分散処理高速化のための
hadoop の実装と検証

島根大学　総合理工学部　数理･情報システム学科

応用情報学講座　田中研究室

S073038　小室 翼

第 1 章　　序論

1.1.　研究背景

現在、計算機のパフォーマンスを向上させるに当たって用いられる手段は 2 つあり、既存の計算機の CPU やメモリ等の性能強化等によって能力をあげたり、より性能の高い計算機に変えるスケールアップという手段と、複数台の計算機に処理を分散させることにより性能を高くするスケールアウトという手段がある。しかし、スケールアップでは計算機の部品の性能にも限界があるため、ある一定以上のパフォーマンスが出ることがなく、またコストも非常に高くなる。一方スケールアウトでは、複製や同期が容易な際にはスケールアップによって向上した計算機の性能と同等程度の性能向上をする際のコストも低く、分散処理を行う台数を増やす度性能が向上する。そのため、複雑な複製や同期などの処理が必要とされない処理においての性能向上をする際には、スケールアップからスケールアウトへと移行していっている。

1.2.　研究目的

スケールアウトを実現するためのアプリケーションのひとつに hadoop[1]が存在し、また hadoop は OSS であるため、誰でも容易に分散処理環境を実現することができる。

そこで、本研究では hadoop における分散処理の高速化を行うための準備として、hadoop のベンチマークテストを行い、その分散処理性能を計測する。

第 2 章　　hadoop の基盤技術[1]

2.1. HDFS

HDFS とは「Hadoop Destributed File System」（hadoop 分散ファイルシステム）の略称であり、分散ファイルシステムとはネットワークで相互に接続された複数台のマシンにまたがってひとつのファイルシステムを構築するものを指す。HDFS はデータの保存と管理を分散環境で扱う仕組みであり、その扱うファイルが大容量であることを想定した最適化をされている。また、分散環境での利用によって扱う計算機の数が増え、ハードウェアの障害に合う確率も増えることを想定して、耐障害性に優れた機能を保有している。

HDFS は「NameNode」と「DataNode」の 2 つのプロセスにより構成されており、「NameNode」はマスターとなる単一のプロセスとして動作し、ファイルやディレクトリの管理情報の保持を行い、データアクセスの制御を行う。「DataNode」は分散処理を行う各計算機上で動作し、NameNode の制御によってブロック単位での書き込みや読み込み、ソート、複製などのファイル操作を行うプロセスである。

2.2. HBase

HBase は key value ストアの一つであり、key value ストアとは従来使用されてきたリレーショナルデータベースと違い、(key,value)のペアを書き込み、key を指定することで value を読み出すという簡単なデータベースである。それ故に、リレーショナルデータベースでは実現することが非常に難しい、複数台にまたがったデータの分散が出来るため、高い性能やスケーラビリティ、耐故障性を確保できる。

HDFS は大容量ファイルを扱うことに特化しており、小さなファイルの大量の読み書きには向いていないのに対し、HBase は Web ページなどの情報を扱う際に構造化された小さなサイズのデータに大してデータベースのようにアクセスをすることに向いている。また、HBase のデータストレージには HDFS を使用している。

2.3. Map/Reduce

Map/Reduce とは hadoop における分散処理の中核となる部分であり、Map 処理・Shuffle 処理・Reduce 処理の三つから成り立っている。分散処理を実行するに当たってプログラマは Map 処理と Reduce 処理に関するプログラムだけ記述すればよく、分散処理を効率よく実行させている Shuffle 処理は hadoop が自動的に行なっている。

Map/Reduce はジョブの実行制御を行う「JobTracker」と「TaskTracker」の 2 つのプロセスによって構成されている。ジョブとは Map/Reduce 処理の実行を依頼するプログラムにとっての実行単位のことを指す。「JobTracker」はジョブの実行制御とジョブの実行状況を管理するためのマスターとなる単一のプロセスで、TaskTracker へのタスクの割り振りも行っている。タスクはジョブの実行過程で行われる Map 処理や Reduce 処理の実行単位を指す。「TaskTracker」は JobTracker に対して、実行可能なタスクが存在するか一定時間ごとに確認し、処理できるタスクが存在すればそのタスクとタスク実行に必要となるリソースを確保し、タスクを実行する。

2.3.1. Hadoop における分散処理の流れ

2.3.1.1. Map 処理

複数の計算機上の処理対象データから、従来扱うファイルサイズより小さなキーと値という必要最小限なデータのみを取り出す処理。(例:図 2.1)

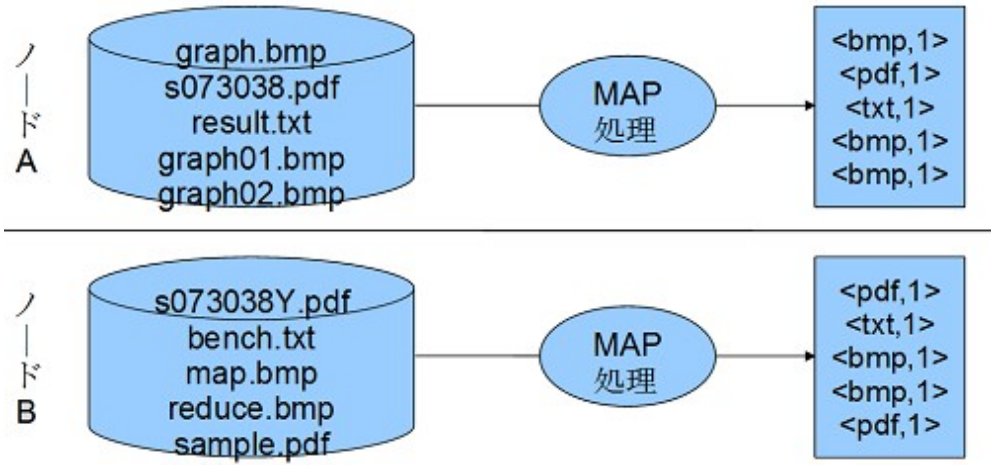

図 2.1 拡張子のカウントにおける Map 処理の例

2.3.1.2. Shuffle 処理

まず、Map 処理によって出力されたデータをパーティショニングする。パーティショニングは hadoop デフォルトの設定だと、Map 処理によって出力された中間データの結果のキー値から算出したハッシュ値によって行われる。パーティショニングされた後、各パーティションごとにキー値によってソートが実行され、集約関数が定義されていた場合は集約処理が行われ、最後にローカルディスクにデータが出力される。(例:図 2.2)

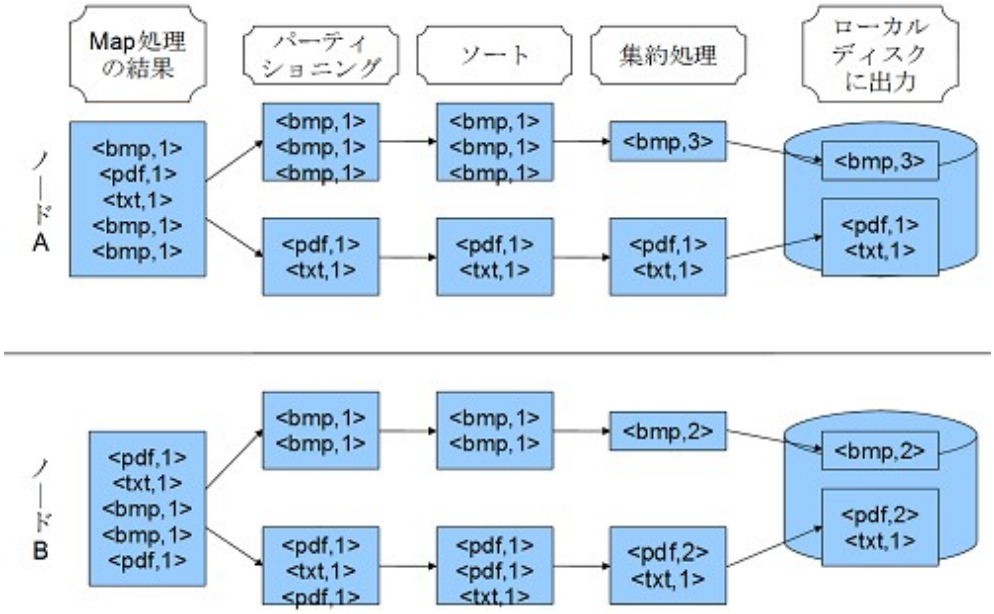

図 2.2　拡張子のカウントにおける Shuffle 処理の例

次に、ローカルディスクに出力されたパーティションのうち、同一のキー値を持つパーティションをローカルディスクにコピーし、集める。そして、パーティションをマージして先程と同様にキー値によってソートを行う。(例:図 2.3)

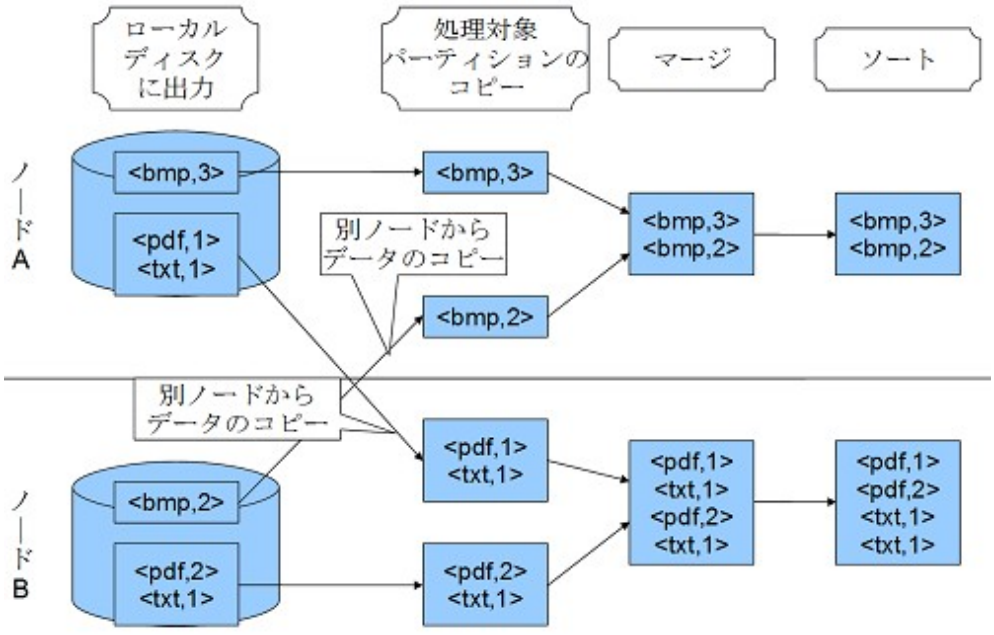

図 2.3　拡張子のカウントにおける Shuffle 処理の例

2.3.1.3. Reduce 処理

Shuffle 処理によって纏められたキーと値のペアから、ユーザが必要とするデータを集約をしてデータの件数のカウントや各データに含まれる値の合計値を求めたり、データの加工などを行って最終的に必要とするデータを取得する。
(例:図 2.4)

