Vol.2017-HPC-162 No.10
2017/12/18

Pwrake/Gfarmによる分散並列相同性検索システムの提案

町田 健太[1,a)]　建部 修見[2]

概要：メタゲノム解析とは，環境サンプルから直接得られたゲノム DNA を扱う研究である。この研究は，環境中に存在する膨大な数の未知の遺伝子を解明する方法として期待されており，近年シークエンサーの技術革新により得られるメタゲノムデータは急速に増大している。メタゲノム解析の一つに相同性検索というものがあるが，これは DNA 配列を比較して 2 つの配列の共通部分のアライメントを行うものである。本研究では，高速に相同性検索が可能なツールである GHOSTZ を拡張して，クエリと DB ファイルを多ノードに分散することで大規模な入力データに適応する分散並列相同性検索システムを提案する。結果としてこのシステムは，実行時間の面で大規模な入力データを用いた際にスケーラビリティを示した。

1. 序論

メタゲノム解析 [1] とは，ある環境中から直接得られたゲノム DNA を用いて，遺伝子プールを構成するゲノム配列をシーケンシングし，その集合の構造を明らかにすることで遺伝子プールの変動や環境との相互作用を解明することを可能にした解析手法である。従来のゲノム解析では，人間の手によって環境中から微生物を分離・培養・増殖させることが必要であったが，メタゲノム解析はその過程を経ずに微生物の集団から直接そのゲノム DNA を調整する事が可能であるため，従来の方法では困難であった難培養菌のゲノム情報が入手できるようになった [2]。地球上に存在するほとんどの細菌は単体では培養できないと推測されているため，メタゲノム解析は，環境中に存在する膨大な数の未知の細菌・遺伝子を解明する手法として期待されている。

メタゲノム解析の一つとして相同性検索というものがあるが，これは DNA 配列を比較して２つの配列の共通部分のアライメントを行うものである。相同性検索を行うことによって，例えば２つの DNA 配列が類似の部分列を共通に持っている場合，それらの配列は相同である可能性が高く，これらの DNA は共通の祖先に由来していると推測できる。この相同性検索を高速に行うアルゴリズムとして，FASTA[3] や BLAST[4], DIAMOND[5], GHOSTX[6], GHOSTZ[7] 等がある。BLAST は NCBI によって WEB 上で利用可能なサービスであり，ユーザがローカルに所持している塩基配列，あるいはアミノ酸配列を検索配列 (クエリ配列) として，GenBank/EMBL/DDBJ 等の公共データベースに登録されている塩基配列あるいはアミノ酸配列に対して相同性検索を実施し，その結果を WEB 上でユーザに提供する。DIAMOND は，高スループットな相同性検索ツールであり，高速モードではイルミナシーケンサから得られるデータに対し，BLAST が見つけ出した全てのアライメント結果の内 80~90%を見つけ出し，BLAST よりも 20,000 倍速く動作する。感度の高いモードでは，BLAST の全ての結果の 94%以上にマッチし，約 2,500 倍速いとされる。GHOSTX は，BLAST のような遠隔相同体を検出することができるものであり，接頭辞配列を用いることにより BLAST よりも約 100 倍効率的であるとされている。さらに，GHOSTZ は，データベース部分配列クラスタリングを使用することによって BLAST よりも約 200 倍効率的であるとされる。

2. 研究背景

相同性検索ツールの入力データとして用いられるシーケンスデータは，シーケンサ [8] と呼ばれる DNA の塩基配列を自動的に読み取り解析する装置によって得られる。DNA シーケンシングの手法が開発されたのは 1975 年頃であるが，それからシーケンサの技術革新は進み，2000 年半ばには米国で次世代シーケンサと呼ばれる，従来よりも高速に塩基配列を読み出せる装置が登場した。その結果 2003 年にはヒトゲノムの全塩基配列の解読に 13 年と 30 億ドルという費用がかかっていたものが，現在では，1000 ドルほどで数日間の内に解析が可能であるほどになった [9]。結果として，得られるゲノムデータサイズは爆発的に増大しており [10]，それを解析するコンピュータの計算性能の向上

[1] 筑波大学情報学群情報科学類
[2] 筑波大学計算科学研究センター
[a)] machida@hpcs.cs.tsukuba.ac.jp

が求められている。

現在最も高速に相同性検索を実行するツールに GHOSTZ があるが，GHOSTZ は単一のノード上での実行という制限があり増大するメタゲノムデータに対して HDD リソースが大量に浪費されてしまうという問題点がある。

そこで本研究では，入力データを多ノードに分散させることで大規模な入力データに適応し，かつ相同性検索を並列実行することで実行時間の効率化を図ったスケーラブルな分散並列相同性検索システムを提案する。

3. 関連研究

3.1 GHOST-MP

GHOST-MP は，GHOSTX の検索アルゴリズムを，MPI ライブラリを用いて並列化することで京や TSUBAME3.0 などの並列分散メモリシステム上での大規模並列検索に用いられるツールである [11]。mpirun や mpiexec を用いて hostfile に記述したホスト上で相同性検索を並列分散実行する。現在公開されているものでは，各実行ノードに分散配置されたファイルを扱うのではなく，一つのクエリ・DB ファイルに対して処理を行う。

