数
Fig. 6　Number of Query hops and number of Queryhit

$$l = \begin{cases} O(\log N) & (m = 1) \\ O(\frac{\log N}{\log \log N}) & (m \geqq 2) \end{cases} \quad (1)$$

であるというスモールワールド性，またはウルトラスモールワールド性を備える．また，ノードの次数 k の分布 $p(k)$ が

$$p(k) \propto k^{-\gamma} \quad (\gamma > 0) \quad (2)$$

となるスケールフリー性も備える(注1)．

シミュレーション内容は，同時に 100 件のデータ検索を行うというもので，初期ノード数 m は $m = 4$，ノード数 N は，$N = 10^2, 10^3, 10^4, 10^5$ とした．各ノード数に対して，ネットワークのトポロジとノードの動作を変えて 10 回のシミュレーションを行った．図 3 に本実験の実験環境を示す．図 6 に実験結果を示す．図 6 からノード数が 10 倍になった時にデータ発見時の Query メッセージのホップ数の平均が，ほぼ等間隔に推移していることが見て取れる．この結果から，このネットワークの平均最短経路長が，ノード数 N に対して $O(\log N)$ 以下であることが確認できる．また，今回行った全ての実験において，100 件の検索は全て成功している．これらは，数式 (1) に示した BA モデルネットワークのスモールワールド性が表れたものと言える．

5.3　実験 3：スケーラビリティの評価

計算機の台数に応じて，シミュレーションの実行時間の短縮

(注1)：BA モデルにおいては $p(k) \propto k^{-3}$ である．

表 4　5.3 節の実験環境
Table 4　Experimental settings for Section 5.3

OS	Linux 3.3.4-3
Java 仮想マシン	Java SE 7 Update 4
CPU	Intel(R) Xeon(R) E5620(2.40 GHz) × 2
MapReduce 実装	Apache Hadoop 0.20.2
mapred.child.java.opts	-Xmx1000m
io.sort.mb	100MB
Hadoop 実行モード	完全分散モード

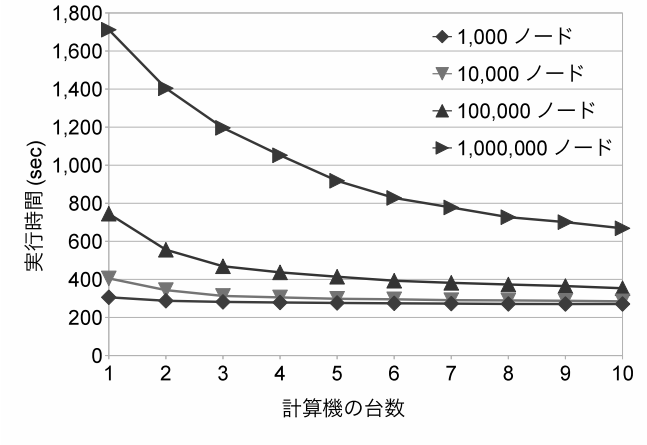

図 7　計算機台数と実行時間
Fig. 7　Number of simulating computers and execution time

および規模の増大が可能であることを確認するために，計算機台数とシミュレーションの実行時間の関係を調べた．シミュレーションの内容は，$m = 1$，$N = 10^3, 10^4, 10^5, 10^6$ の BA モデルネットワークにおいて，100 件のデータ検索を同時に行うものとした．本実験では表 4 に示した計算機を 10 台用意し，シミュレーションを実行する計算機の台数を変化させたときの実行時間を測定した．シミュレーションにおいて，各計算機の内蔵ディスクは使用せず，Hadoop 分散ファイルシステムを利用した．この際，1 台の計算機あたり 1 個のプロセスを起動させてシミュレーションを行った．

図 7 に実験結果を示す．この結果から，シミュレーションを行う計算機の台数を増やしたとき，実行時間が減少することを確認できた．したがって，台数を増やすことでシミュレータの性能がスケールアウトしていると言うことができる．しかし，台数を 1 台から 2 台にした時の高速化の程度は，処理速度 2 倍には遠く及ばず，そして 1,000,000 ノードでの実験結果では，計算機の台数を増やしたとしても実行時間が 600 秒程度でまでしか縮まらなかった．しかしこの結果は，計算機 10 台で 1,000,000 ノードの実験を行った場合とほぼ同じであることがわかる．計算機 1 台あたりの処理ノード数が同じであることから，1,000,000 ノードの場合においても実験を行う計算機の台数を 10 台よりも多く増やすことで，さらなる高速化が望めるものと考えられる．また，ノード数が少ない場合には，台数を増やすことで実行時間がほとんど変化していない．Hadoop 上でのタスク起動に一定の時間を要するため，それが実行時間の下限となり，台数によらず実行時間が横ばいとなっているものと考えられる．