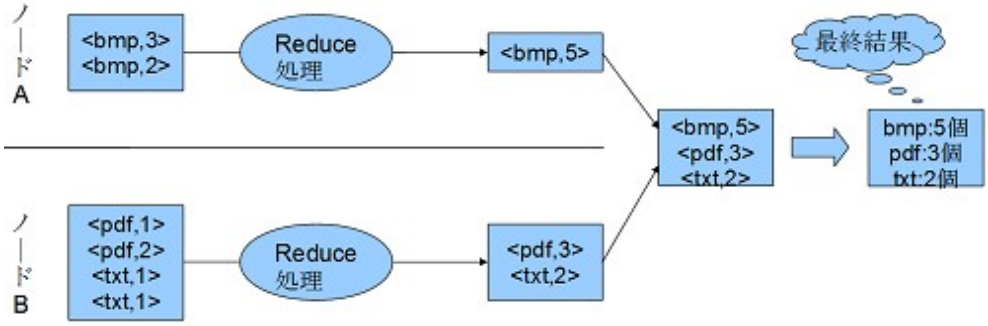

図 2.4 拡張子のカウントにおける Reduce 処理の例

第 3 章　　ベンチマークテスト

3.1.　評価環境

評価環境として 6 台の計算機を用意した。その内 5 台の計算機上で DataNode と TaskTracker を動かした。残りの 1 台をマスターノードとして、NameNode と JobTracker を動かした。

3.2.　計算機の構成

それぞれのノードのスペックを表 3.1 に示す。

	node01	node02	node03	node04	node05	MasterNode
OS	Windows 7 Professional 64bit	Windows Server2008 R2 64bit	Windows Server2008 R2 64bit	Windows Server2008 R2 64bit	Windows XP 32bit	Windows Server2008 R2 64bit
CPU	Pentium4 3.0GHz	Celeron 1.80GHz	Celeron 1.80GHz	Celeron 1.80GHz	Pentium4 3.20GHz	Core2 Quad 3.0GHz
メモリ	2.0GB	2.0GB	512MB	512MB	512MB	8.0GB
仮想メモリ	3057MB	3057MB	763MB	763MB	1024MB	12091MB
HDD	160GB	80GB	160GB	80GB	40GB	210GB

表 3.1　各ノードのスペック

3.3.　HDFS ベンチマーク

HDFS の性能を測定するため、全ての計算機の組み合わせ（表 3.2）ごとに Map/Reduce を使用して書き込みと読み込みを行った。書き込み、読み込みベンチマークはともに hadoop に付属している TestDFSIO(hadoop-0.20.2-test.jar に含まれている)を使用した。なお、ベンチマークの実行はすべてレプリケーション数を 1 にして行っている。

1 台	node01	node02	node03	node04	node05					
2 台	node01 node02	node01 node03	node01 node04	node01 node05	node02 node03	node02 node04	node02 node05	node03 node04	node03 node05	node04 node05
3 台	node01 node02 node03	node01 node02 node04	node01 node02 node05	node01 node03 node04	node01 node03 node05	node01 node04 node05	node02 node03 node04	node02 node03 node05	node02 node04 node05	node03 node04 node05
4 台	node01 node02 node03 node04	node01 node02 node03 node05	node01 node02 node04 node05	node01 node03 node04 node05	node02 node03 node04 node05					
5 台	node01 node02 node03 node04 node05									

表 3.2　各台数における計算機の組み合わせ

3.3.1.　書き込みベンチマーク

書き込みベンチマークでは図 3.1 のコマンドを用いて 1MB のファイルを 128 個生成を 3 回行い、その平均値を求めた。

```
$ /usr/local/hadoop/bin/hadoop jar $(cygpath -w /usr/local/hadoop/hadoop-0.20.2-test.jar) TestDFSIO -write -nrFiles 128 -fileSize 1
```

図 3.1　1MB×128 個データ生成コマンド

まず、全ての計算機の性能が異なっているため単体での書き込みベンチマークを行い、その性能を図 3.2.1 に示す。

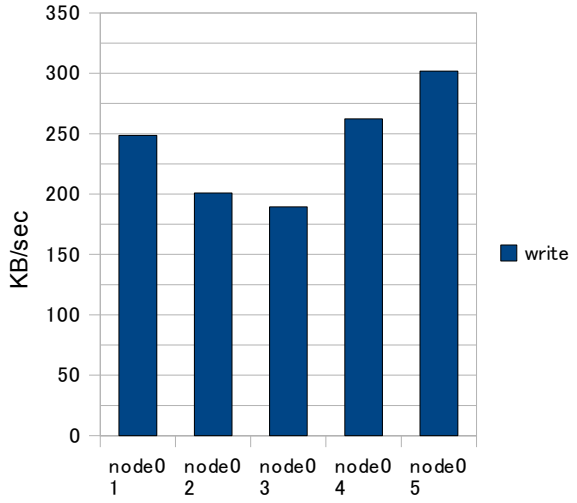

図 3.2.1　各計算機の write 単体性能

測定結果から、write の単体性能は node05>node04>node01>node02>node03 の順であった。

次に、計算機の台数ごとの全ての組み合わせにおける 1 秒あたりのデータ書き込み量と、計測結果からの予測データ書き込み量の関係を図 3.2.2,図 3.2.3,図 3.2.4,図 3.2.5 に示す。予測データは、計算機 1 台を新規追加して計測を行う際に、その組み合わせから 1 番性能の低い計算機を除いた組み合わせの実測値と、1 番性能の低い計算機 1 台の時の実測値を足し合わせによって算出している。