また，前述した GHOST-MP は MPI ベースのマスターワーカーフレームワーク上で動作するが，これを MapReduce を用いて動作するようにする研究もある [12]。この研究は，予備実験として GHOSTX を Hadoop や Spark を用いて拡張した結果，オリジナルの GHOSTX・GHOST-MP と同等の性能であり，大きなデータセットを用いた際にはそれ以上の性能が出ることを示した。今後の研究には，MapReduce に適した分散型データベースや相同性検索アルゴリズムの設計・実装も含まれる。

4. 関連システム

本研究では，分散ファイルシステムとして Gfarm，動的ワークフローエンジンとして Pwrake を用いる。

4.1 Gfarm

Gfarm ファイルシステム [13] は，2000 年より研究開発が進められている広域分散ファイルシステムである。Gfarm ファイルシステムは**図 1** のように，システム全体を管理するメタデータサーバ (MDS) とファイルの実体を保持する計算機であるファイルシステムノード (FSN) で構成される。クライアントは，用意されたコマンド群を用いることで Gfarm ファイルシステムにアクセスすることができる。特徴としては，MDS のスレーブを設置することによる耐障害性の向上や，遠隔地でのファイル共有，スケールアウトアーキテクチャが挙げられる。特に，スケールアウトアーキテクチャに関して，ファイルシステム上のファイルを並列に読み書きする際にファイルアクセスが各ノードに分散していれば，MDS に対するアクセスがボトルネックにならない限り，合計の転送速度はノード数に対してスケールアウトする。また，Gfarm 上の実行プロセスとファイルの実体が同一ノード上にある場合，プロセスによるファイル I/O はローカル読み書きとなり，スケールアウトの効果の向上が期待できる。

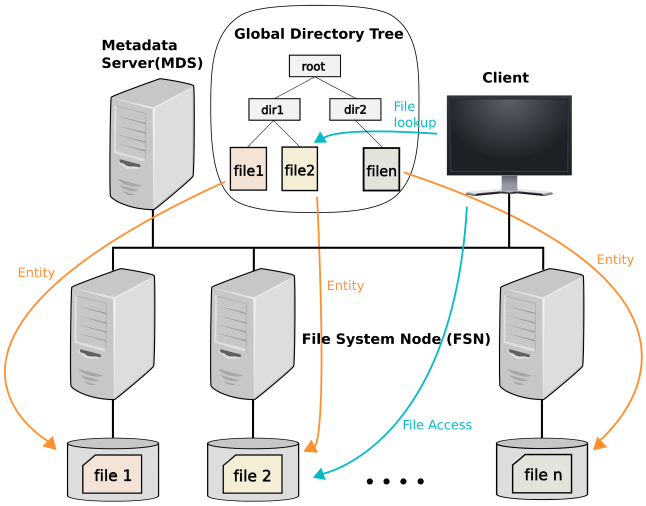

図 1 Gfarm ファイルシステム

4.1.1 Gfarm2fs

Gfarm2fs は，FUSE の仕組みを用いてユーザごとに Gfarm をマウントするコマンドである。Gfarm をマウントすることで，gf で始まる専用のコマンドを用いなくても通常のコマンドによるファイル操作が可能となり，C 言語の stdio 等の標準の I/O 操作も可能となる。

4.2 Pwrake

C 言語で実装されたプログラムのビルド作業を自動化するツールとして make があるが，ruby で実装された同様なツールに Rake がある。Pwrake とは並列分散実行可能な Rake である [14]。ワークフロー記述言語は Rake 用の Rakefile と同様であり，依存関係のないタスクを自動で並列実行し，ssh 接続を用いて複数の計算ノードを用いた分散実行が可能である。Gfarm ファイルシステムを用いた場合，出力ファイルの格納先としてローカルストレージが選択され，また Pwrake のスケジューリング機能により入力ファルが保存されている計算ノードをタスク実行ノードとして選択するため，Pwrake は Gfarm と親和性が高いワークフローエンジンとなっている。

4.2.1 Rakefile の記述

Rakefile には，実行したい処理をタスクとして記述する。記述例は**図 2** のとおりである。

タスクは task の記述に始まり，続いてタスク名を文字列かシンボルで指定する。タスクの依存関係は 依存元 => 依存先 のように記述し do に続いてタスク処理を記述する。default という名前のタスクの依存先がデフォルトで実行されるタスクである。

```
CC = "gcc"
task :default => "hello"
  sh "#{CC} -o hello hello.o world.o"
end

file "hello.o" => "hello.c" do
  sh "#{CC} -c hello.c"
end

file "world.o" => "world.c" do
  sh "#{CC} -c world.c"
end
```

図 2 Rakefile の記述例

4.2.2 Pwrake の実行

Pwrake の実行には，pwrake コマンドを用いる。

```
pwrake -f Rakefile
```

Pwrake が起動すると，カレントディレクトリ以下にある pwrake_conf.yaml ファイルより使用するホスト名及びコア数，ファイルシステム等の設定を読み込む (引数でも指定可能)。Pwrake の動作の概要を**図 3** に示す。