6. まとめと今後の課題

本研究では，汎用分散処理システムである MapReduce を用いた分散システムのシミュレーション手法を提案し，その提案に基いてシミュレータを実装した．そして，それを用いて P2P ファイル共有システムのプロトコルである Gnutella のシミュレーションを行った．

実験結果から，今回シミュレートした分散システムについて，シミュレート結果の妥当性と，提案手法により，1 台の計算機により 100 万ノードのシミュレーションが可能であることが確認できた．

今後の課題として 3 点が挙げられる．1 点目は，時間の取り扱い，および，取り扱い手法の評価である．提案手法では，シミュレート対象のノード間通信は，MapReduce の Shuffle 処理のタイミングでまとめて行われる．これにより，メッセージの受信は Reduce 処理の際にまとめて発生するため，本来発生すべき順番とは違った順で発生することがある．受信の発生順をあるべき順とするためには，単純には，1 回の Shuffle 処理で取り扱う送受信を 1 つだけに限定すればよい．しかし，それではシミュレーション性能が非常に低くなってしまうことが予想される．シミュレーション時間の楽観的な取り扱い手法により，1 回の iteration で処理できるメッセージ数を増やすことを検討中であり，実現の目処が立っている．今後，この手法のさらなる検討と評価を行っていく．

2 点目は，シミュレーションの実行時間の短縮である．Hadoop MapReduce はその設計において，iteration を何度も繰り返す際の性能をあまり重視していない．iteration のたびに key-value ペアを分散ファイルシステム HDFS に書き込み，次の Map 処理の際にはそれを読み出す．その入出力が，iteration を繰り返す際に時間を要し，iteration ごとのオーバヘッドとなっている．また，1 回の iteration を終える際に，必ず数秒の無駄な待ち時間が発生するという不具合があり，2012 年 2 月にようやく解消された [14]．この不具合が長い間放置されていたことも， 繰り返し時の性能が軽視されていたことの証左である．とはいえ，Hadoop MapReduce も改善されてきており，また，SSS [15] といった他の MapReduce 実行系を用いることで改善できる可能性もある．

3 点目は，扱うことのできるシミュレーションの規模の拡大である．本研究では 10 台の計算機を用いて実験を行ったが，さらに台数を増やすことで，より大規模なシミュレーションをより高速に行えると見込んでいる．現在，10 台の計算機を用いて，1 億ノード（初期ノード数 2）の BA モデルネットワーク上での 100 件のデータ検索をシミュレートできている．このシミュレーションの際にネットワーク上で送信される `Query` メッセージの数は，数十億程度に達している．今後は，1 億ノードよりも大規模な実験を行い，そのときのシミュレータの性能評価を行っていく．そして，実験規模の拡大を制限する要因が何

かを調べていく必要がある．現時点では，大量のメッセージ受信などにより，1 つのノードが保持する情報が膨大なものとなり，処理を行う Reducer が使用できるメモリ全てを使い尽くしてしまう場合などが制限の要因になりうると考えている．規模の拡大を制限する要因を調べ，検討し，どの程度までシミュレーション規模を拡大できるのか評価していく．

謝辞　本研究は科研費 (24650025) および独立行政法人情報通信機構（NICT）委託研究「新世代ネットワークを支えるネットワーク仮想化基盤技術の研究開発」の助成を受けたものである．