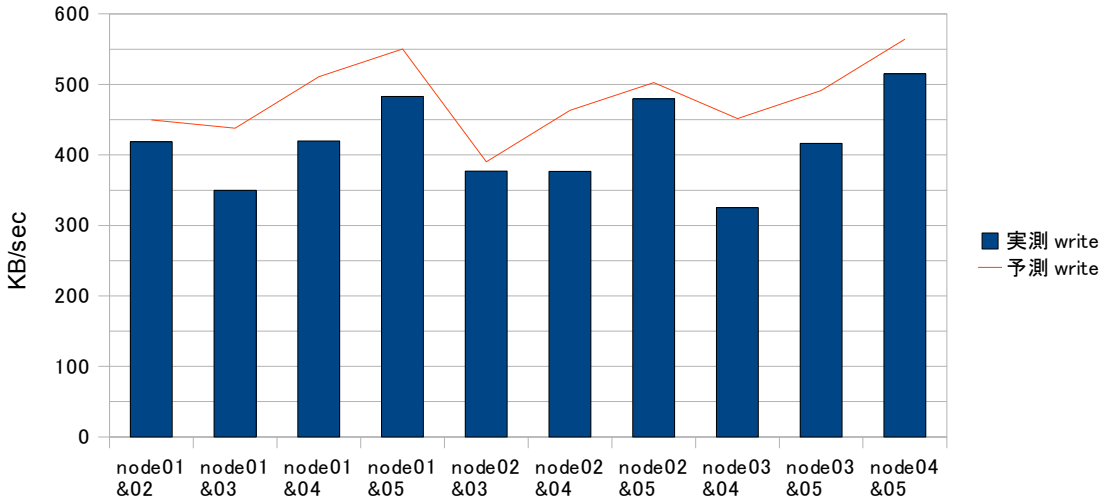

図 3.2.2　計算機 2 台での全ての組み合わせの write の実測値及び予測値

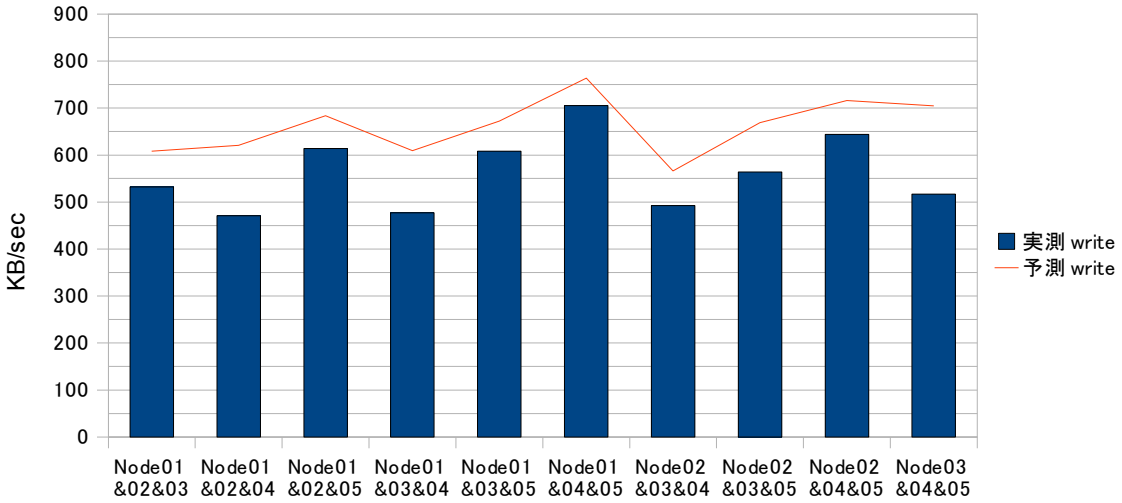

図 3.2.3　計算機 3 台での全ての組み合わせの write の実測値及び予測値

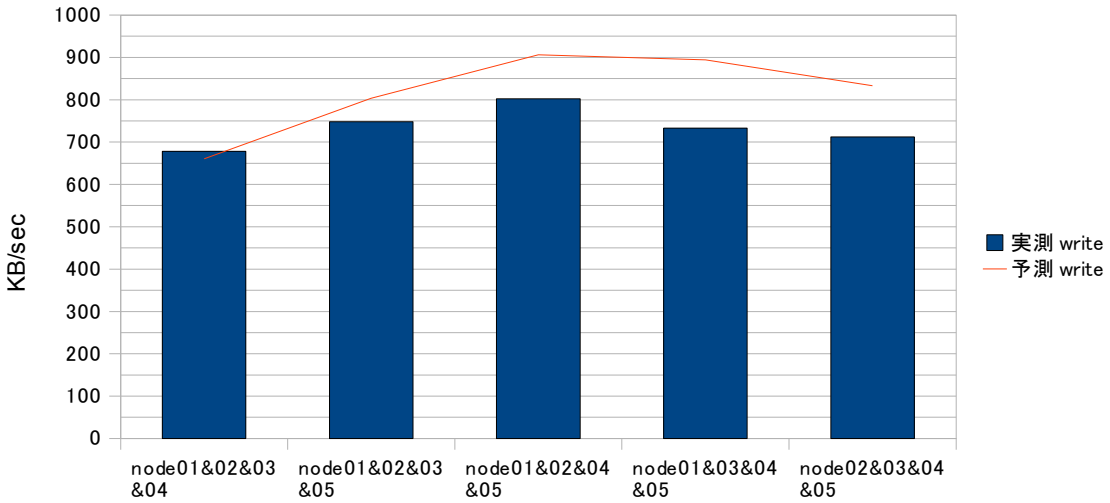

図 3.2.4　計算機 4 台での全ての組み合わせの write の実測値及び予測値

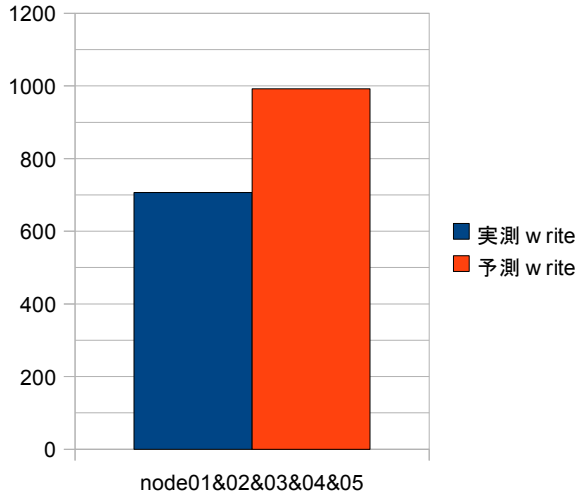

図 3.2.5　計算機 5 台の組み合わせでの write の
実測値及び予測値

3.3.2.　読み込みベンチマーク

次に書き込んだ 128MB のデータを図 3.3 のコマンドによって読み込みを 3 回行い、その平均値を求めた。

```
$ /usr/local/hadoop/bin/hadoop jar $(cygpath -w /usr/local/hadoop/hadoop-0.20.2-test.jar) TestDFSIO -read -nrFiles 128 -fileSize 1
```

図 3.3　1MB×128 個データ読み込みコマンド

書き込みベンチマークと同様に、全ての計算機の性能が異なっているため単体での読み込みベンチマークを行い、その性能を図 3.4.1 に示す。

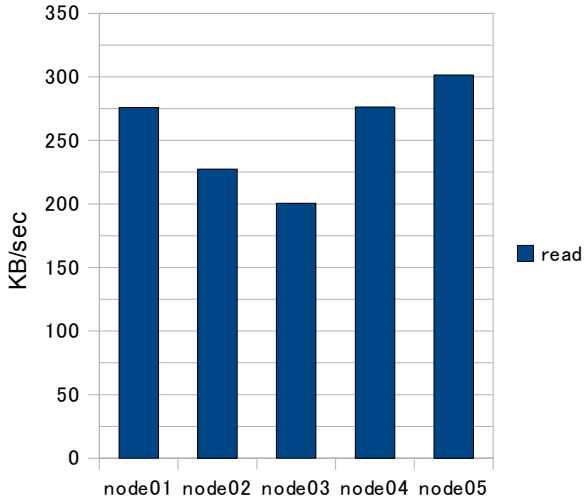

図 3.4.1　各計算機の read 単体性能

測定結果から、read の単体性能は node05>node04>node01>node02>node03 の順であった。

次に、計算機の台数ごとの全ての組み合わせにおける 1 秒あたりのデータ読み込み量と、計測結果からの予測データ読み込み量の関係を図 3.4.2,図 3.4.3,図 3.4.4,図 3.4.5 に示す。予測データは、計算機 1 台を新規追加して計測を行う際に、その組み合わせから 1 番性能の低い計算機を除いた組み合わせの実測値と、1 番性能の低い計算機 1 台の時の実測値を足し合わせによって算出している。

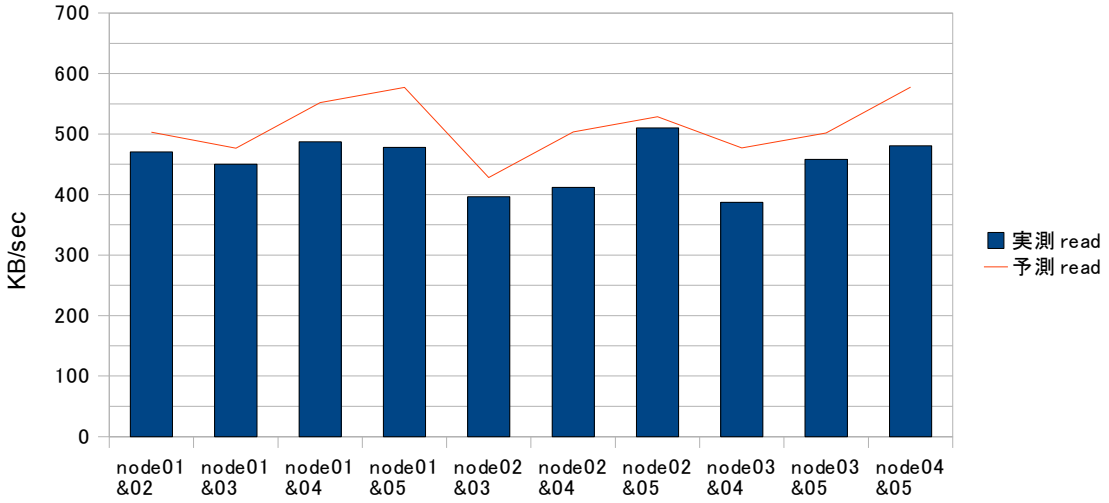

図 3.4.2　計算機 2 台での全ての組み合わせの read の実測値及び予測値

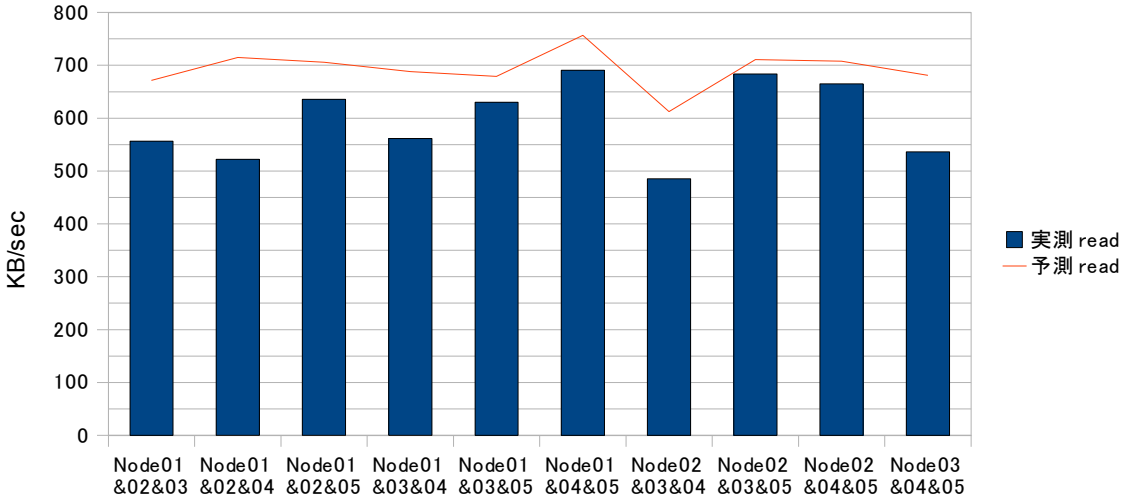

図 3.4.3　計算機 3 台での全ての組み合わせの read の実測値及び予測値

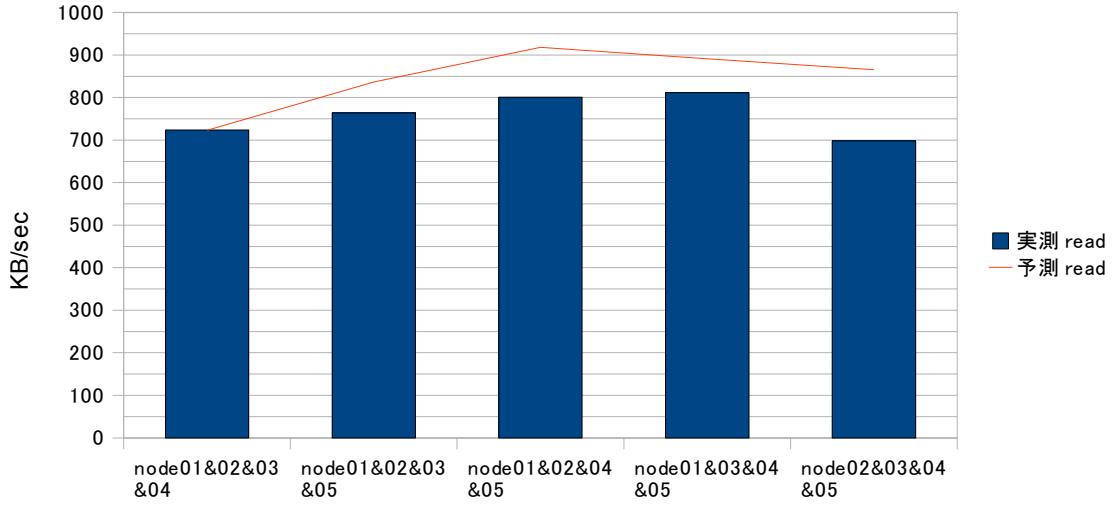

図 3.4.4　計算機 4 台での全ての組み合わせの read の実測値及び予測値

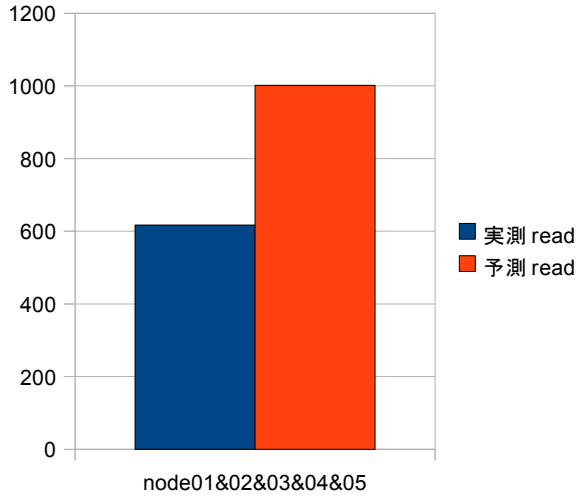

図 3.4.5　計算機 5 台の組み合わせでの read の
実測値及び予測値

3.4.　ソートベンチマーク

Map/Reduce の基本的な性能を測定するため、全ての計算機の組み合わせごとにサンプルプログラムに添付されているソートプログラムを用いて性能評価を行う。今回の実験では 128MB のデータを生成し、それをソートした。こちらも HDFS ベンチマークと同様にレプリケーション数を 1 として実験を行った。