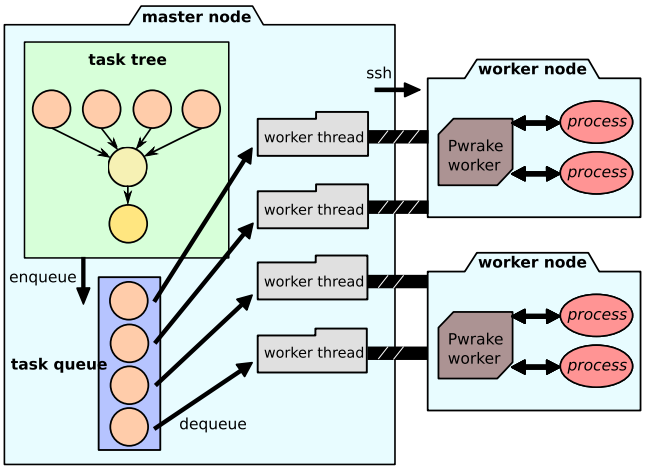

図 3 Pwrake の動作概要

Pwrake は，最終的に実行されるタスクを起点に深さ優先探索を行い，直ちに実行できるタスクがあれば，その動作をその都度実行せずに (Rake はその都度実行) タスクをキューに入れて探索を続ける。探索終了後キュー内のタスクを並列実行し，キュー内のタスクが無くなったら再度探索を実行する。各ワーカスレッドにおいて各タスクは ssh によって接続先の担当ノードに渡され，プロセスが生成されてタスクが実行される。

4.3 GHOSTZ

GHOSTZ は，東京工業大学の秋山らによって開発された高速な相同性検索ツールである。**図 4** で示すように ghostz db コマンドによって DB ファイルを生成し，**図 5** で示す ghostz aln コマンドによってクエリと DB ファイルとのアライメントを実行する。

```
ghostz db -i fastafile_name -o dbfile_name
                    — argument —
    -i STR    Protein sequencein FASTA
              format for a database
    -o STR    The name of the database
    -l INT    Chunk size of the database
              (bytes) [1073741824 (=1GB)]
```

図 4 ghostz db コマンド

```
ghostz aln -i query_name -d dbfile_name
                              -o output_name -q d -a 1
                    — argument —
    -i STR    DNA or protein sequences in
              FASTA format for queries
    -o STR    Output file
    -d STR    database name (must be formatted)
    -q STR    Query sequence type
              p (protein) or d (dna) [p]
    -a INT    The number of threads [1]
```

図 5 ghostz aln コマンド

入力データのフォーマットには FASTA を用いることができる。FASTA ファイルは，**図 6** のように > で始めるヘッダ部 (データの説明) と実際の配列情報を記述するデータ部から成り，一つのシーケンスを構成する。FASTA ファイルにはこのシーケンスが複数行記述されている。ghostz db コマンドによって FASTA ファイルから DB ファイルを生成すると**表 1** のようなファイル群が生成される。表 1 は，チャンクサイズが 1 GB の FASTA ファイルを ghostz db コマンドによって DB ファイルにしたものである。DB ファイルのチャンクサイズは表 1 の.seq ファイルに相当し，これは db コマンドのオプションとして変更することが可能である (デフォルトは 1 GB)。また，GHOSTZ は CPU を用いて相同性検索を行うが，GPU を用いる GHOSTZ-GPU もある。

```
>61G9GAAXX100521:7:100:10001:11453/2
AAGTTTAGAGATTTAGGTGTTCCAGAATCATATATAACATTAATGGGATATCAATC
>61G9GAAXX100521:7:100:10001:20977/2
CAAGATTTGTCTGCATGGAATTTTGATTACAAGATTTAGGTGTTCATGATAG
>61G9GAAXX100521:7:100:10001:3436/2
CGATACCAATTTAGGTGTTCCAGAATATGAGCTGGGCATCAACCTTCTAAAGCC
                    ⋮
```

図 6 FASTA ファイル

表 1　各 DB ファイルのサイズ

	size[MB]
.inf	1
.nam	1577
.seq	1028
.src	4615
.off	13

5. 事前実験

提案システムを設計・実装する準備としていくつかの事前実験を行った。実験環境を**表 2** に示す。chris は筑波大学のストレージクラスタの一つである。また，ppx は Pre-PACS-X の略であり筑波大学が研究開発しているスーパーコンピュータ PACS シリーズの次世代機 PACS-X の実験機にあたる。

表 2　実験環境

	chris	ppx
OS	CentOS 6.9	CentOS 7.3.1611
CPU	Intel Xeon E5-2665	Intel Xeon E5-2660 v4
memory	64GB	64GB
GPU	-	NVIDIA P100 * 2

各使用ソフトウェアのバージョンは，ghostz-1.0.0, gfarm-2.7.6, gfarm2fs-1.2.11, pwrake-2.2.4, ghostmp-1.3.4, OpenMPI-3.0.0 である。また，GHOSTZ・GHOST-MP の入力データとしてはクエリに
https://www.hmpdacc.org/HMASM/
からダウンロードできる SRS011098 という歯肉縁上プラークの DNA 配列を記述した FASTA ファイルを用いる。また，DB ファイルの生成元データには NCBI の公開している nr という配列データを用いる。
ftp://ftp.ncbi.nlm.nih.gov/blast/db/FASTA/nr.gz
それぞれのファイルに関して，SRS011098.fasta は 5.34 GB nr は 61.17 GB とサイズが大きいため，適宜 split コマンドを用いてリサイズしたものを使用する。実行時間等の測定には GNU time を用いた。

5.1 GHOSTZ

事前実験の一つとして，GHOSTZ の性能・特徴を理解するための諸々の測定を行った。

5.1.1 スレッド数による実行時間の変化

図 5 で示したように，GHOSTZ はマルチスレッドで実行することができる。**図 7** に，コア数 (スレッド数) を増やした時の実行時間の変化を示す。

CPU, GPU 共にスレッド数が増えるに従って実行時間が短縮しているのが見て取れる。また，GPU を用いた場合は CPU の約 5~6 倍実行時間が速いことがわかる。