文　献

[1] K. Shudo, Overlay Weaver: An Overlay Construction Toolkit, `http://overlayweaver.sourceforge.net/`.
[2] Decentralized Systems and Network Services Research Group in Karlsruhe Institute of Technology, Live Monitoring of the BitTorrent DHT. `http://dsn.tm.unikarlsruhe.de/english/2936.php`.
[3] Cisco Systems, Inc, Internet of Things, D. Evans, `http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf`, 2011.
[4] Welcome to Hadoop^{TM} MapReduce! , `http://hadoop.apache.org/mapreduce/`.
[5] ns-2, `http://www.isi.edu/nsnam/ns/`.
[6] ns-3, `http://www.nsnam.org/`.
[7] Alberto Montresor and Mark Jelasity. PeerSim: A scalable P2P simulator. Proc. The 9th Int. Conference on Peer-to-Peer (P2P'09), pages 99-100. Seattle, WA, September 2009.
[8] K. Shudo, Y. Tanaka, S. Sekiguchi: Overlay Weaver: An Overlay Construction Toolkit, Computer Communications, Vol. 31, No. 2, pp.402-412, 2008.
[9] Towards Yet Another Peer-to-Peer Simulator (S. Naicken, A. Basu, B. Livingston, S. Rodhetbhai, I. Wakeman), Proc. The Fourth International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks (HET-NETs), 2006.
[10] Apache Hadoop Project: Hadoop Wiki - PoweredBy, 2012. `http://wiki.apache.org/hadoop/PoweredBy`.
[11] J. Dean, and S. Ghemawat: MapReduce: Simplified Data Processing on Large Clusters, Proc. OSDI ' 04, 2004.
[12] The Annotated Gnutella Protocol Specification v0.4, `http://rfc-gnutella.sourceforge.net/developer/stable/index.html`.
[13] A. -L. Barabasi and R. Albert, Emergence of scaling in random networks, Science 286, 509, 1999
[14] [#MAPREDUCE-3809] Tasks may take upto 3 seconds to exit after completion, `https://issues.apache.org/jira/browse/MAPREDUCE-3809`, 2012.
[15] H. Ogawa, H. Nakada, R. Takano, and T. Kudoh: SSS: An Implementation of Key-Value Store Based MapReduce Framework, Proc. CloudCom 2010, pp.754-761, 2010.

付　録

A　分散アルゴリズムの例：Gnutella

Gnutella は，Pure P2P のファイル共有ソフトウェアであり，その分散検索プロトコルである．Gnutella では，通信メッセージとルーティング方法が規定されている．ノード間通信には `Ping`, `Pong`, `Query`, `QueryHit`, `Push` という 5 つのメッセージが用いられ，それによりファイルの検索などを行う．以下ではファイル検索に用いられる `Query` および `QueryHit` の説明する．

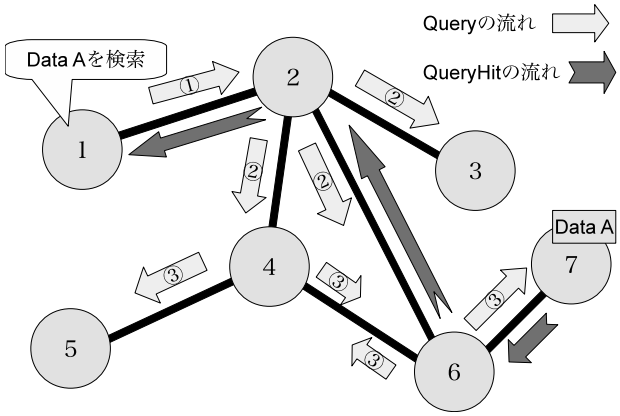

図 A·1　`Query` および `QueryHit` の送信の例
Fig. A·1　An example of sending `Query` and `QueryHit`

`Query` はアプリケーションデータを検索する際に送信される．アプリケーションデータの検索をしたいノードは，全ての隣接ノードに対して `Query` を送信する．`Query` を受け取ったノードは，その `Query` の送信ノードを除く全ての隣接ノードに対して `Query` の転送を行う．`Query` には生存時間 (TTL) が設定されており，転送が行われるたびに 1 だけ引かれる．TTL が 0 となったメッセージは削除される．Gnutella では TTL の初期値は 7 である．

`QueryHit` は，検索されたアプリケーションデータが見つかったことを知らせるメッセージである．`Query` を受信したノードが，検索していたアプリケーションデータを持っているときは，その `Query` の送信ノードに対して，`QueryHit` を送信する．`QueryHit` を受信したノードは，それに対応する `Query` の送信ノードに対して `QueryHit` の転送を行う．対応する `Query` を受信していなかったら `QueryHit` は破棄される．このようにして `QueryHit` が最初に `Query` の送信を行ったノードに到達すると，そのノードはネットワーク上に探していたアプリケーションデータがあるということを知る．

図 A·1 はネットワーク上でファイル検索を行った場合のメッセージ送信の様子を表している．ノード 1 が Data A の検索を行うために，`Query` を隣接ノードであるノード 2 へ送信する．`Query` を受け取ったノード 2 も同様に，ノード 1 を除く全ての隣接ノードに対して `Query` を送信する．これを繰り返すことで，`Query` はノード 1 から 2，6 を経由して，Data A を持つノード 7 へと送信される．ノード 7 は，受け取った `Query` の送信者であるノード 6 に対して `QueryHit` を返送する．ノード 6 は同様に，受け取った `Query` の送信者であるノード 2 に対して `QueryHit` を返送する．同じく，ノード 2 はノード 1 に `QueryHit` を送信し，検索を開始したノード 1 にデータが発見されたことを知らせる．