3.4.1.　ランダムサイズデータの生成

データの生成には hadoop に添付されている randomwrite プログラムを使用した。128MB のデータを生成するために図 3.5 に示される設定ファイルを用意した。この設定ファイルは key,value それぞれの長さが 10~1000 バイト分のレコードが合計 128MB 生成される。それを 1MB ごとに分割し、それぞれを 1 つの Map タスクに割り当てる。key,value の長さの合計値の平均は約 1KB なので、全体のレコード数は平均 100M エントリと考えられる。
図 3.6 のコマンドでこの設定ファイルを利用して、データの生成をを 3 回行い、その平均値を求めた。

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
 <property>
 <name>test.randomwrite.min_key</name>
 <value>10</value>
 </property>
 <property>
 <name>test.randomwrite.max_key</name>
 <value>1000</value>
 </property>

 <property>
 <name>test.randomwrite.min_value</name>
 <value>10</value>
 </property>
 <property>
 <name>test.randomwrite.max_value</name>
 <value>1000</value>
 </property>

 <property>
 <name>test.randomwrite.bytes_per_map</name>
 <value>1000000</value>
 </property>
 <property>
 <name>test.randomwrite.total_bytes</name>
 <value>128000000</value>
 </property>
</configuration>
```

図 3.5　128MB 生成のための設定ファイル(randomwriter.conf)

```
$ /usr/local/hadoop/bin/hadoop jar $(cygpath -w /usr/local/hadoop/hadoop-0.20.2-examples.jar) randomwriter -conf randomwriter.conf random
```

図 3.6　128MB 生成のためのコマンド

まず、全ての計算機の性能が異なっているため単体でのランダム書き込みベンチマークを行い、その性能を図 3.7.1 に示す。

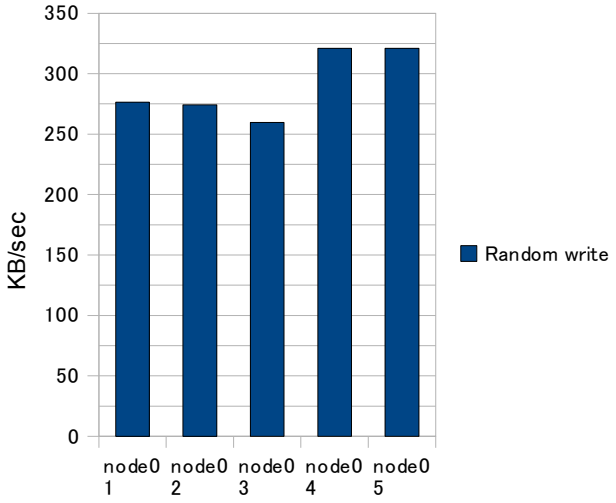

図 3.7.1　各計算機の random write 単体性能

測定結果から、random write の単体性能は node05>node04>node01>node02>node03 の順であった。

次に、計算機の台数ごとの全ての組み合わせにおける 1 秒あたりのランダムデータ書き込み量と、計測結果からの予測ランダムデータ書き込み量の関係を図 3.7.2,図 3.7.3,図 3.7.4,図 3.7.5 に示す。予測データは、計算機 1 台を新規追加して計測を行う際に、その組み合わせから 1 番性能の低い計算機を除いた組み合わせの実測値と、1 番性能の低い計算機 1 台の時の実測値を足し合わせによって算出している。

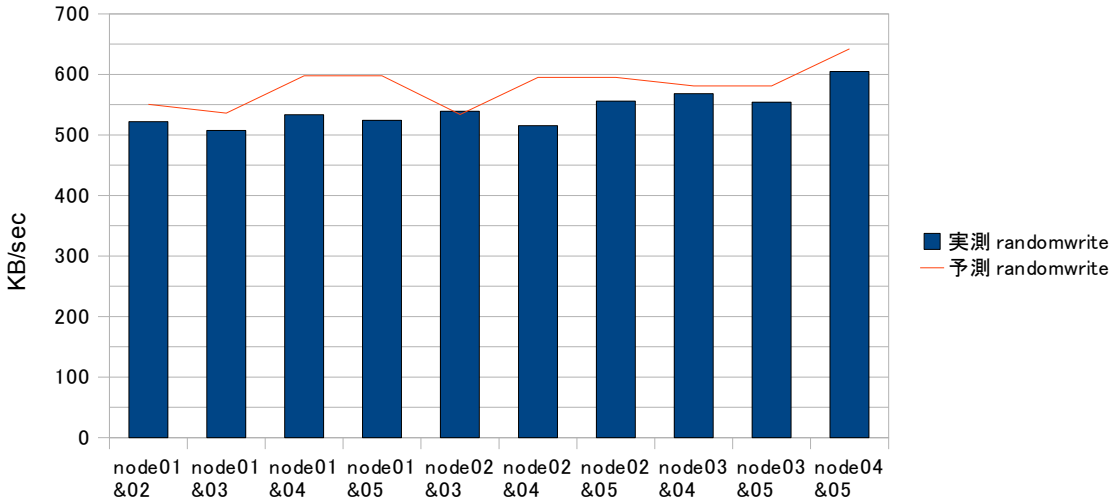

図 3.7.2　計算機 2 台での全ての組み合わせの random write の実測値及び予測値

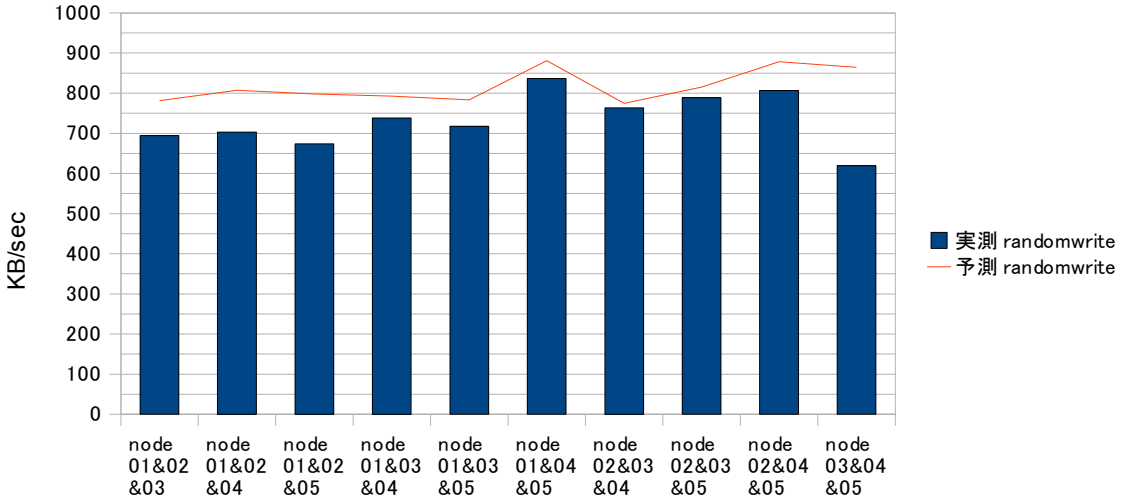

図 3.7.3　計算機 3 台での全ての組み合わせの random write の実測値及び予測値

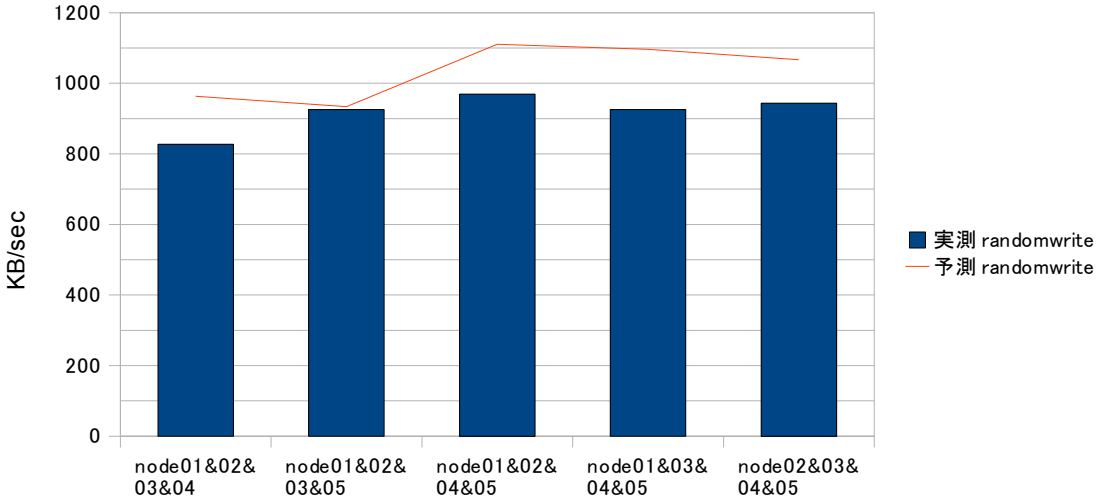

図 3.7.4　計算機 4 台での全ての組み合わせの random write の実測値及び予測値

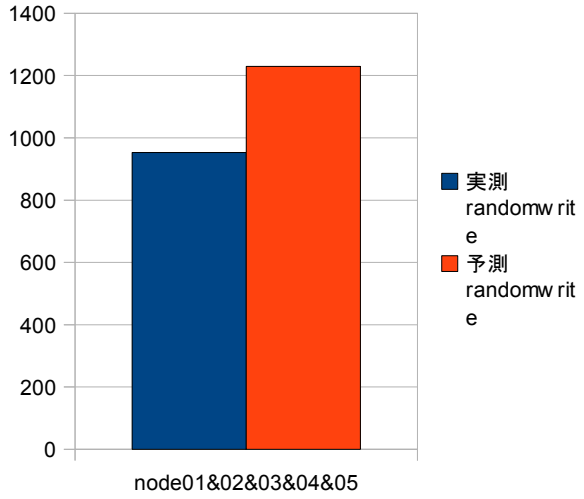

図 3.7.5　計算機 5 台の組み合わせでの random write の実測値及び予測値

3.4.2.　データのソート

次に、上記の設定で生成したファイルを図 3.8 のコマンドでソートを 3 回行い、その平均値を求めた。