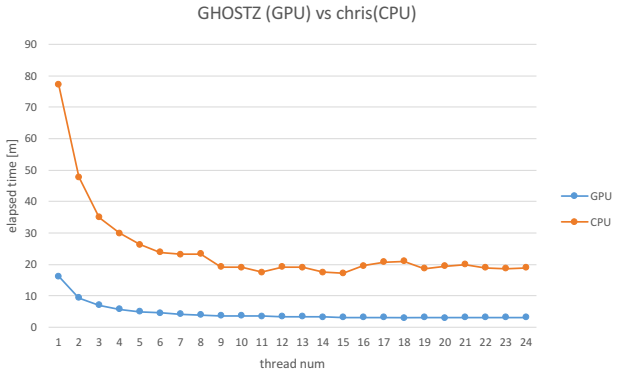

図 7　スレッド数に対する実行時間の変化

5.1.2 入力データと実行時間の相関

次に，DB ファイルを固定にしてクエリサイズを可変にしたときのアライメントの実行時間の変化，及びクエリを固定して DB ファイルの生成元データのサイズを可変にしたときの実行時間の変化を測定した。前者の測定結果を**図 8** に，後者の測定結果を**図 9** に示す。また，前者に関しては DB ファイルの生成元データのサイズは 1 GB，後者に関してクエリサイズは 100 MB のものを使用している。

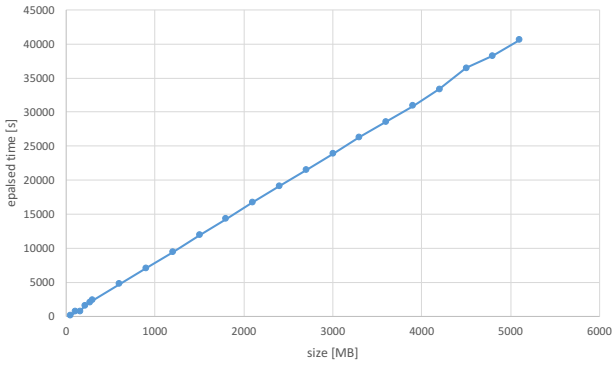

図 8　クエリサイズに対する実行時間の変化

以上の測定により，GHOSTZ の実行時間は入力データのサイズと強い相関がありほぼ比例する事がわかる。よって，入力データを分割して多ノードで分散並列実行することで実行時間は分割した分だけ速くなると推測できる。

5.1.3 入力データとメモリ使用量

また，前節の測定において実行時間と同時に得られた Max Resident Set Size(プロセスの最大メモリ使用量) の入力データのサイズに対する変化をクエリ可変 (DB ファイル固定) のものを**図 10** に，DB ファイル可変 (クエリ固定) のものを**図 11** に示す。ここで図 11 に関して，x 軸

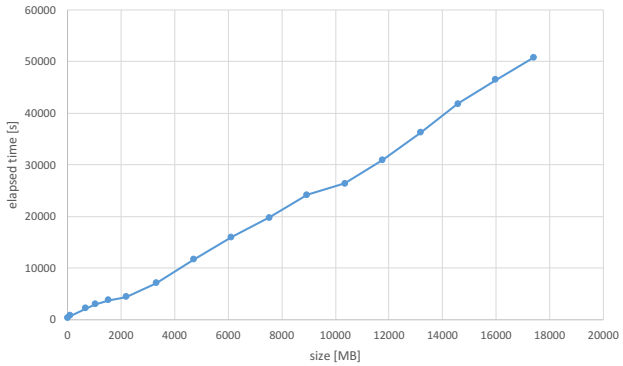

図 9 DB ファイルの生成元データのサイズに対する実行時間の変化

は DB ファイルのチャンクサイズではなく生成元データのサイズであることに注意されたい。参考までに，入力として 4.5 GB のものを用いた際には ghostz db 実行時に std::bad_alloc エラー (メモリ不足) となった。

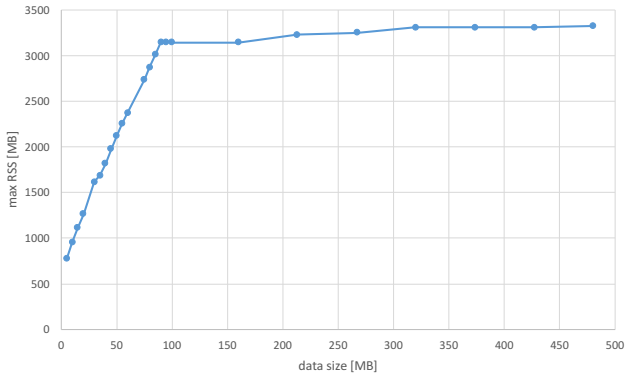

図 10 クエリサイズに対する maxRSS の変化

以上の測定より，DB ファイルに関してはサイズと RSS に相関があり，データサイズに応じて増加しているのがわかる。また，クエリに関しては一定サイズ (およそ 100 MB) 以降は RSS は一定値を取ることがわかった。これより，アライメントの際にクエリはサイズに依らず一定サイズ分がメモリに載せられ，DB ファイルはチャンクサイズ分を一気にメモリに載せているということが予測できる。