```
$ /usr/local/hadoop/bin/hadoop jar $(cygpath -w /usr/local/hadoop/hadoop-0.20.2-examples.jar) sort random random-sort
```

図 3.8　ソートのためのコマンド

まず、全ての計算機の性能が異なっているため単体でのソートベンチマークを行い、その性能を図 3.9.1 に示す。

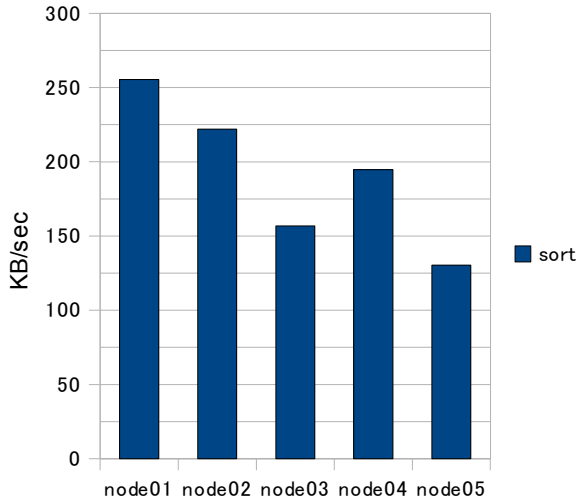

図 3.9.1　各計算機の sort 単体性能

測定結果から、sort の単体性能は node01>node02>node04>node03>node05 の順であった。

次に、計算機の台数ごとの全ての組み合わせにおける 1 秒あたりのデータソート速度と、計測結果からの予測データソート速度の関係を図 3.9.2,図 3.9.3,図 3.9.4,図 3.9.5 に示す。予測データは、計算機 1 台を新規追加して計測を行う際に、その組み合わせから 1 番性能の低い計算機を除いた組み合わせの実測値と、1 番性能の低い計算機 1 台の時の実測値を足し合わせによって算出している。

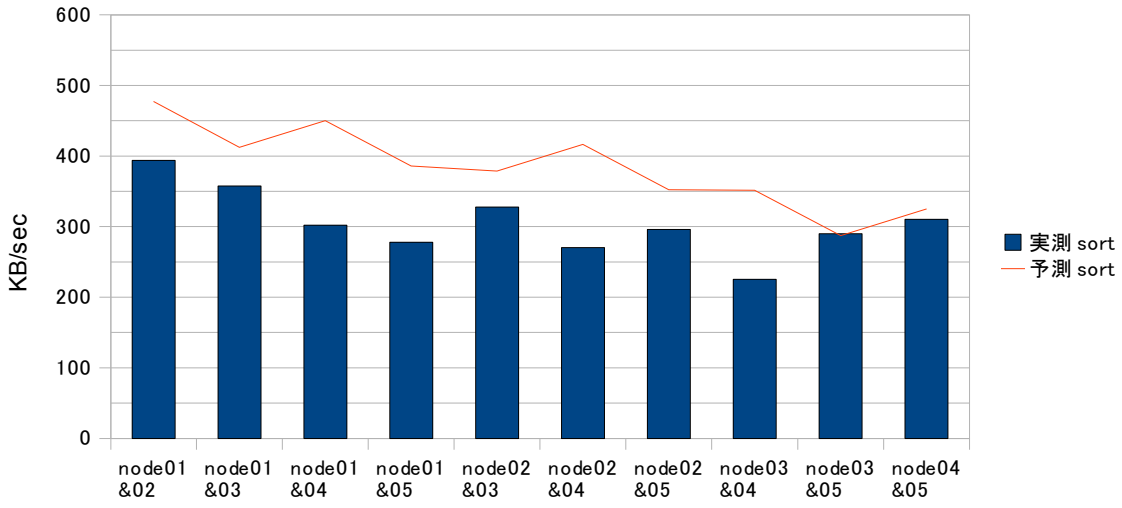

図 3.9.2　計算機 2 台での全ての組み合わせの sort の実測値及び予測値

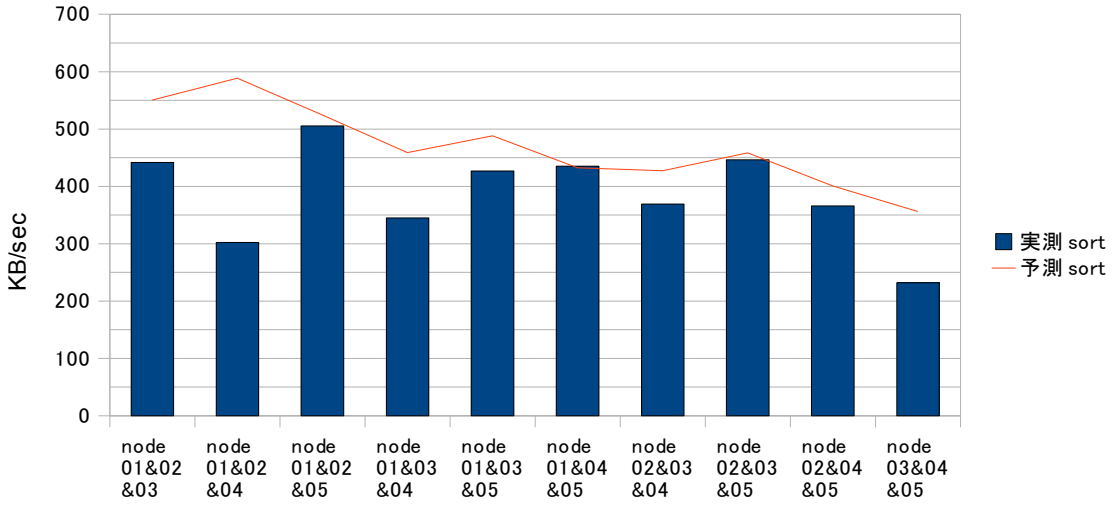

図 3.9.3　計算機 3 台での全ての組み合わせの sort の実測値及び予測値

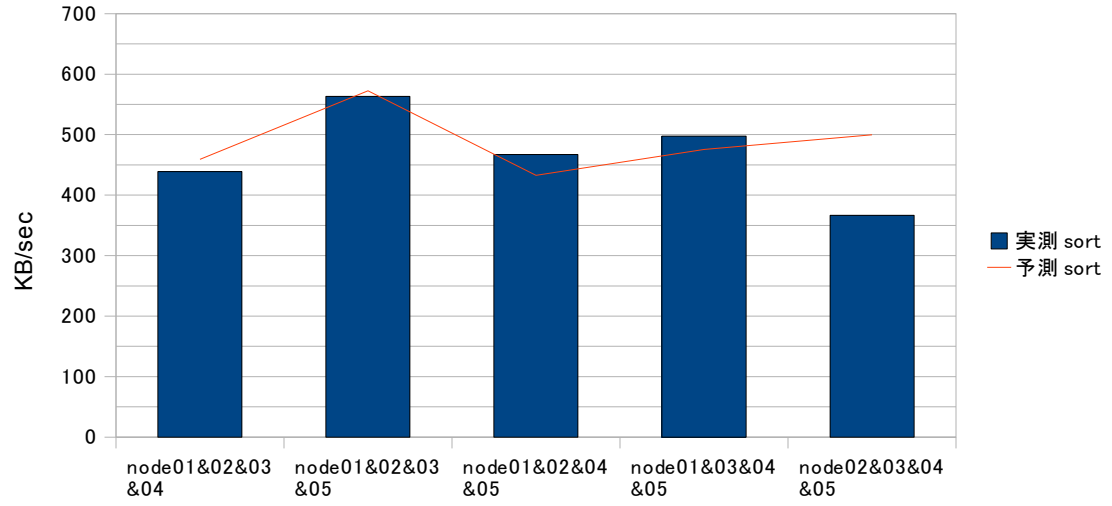

図 3.9.4　計算機 4 台での全ての組み合わせの sort の実測値及び予測値

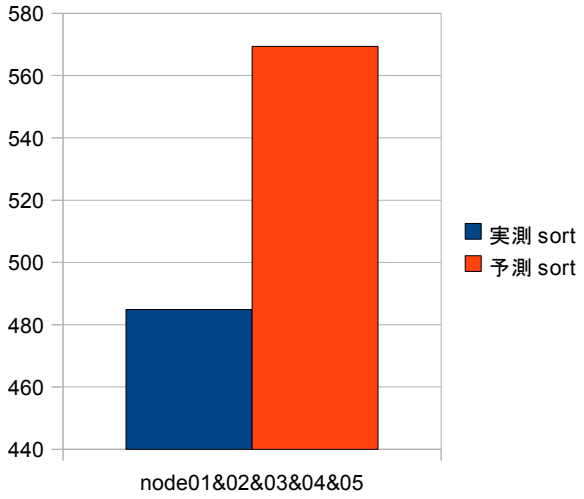

図 3.9.5　計算機 5 台の組み合わせでの sort の
実測値及び予測値

3.5.　考察

write、read、random write、sort のそれぞれのグラフを見ると、ほとんどの組み合わせにおいて予測値よりも実測値のほうが下回っている。稀に実測値のほうが予測値を上回っている場合もあるが、これは予測値を算出するに当たり使用したベンチマーク結果のデータが、ベンチマークテストの試行回数が少ないために実際に計測されるべき値より少なくなった可能性が挙げられる。

次に、hadoop のスケールアウト性能について考察を行う。write、read、random write、sort それぞれについて、計算機 2~5 台での全ての組み合わせにおけるベンチマークの測定値と、その測定を行うにあたって新しく 1 台追加した計算機を除いたときの測定値を比較するため、それぞれの結果をグラフ化する。

write ベンチマーク（図 3.10.1、図 3.10.2、図 3.10.3、図 3.10.4）

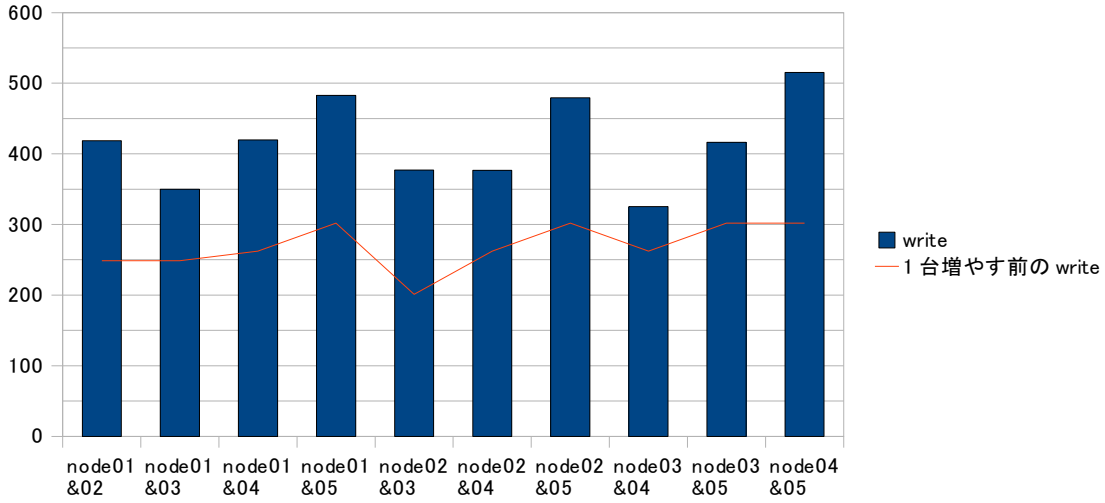

図 3.10.1　2 台及び 1 台の際の write 性能

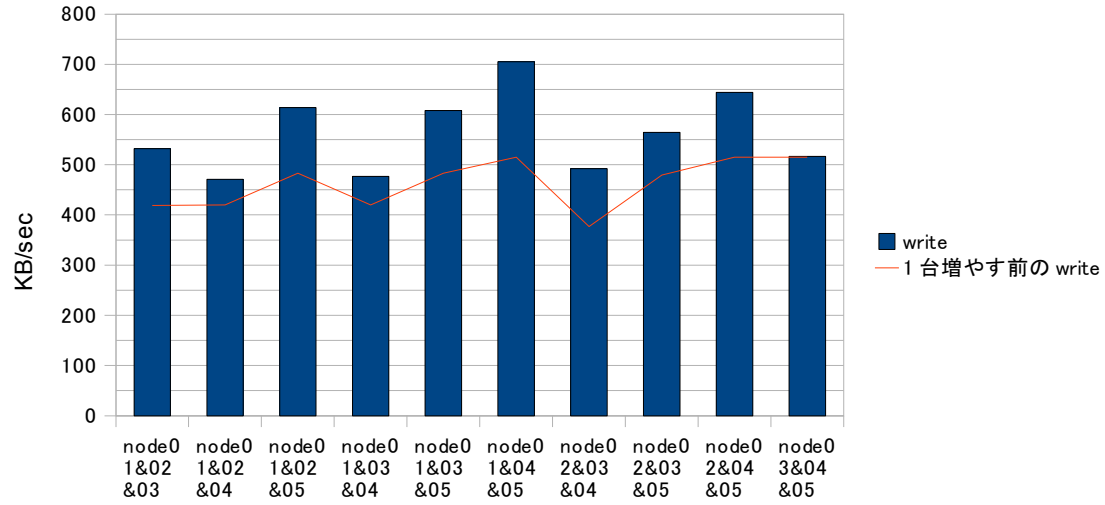

図 3.10.2　3 台及び 2 台の際の write 性能

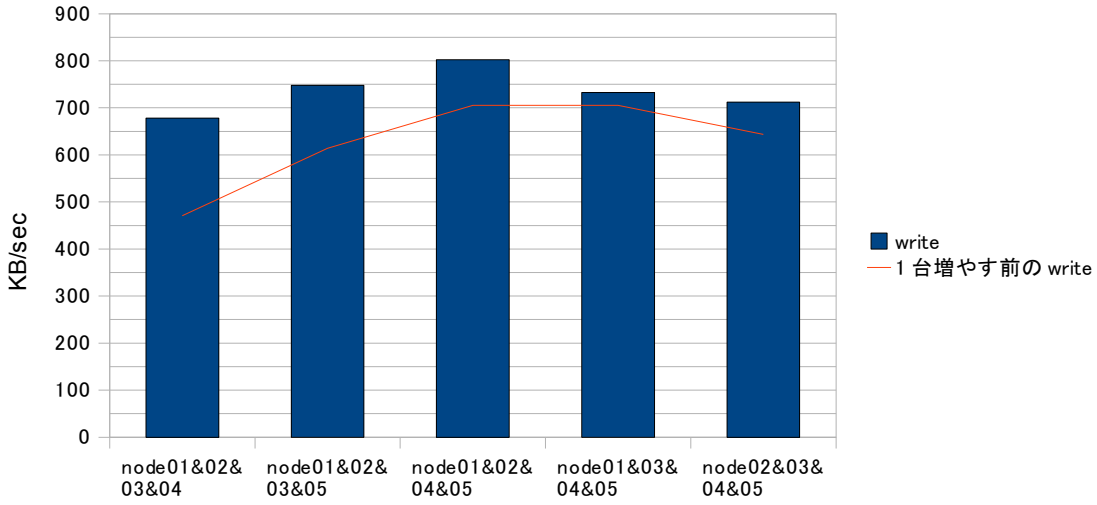

図 3.10.3　4 台及び 3 台の際の write 性能

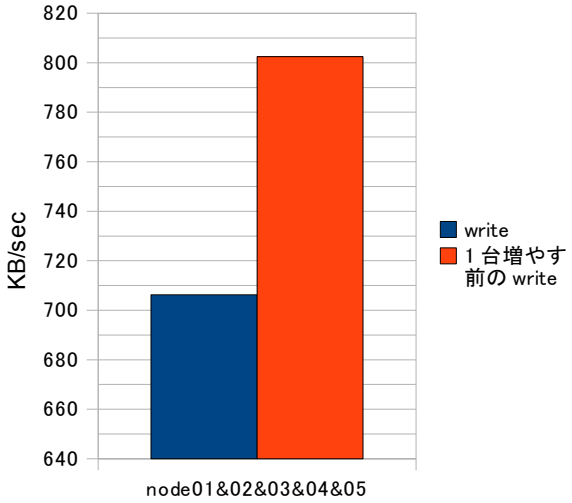

図 3.10.3　5 台及び 4 台の際の write 性能

write 性能について、グラフを見ると概ね少なからず新しい計算機を追加するにつれ、性能は上がっている。しかし、5 台測定時の性能を見ると著しく性能が下がっている。これは、計算機の台数が増えて Map/Reduce 処理の際に輻輳が起こった可能性が考えられるが、後述する random write 性能の際も 5 台測定時の性能が下がっており、こちらは Map 処理のみで Shuffle 処理・Reduce 処理が行われないため、Reduce 処理において時間がかかっている可能性が高い。また、ベンチマークテストの試行回数が少ないため誤差が発生した可能性も少なからずあると考えられる。

read ベンチマーク（図 3.11.1、図 3.11.2、図 3.11.3、図 3.11.4）

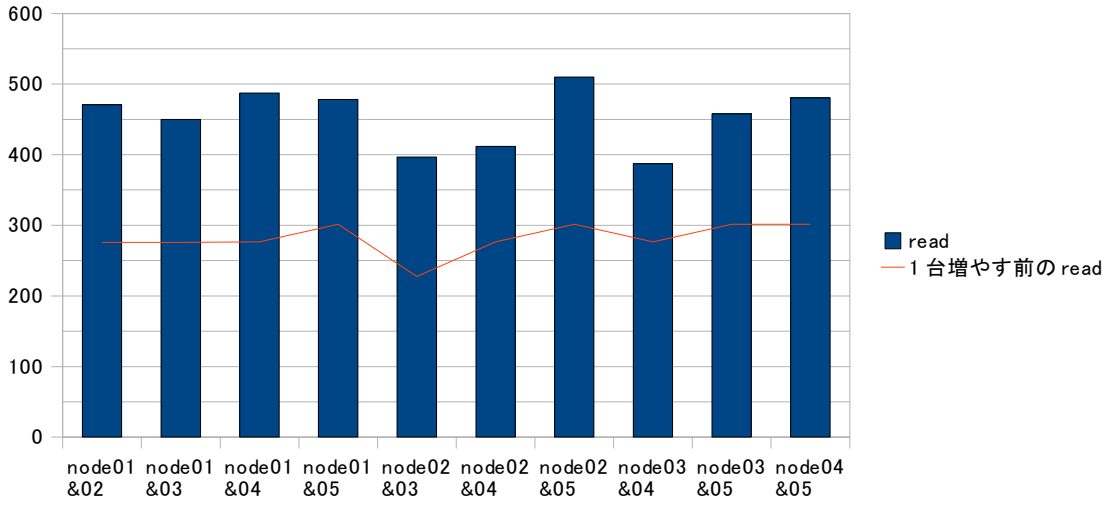

図 3.11.1　2 台及び 1 台の際の read 性能

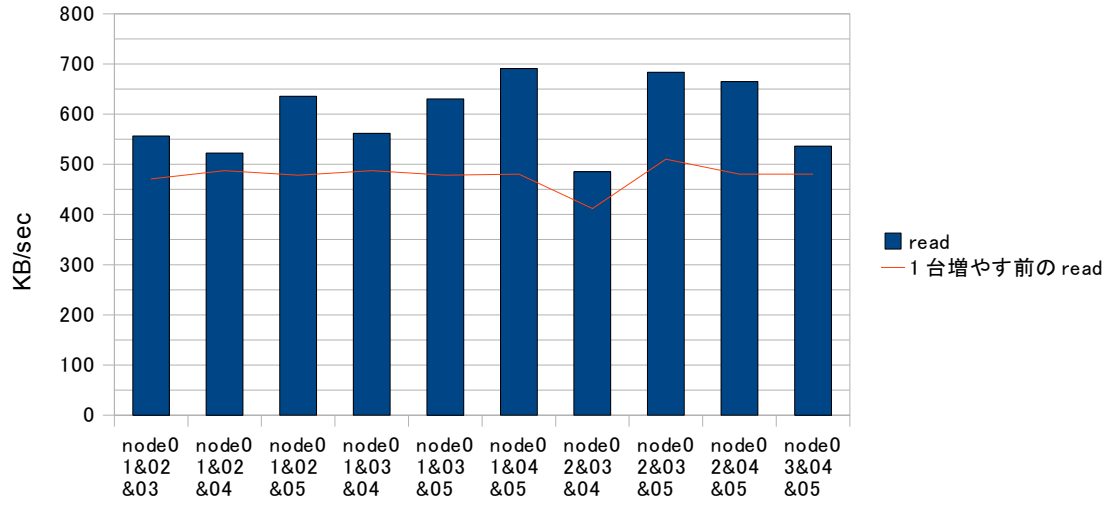

図 3.11.2　3 台及び 2 台の際の read 性能

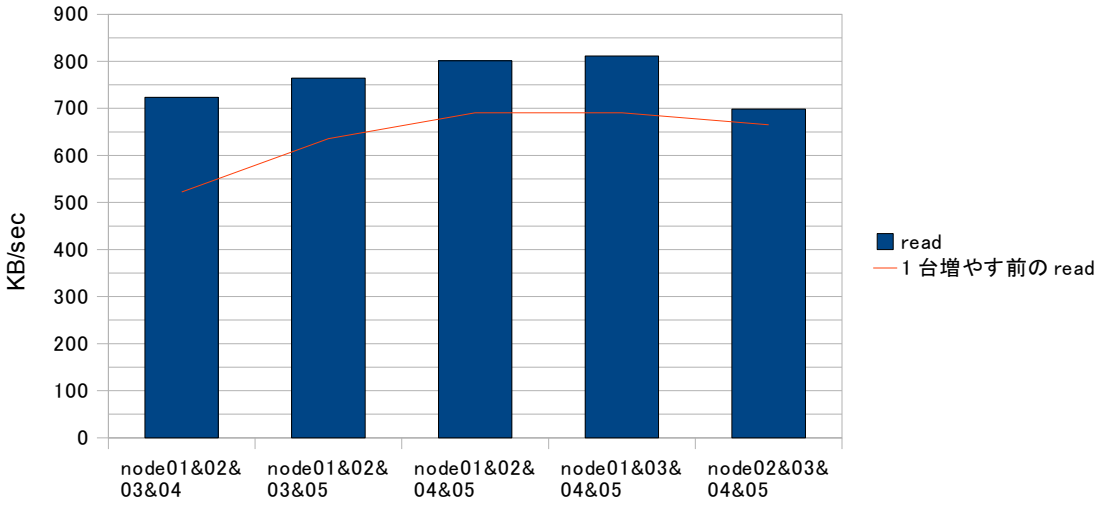

図 3.11.3　4 台及び 3 台の際の read 性能