5.2 Gfarm

5.2.1 リモート/ローカルデータに関する GHOSTZ

Gfarm ファイルシステム上では，ローカルストレージに保存されているファイルを参照する場合にはローカルアクセスとなるが，他 FSN に実体があるファイルへのアクセスはリモートアクセスとなり通信が発生する。GHOSTZ

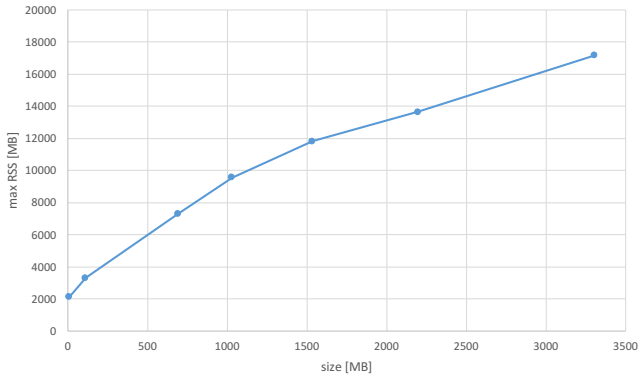

図 11 DB ファイルの生成元データのサイズに対する maxRSS の変化

の実行においてクエリをローカルノード/リモートノードに置いた場合に関して実行時間の違いを測定するための実験を行った。結果を**図 12** に示す。

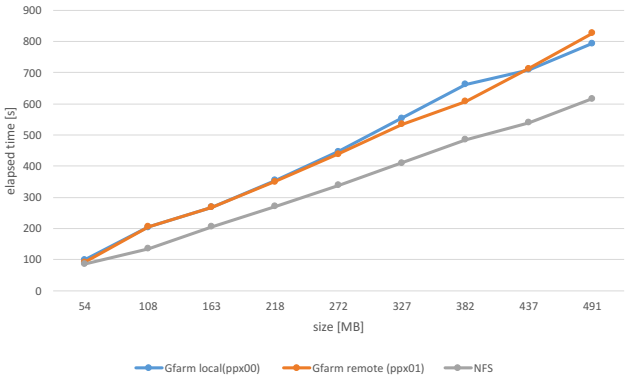

図 12 クエリの Gfarm リモート/ローカル読み込み及び NFS における GHOSTZ の実行

ここで，図 12 において DB ファイルに関してはローカル読み込みである。Gfarm ファイルシステムを ppx の 2 ノードに渡って構成し，一方で測定を行った。結果として，クエリに関してはリモート/ローカルで実行時間 (IO 時間) にあまり差異は見られなかった。

次に，クエリをローカル読み込みとし，DB ファイルをローカル/リモートノードに置いた場合に関して同様な実験を行った結果を**図 13** に示す。

リモートにある場合の方がローカルにある場合よりも実行時間 (IO 時間) が長くその差異は明らかである。そのため，DB ファイルはローカルのものを使用しクエリに関してはローカル/リモート読み込みとする。

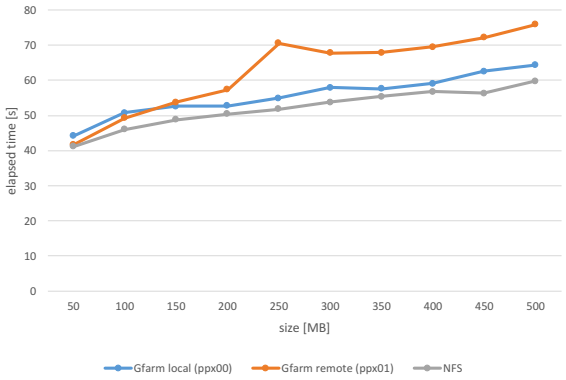

図 13 DB ファイルの Gfarm リモート/ローカル読み込み及び NFS における GHOSTZ の実行

5.2.2 Gfarm2fs のマウント及びバックエンドデータベース

4.1.1 節で述べたように Gfarm は Gfarm2fs を用いてマウントすることができる。Gfarm2fs は FUSE を利用しているので FUSE のオプションとして –o direct_io を指定することでダイレクト I/O を用いることができる。ダイレクト IO とは，ファイルキャッシュを経由せずに直接ディスクにアクセスすることができる機構である。また，Gfarm のバックエンドデータベースとしてデフォルトの postgreSQL を用いずに DB を介さないこともできる。これらについて GHOSTZ の実行時間を測定した結果を**図 14** に示す。

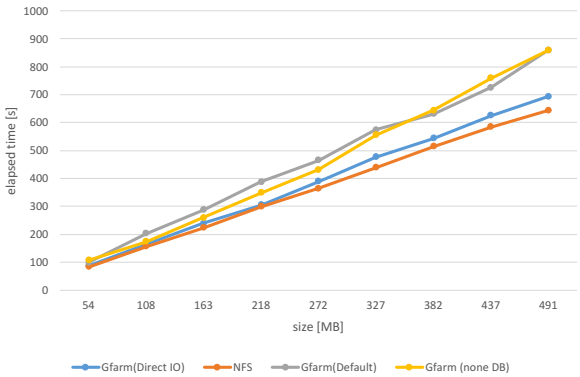

図 14 directIO/noneDB の効果

これより，バックエンド DB に関してはデフォルトと比べて noneDB の方が十分速いとは言えないが direct_io に関しては，実行時間がほとんど NFS と変わらなく Gfarm のデフォルト I/O よりも十分速いということがわかった。

6. 分散並列相同性検索システムの設計と実装

6.1 システムの設計

5 章での事前実験の結果，クエリと DB ファイルの生成元データを分割し Gfarm 上の FSN に分散配置して，実行ステップ毎に担当クエリを交換し全ての組み合わせについて GHOSTZ を並列実行させることで，実行時間，使用メモリ量共にスケールアウトすることが期待できる。提案システムの概要を**図 15** に示す。

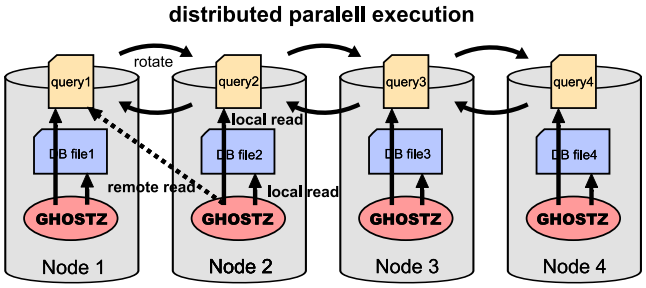

図 15 提案システムの概要

図 15 において注意することは，各ノードが担当する DB ファイルは固定とし (常にローカルのものを使用)，クエリファイルの実体は動かさず参照先を回転させるということである。そのため，例として Node 2 における処理を考えると step 1 では query 2 へのローカルアクセスとなるが，step 2 では担当クエリの参照を回転させるため Node 1 に実体のある query 1 へのリモートアクセスとなる。図の例では GHOSTZ の分散並列実行を 4 step 行うことで全てのクエリ・DB ファイルの組み合わせについてのアライメントが完了する。

6.2 システムの実装

提案システムのワークフローを Rakefile に記述することによって分散並列実行を実現する。Rakefile の存在するディレクトリ以下にある FASTA 形式のファイルを取得し，その所在地を gfwhere コマンドによって得てそれを元にタスクをスケジュールする。なお，各クエリ・DB ファイルは事前に各ノードに配置するか，または実行時に一つの大きなサイズのファイルを分割・分散配置してから実行することが可能である。Pwrake のワークフローを**図 16**, **図 17** に示す。

図 16 は事前タスクのワークフローである。split query/dbfile はファイルサイズ (塩基配列のシーケンス数) を元に分割後のサイズ (シーケンス数) が均等になるようにファイルを分割する。ファイル分割プログラムは C++で実装した。ファイルの分割における GHOSTZ の実行時間のバランスについては後述する。ファイル分割は，クエリと DB ファイルの生成元データについて 2 並列で

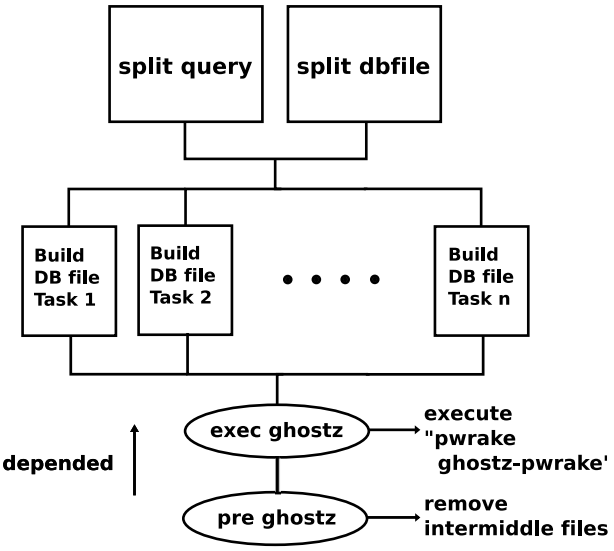

図 16 Pwrake ワークフロー (事前タスク部)

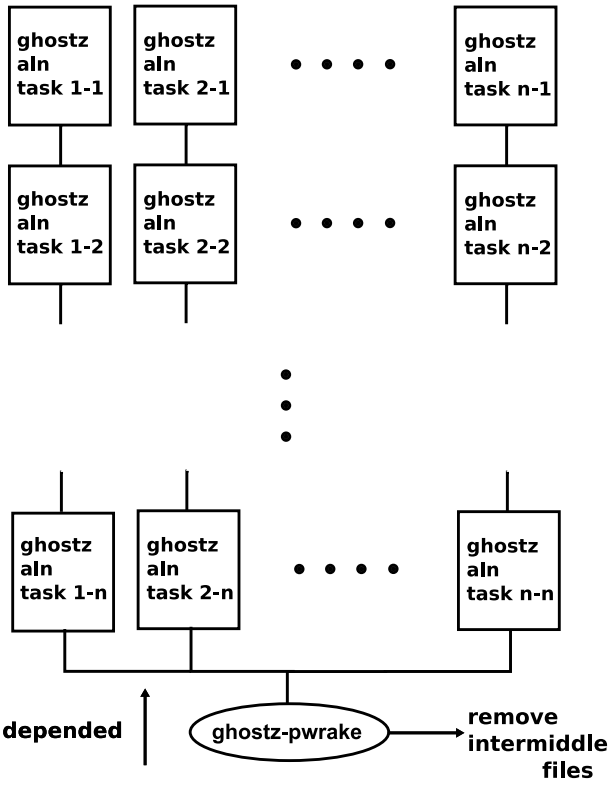

図 17 Pwrake ワークフロー (実行タスク部)