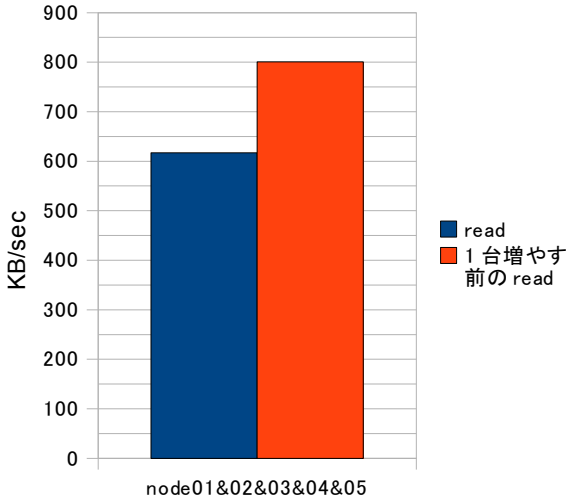

図 3.11.4　5 台及び 4 台の際の read 性能

read 性能について、グラフを見ると write ベンチマークの結果と同様に新しい計算機を追加するにつれ、性能は上がっている。しかし、5 台測定時の性能も write のときのように著しく性能が下がっている。こちらも、計算機の台数が増えて Map/Reduce 処理の際に輻輳が起こった可能性が考えられるが、後述する random write 性能の際も 5 台測定時の性能が下がっており、こちらは Map 処理のみで Shuffle 処理・Reduce 処理が行われないため、Reduce 処理において時間がかかっている可能性が高い。また、ベンチマークテストの試行回数が少ないため誤差が発生した可能性も少なからずあると考えられる。

random write ベンチマーク（図 3.12.1、図 3.12.2、図 3.12.3、図 3.12.4）

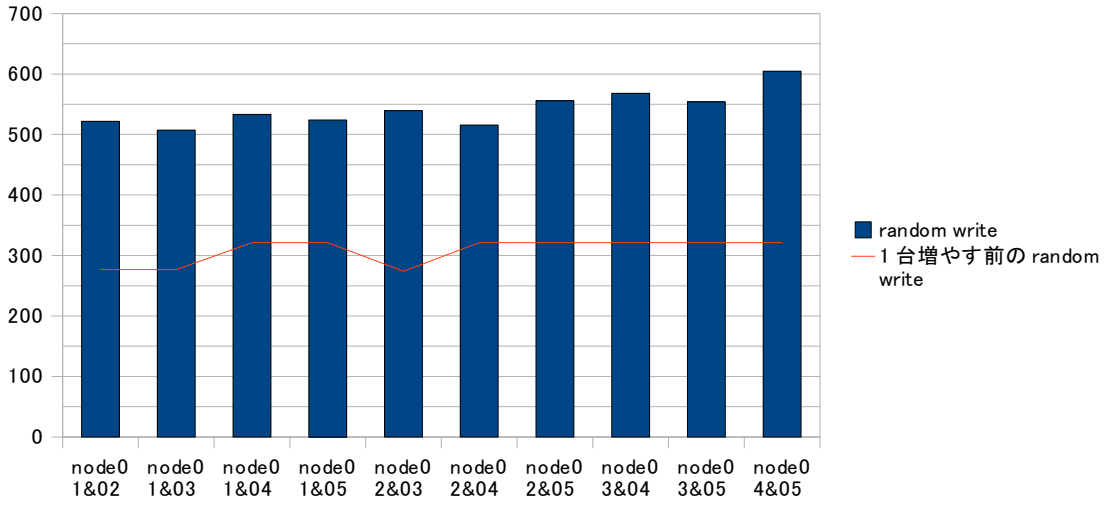

図 3.12.1　2 台及び 1 台の際の random write 性能

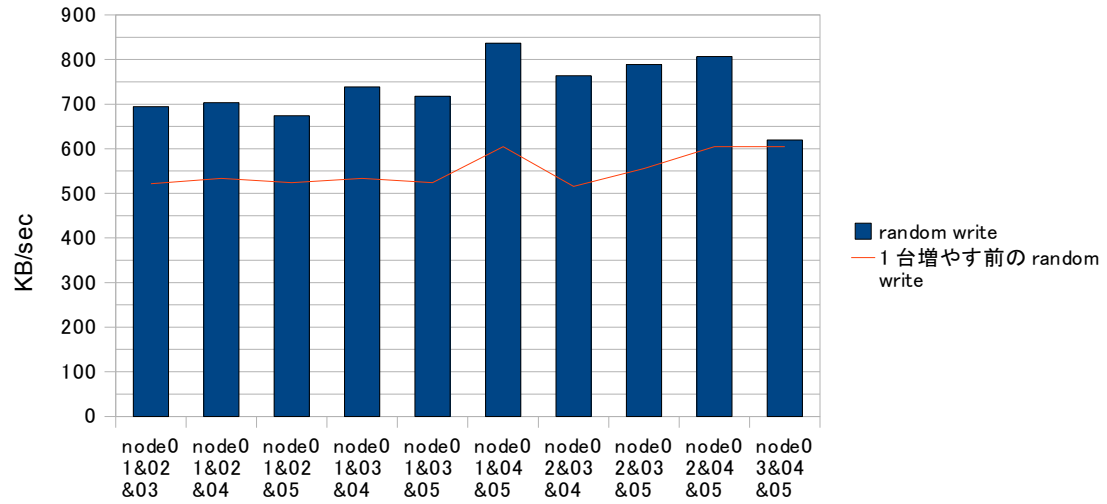

図 3.12.2　3 台及び 2 台の際の random write 性能

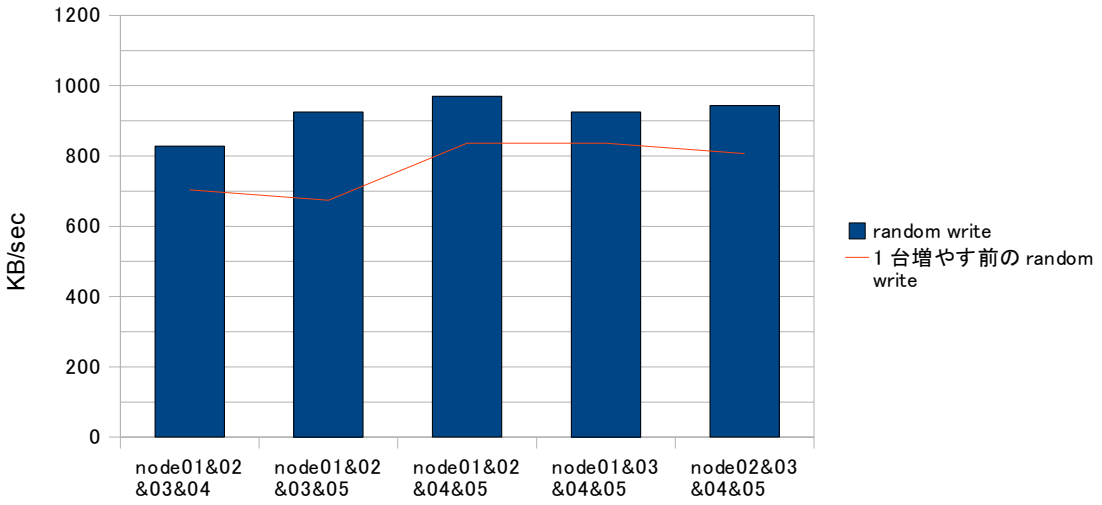

図 3.12.3　4 台及び 3 台の際の random write 性能

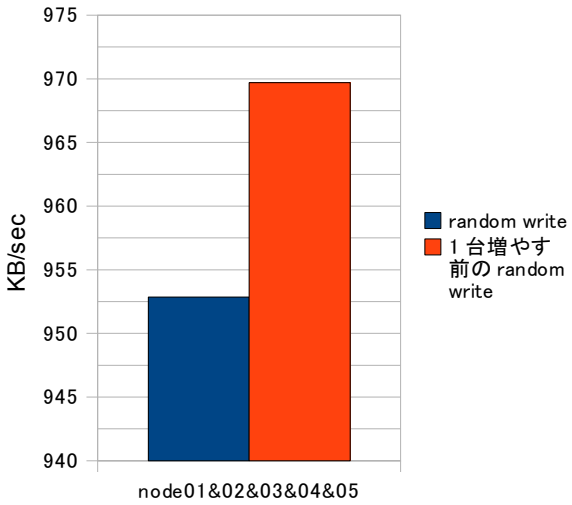

図 3.12.4　5 台及び 4 台の際の random write 性能

random write 性能について、グラフを見と write,read ベンチマークの結果と同様に新しい計算機を追加するにつれ、性能は上がっている。しかし、5 台測定時の性能も write,read の例にもれず著しく性能が下がっている。random write の実行には Map 処理のみが行われるため、Reduce 処理や Shuffle 処理による輻輳が原因であると考えづらい。よって、Map 処理において時間がかかっているか、ベンチマークテストの試行回数が少ないため誤差が発生した可能性が考えられる。

sort ベンチマーク（図 3.13.1、図 3.13.2、図 3.13.3、図 3.13.4）