実行される。ファイル分割が完了したら，ghostz db コマンドによって各ノードの保持する DB ファイルの元データから DB ファイルを生成する。その後生成した DB ファイルを gfrep -m コマンドによって各ノードに再配置する (ghostz db によって生成する DB ファイルの所在地は指定することが出来ないため)。以上のタスクが完了すると ”pwrake ghostz-pwrake” をシェル上で実行し，アライメント実行タスクである ghostz-pwrake タスクが生成・実行される (図 17)。ghostz-pwrake タスクは ghostz aln コマンドによって配列のアライメントを実行するタスクである。n 分割されたファイルが配置されているノード上で実行されるため n 並列である。これを n ステップ実行する。最後に中間ファイルを削除して全タスクの実行が完了する。

7. 性能評価

7.1 提案システムの評価

実装したシステムを，クエリ 2 GB，DB ファイルの生成元データ 5 GB のものを用いて各ノード上のプロセス数を 1 つとして実行した結果を**図 18** に示す。なお，実験環境は 5 章で示した chris を 7 ノード使用しそのうちの一つが MDS，7 つが FSN である。また，結果は GHOSTZ のアライメント実行部 (図 17) のものであり時間の測定には GNU time を用いた。

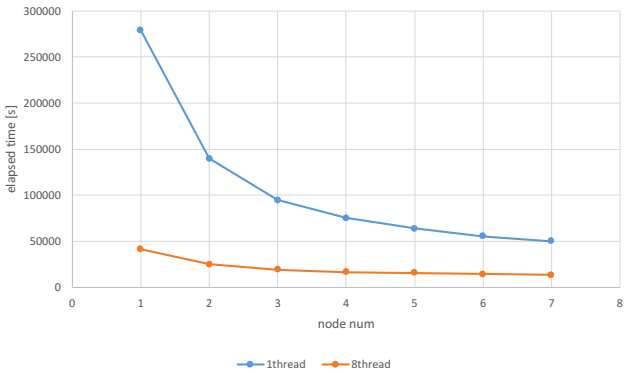

図 18 ghostz_pwrake の実行結果

また，Pwrake の report 機能によって得られた各ノード上の並列度として 1 thread 7 ノードで実行したものを**図 19** に示す。

図 19 は，縦軸が各ノードの並列度，横軸が経過時間を表す。開始から 45,000 秒あたりまでは各ノード上で 1 並列，つまり各ノード上で 1 プロセスが動作し全体としては 7 並列で動作している。

図 18 において，8 thread を用いた結果は 1 thread を用いた結果よりもスケールアウトの度合いが小さいように見える。つまり，ノード全体の述べ実行時間の増加率が 1 thread よりも 8 thread の方が高い。**図 20** は，クエリと DB ファイルのサイズを x 軸の値と同じにした場合に GHOSTZ の実行時間がどう変化するかを示している。

5.1.2 節の結果より，クエリ・DB ファイルのサイズがそれぞれ 1/2 になったとすると，実行時間はおよそ 1/4 になると考えられるが図 20 はそうなっていない。これには，Pwrake の通信時間や nr のサイズに対する実行時間のグラフはきれいな線形になっていない等の要因が考えられるが，図を見てわかるようにサイズが大きくなるほどグラフ

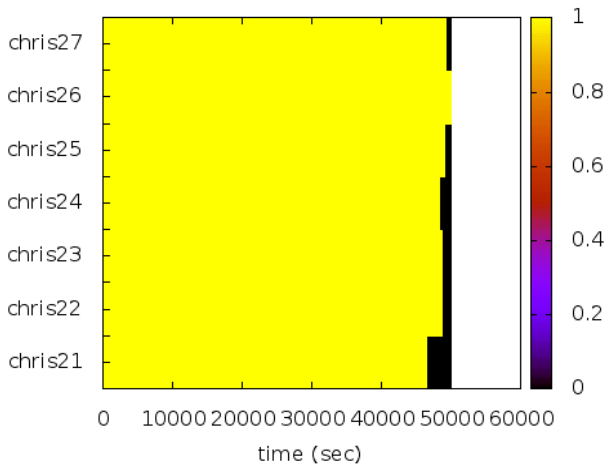

図 19 ghostz_pwrake 並列度

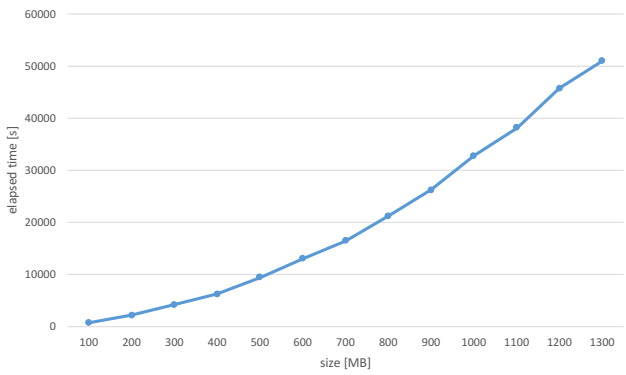

図 20 同サイズのクエリ・DB ファイルに対する GHOSTZ の実行時間

の概型は $y = x^2$ に近づき，延べ実行時間の増加率は抑えられると予測できる。

7.2 ファイル分割による実行時間のバランス

図 19 より，各ノード間で実行時間に差が見られる。これには，入力データの分割方法が強く影響を与えている。FASTA ファイルの分割方法として，真っ先にサイズによる分割が考えられるが，実験を通して実行時間には入力データのサイズというよりもシーケンス数が特に影響を与えていることがわかった。ファイルサイズとシーケンス数には強い相関があるが例えば，サイズが大きくてシーケンス数が小さいファイルというのは，シーケンスのヘッダが膨大な量記述されているものが考えられる。このようなファイルの場合には，シーケンス数で分割してしまうと，サイズが極端に大きいファイルが出てきてしまい実行時間のバランスが大きく崩れてしまう。そのため，今後の課題として GHOSTZ の実行時間と入力データのシーケンス数及びサイズ等の関係を調べ，各ノードの実行時間のバランスを保つような分割方法を考案する必要がある。

7.3 GHOST-MP との比較

関連研究との比較の一つとして，GHOST-MP を 7.1 章の ghostz_pwrake の測定の際に使用したのと同じデータを用いて同様の測定を行おうとしたが，2~4 ノードにおいて std::bad_alloc のエラー (メモリ不足) によって測定することができなかった。参考までに，5 ノードを用いた時の実行時間は 142,835 秒であった。5 ノードの実行においては提案システムの方が実行時間が速い。そのため今後の課題の一つとして，GHOST-MP においてエラーが出ないサイズのものを用いて比較を進めていきたいと考えている。

8. 結論

本研究を通して，使用メモリ・実行時間ともにスケールアウトする分散並列相同性検索システムを提案した。

今後の課題として，7.2 節で述べた各ノードの実行時間のバランスを保つのが困難な問題に着手していきたい。現在の考えでは，まずシーケンス数等の各パラメータによって実行時間を予測するモデルを構築し，ファイル分割プログラムにおけるファイル走査の際に，ヘッダの文字数が大きいシーケンスに関してタグを付ける。その後，モデルを元にシーケンスの独立性を利用して必要であればシーケンスの入れ替えを行いファイルを分割するという方法を考えている。さらに，小さいサイズのデータを用いた際にスケールアウトしない原因の追求や関連研究との比較を進めることで提案システムの有用性を示していきたい。

謝辞 本研究の一部は JST-CREST JPMJCR1303「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 JPMJCR1413「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」，JSPS 科研費 JP17H01748 による．

参考文献

[1] Handelsman, J., et al：”Molecular biological access to the chemistry of unknown soil microbes : a new frontier for natural products”，Chemistry and Biology 1998, vol. 5, no. 10, p. 245-249
[2] 森 宙史, 山田 拓司, 黒川 顕：メタゲノム解析の現状と将来知識データベースの開発，情報管理 Vol. 55(2012) No. 12 .
[3] Lipman, D.J. & Pearson, W.R. Science 227, 1435-1441.(1985)
[4] Stephen F.Altschul, Warren Gish, Webb Miller, Eugene W.Myers , David J.Lipman1Basic：local alignment search tool，Biol. 1990,vol.215(pg. 403-410)
[5] Benjamin Buchfink, Chao Xie & Daniel H. Huson：Fast

and Sensitive Protein Alignment using DIAMOND，Nature Methods, 12, 5960 (2015) doi:10.1038/nmeth.3176.

[6] Shuji Suzuki, Masanori Kakuta, Takashi Ishida, and Yutaka Akiyama：GHOSTX: An Improved Sequence Homology Search Algorithm Using a Query Suffix Array and a Database Suffix Array，PLoS One. 2014; 9(8):e103833.

[7] Shuji Suzuki, Masanori Kakuta, Takashi Ishida and Yutaka Akiyama：Faster sequence homology searches by clustering subsequences，Bioinformatics (2014) doi: 10.1093/bioinformatics/btu780

[8] 先端メタゲノミクス推進センター：ゲノムの歩き方 (用語解説)，入手先 ⟨http://genome.nig.ac.jp/glossary/glossary11.html⟩(2012)

[9] TechCrunch Japan：ゲノム学がもたらす遺伝子情報革命，入手先 ⟨http://jp.techcrunch.com/2017/01/24/20170121the-genomics-intelligence-revolution/⟩(2017)

[10] illumina 社：微生物研究における次世代シーケンサーがもたらす利点，入手先 ⟨https://jp.illumina.com/content/dam/illumina-marketing/apac/japan/documents/pdf/primer_sequencing_introduction_microbiology.pdf⟩(2014)

[11] Yutaka Akiyama, GHOST-MP: an ultra-fast and high-sensitive homology search tool for metagenome analysis，ADAC Workshop, 2016.

[12] Chaojie Zhang, Koichi Shirahata, Shuji Shuzuki, Yutaka Akiyama, Satohsi Matsuoka：Performance Analysis of MapReduce Implementations for High Performance Homology Search，IPSJ SIG Technical Report Vol.2014-HPC-147 No29 (2014)

[13] Osamu Tatebe, Kohei Hiraga, Noriyuki Soda：Gfarm Grid File System，New Generation Computing, Ohmsha, Ltd. and Springer, Vol.28, No.3, pp.257-275, 2010

[14] Masahiro Tanaka, Osamu Tatebe：Pwrake: A parallel and distributed flexible workflow management tool for wide-area data intensive computing，IProceedings of ACM International Symposium on High Performance Distributed Computing (HPDC), pp.356-359, 2010