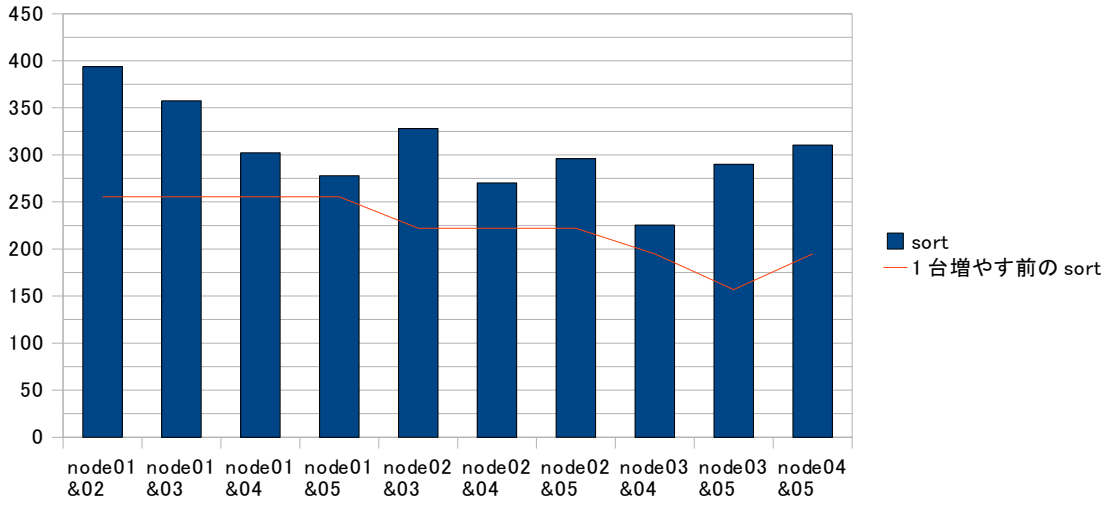

図 3.13.1　2 台及び 1 台の際の sort 性能

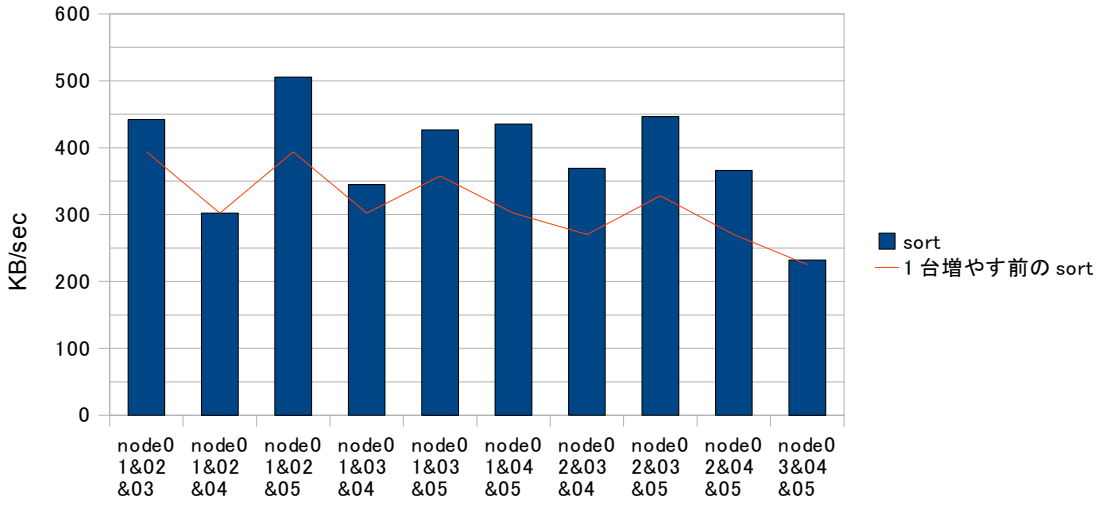

図 3.13.2　3 台及び 2 台の際の sort 性能

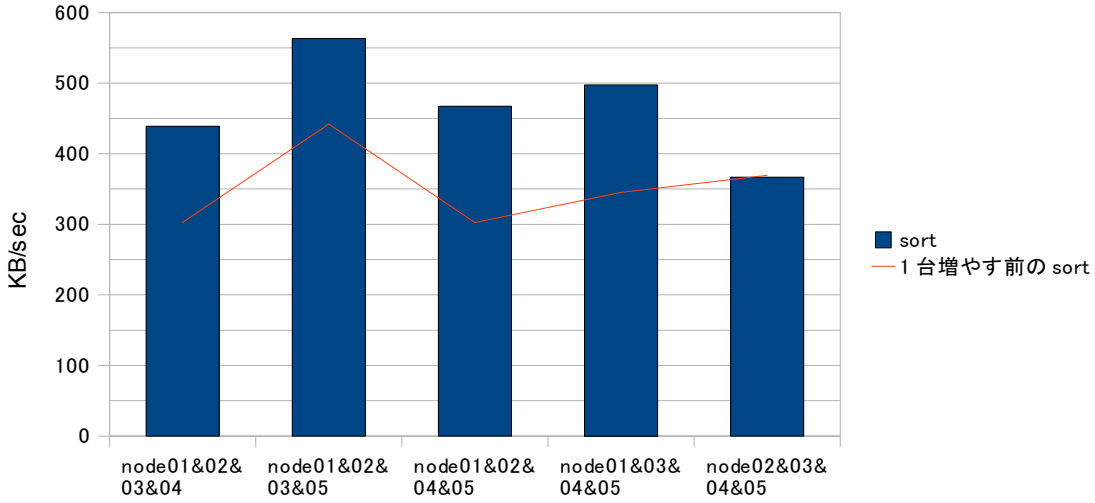

図 3.13.3　4 台及び 3 台の際の sort 性能

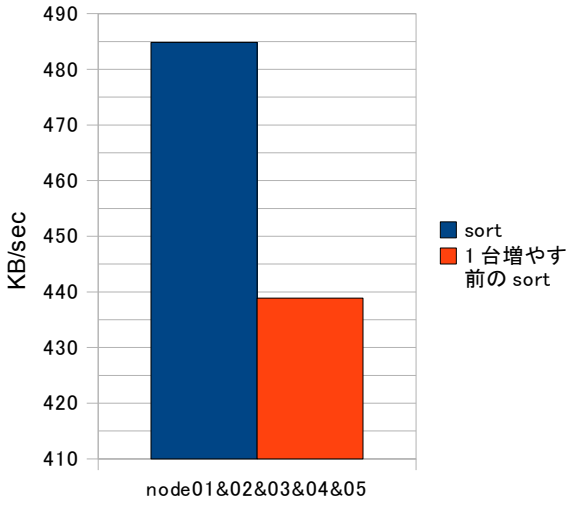

図 3.13.3　4 台及び 3 台の際の sort 性能

sort 性能について、グラフを見ると新しい計算機を追加するにつれ、ほとんどの組み合わせに性能は上がっている。write や read の時のように 5 台測定時にきちんと性能が上がっているのは、Map 処理にかかる時間よりもソートにかかる時間のほうが長いため、sort の性能は上がっていると考えられる。

第 4 章　　今後の課題

4.1.　ベンチマークテストの実行条件の変更

本研究では、書き込み・読み込み・ランダムサイズ書き込み・ソートのベンチマークテストにおいて、すべて 128MB で行っている。この測定のサイズを小さく、また大きくした際にどの程度予測値と差が生じるのか調査したい。また、測定回数も 3 回と若干少なく、正確なデータであると言えない面があるため、測定回数を増やしてより正確なデータを求める必要もあると思われる。

4.2.　性能低下の原因の調査及び解決

write,read,random write において、きっちりとスケールアウト性能を確保するために 5 台測定時の性能低下の原因の調査、及び解決案を考える必要がある。

謝辞

本研究を行うにあたり、御迷惑をおかけしながらも最後まで熱心な御指導をしていただきました田中章司郎教授には心より御礼を申し上げます。また、同じ研究室に所属する李昊さん、韋于思さん、三島健太さん、橋本沙優さんには本研究に関して、様々な御協力と御助言を頂きました。この場で厚く御礼申し上げます。

なお、本論文、本研究で作成したデータ、並びに関連する発表資料などの全ての知的財産権を本研究の指導教官である田中章司郎教授に御譲渡いたします。

参考文献

[1] Apache Hadoop
http://hadoop.apache.org/