Vol.2019-HPC-170 No.31
2019/7/26

大規模ゲノムデータに対す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：ゲノムデータは，代表的なビッグデータの一つであり，次世代シーケンサの性能向上や普及により得られるデータの量は年々増大している．ゲノム解析の一つである相同性検索は，未知の DNA 配列をクエリとして，既知の DNA 配列データベースから相同なものを収集する手法である．現在までに様々なアルゴリズムを用いた相同性検索ツールが開発されており，分散並列処理フレームワークを用いることで，これを高速に実行させる研究も近年行われている．本研究では，分散ファイルシステムである Gfarm を基盤として開発されている分散並列相同性検索システムについて，システムのチューニングを行いパフォーマンスを向上させた．また，関連研究との比較を通してシステムの有用性を確認した．

1. はじめに

細菌は地球上の生物を含むあらゆる環境中に存在しており，周囲の環境と共生・総合扶助の関係を持ち，独自のエコシステムを築き上げている．ヒトは自身の細胞数の 10 倍以上に相当する細菌を有しているとされており，それらの細菌叢はヒトが食べた物を栄養分に代謝したり，病原性細菌からの感染を防いだりと，ヒトの健康状態の維持や発病の抑制に関与している [1]．一方，これらの細菌叢の異常が人体に及ぼす影響も大きく，生活習慣病やがん，アレルギー，自閉症など様々な疾患の原因となることが明らかになっている．そのため，細菌叢の群集構造や機能を明らかにすることがこれらの疾患の治療や予防に繋がると期待されているが，地球上のほとんどの細菌は培養が困難であるため，従来の手法では解析が容易ではなかった．

メタゲノム解析は，ある環境中から直接得られたゲノム DNA を用いて，その集合構造を明らかにすることにより，遺伝子プールの変動や環境との相互作用の解明を可能にしたゲノム解析手法である [2]．メタゲノム解析は細菌に対する培養過程を経ずに細菌叢から直接そのゲノム DNA を調製する事が可能であるため，ヒトの体内の微生物群集構造を明らかにする手段として有用である．また，メタゲノム解析は数千種類もの生物を同時に解析することが可能な次世代シーケンサー (NGS) テクノロジーと親和性が高く，近年その関連論文数は増加し続けており [3]，メタゲノム関連市場のさらなる活性化も予測されている [4]．

メタゲノム解析のプロセスは大まかに以下のステップを踏む．まず初めに，被験者のメタデータとしてサンプルを取得し，そのサンプル中から細菌叢 DNA を NGS により直接シーケンシングする．さらに，シーケンシング結果のリードをクオリティなどでフィルタリングする．続いて，KEGG[5][6] や COG[7] 等が提供するリファレンス DB に対し類似度や相同性の検索を行うことで，各遺伝子の機能や代謝経路等の情報を収集し，含まれる種の構成と存在量などを知る．そして，必要であれば可視化やさらなる解析を実施し，環境中の微生物や細菌の群集構造や機能を推測する．

近年，NGS の性能向上や普及により，リファレンス DB だけではなくサンプルから取得されるゲノムデータの量も爆発的に増大しており，DNA の相同性の検索ステップにおいて，従来の解析手法では膨大な時間を要してしまう点が問題となっている．

2. 背景

相同性検索ツールとして，様々なアルゴリズムを用いたソフトウェアの開発が現在に至るまで行われている．BLAST[8] は，NCBI によって提供されている有名なツールであり，ここ 20 年間で最も広まったアプローチであるが，近年のゲノムデータの増大する速度に対して，生命科学の解析サイクルでその効率性がボトルネックとなっている．その後，GHOSTX[9]，GHOSTZ[10]，DIAMOND[11] などといった相同性検索ツールが開発されてきた．GHOSTX は接頭辞配列を用いることにより，BLAST の約 50 倍高

[1] 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻
[2] 筑波大学計算科学研究センター
[a)] machida@hpcs.cs.tsukuba.ac.jp

速であるとされ，GHOSTZ はデータベース部分配列クラスタリングを用いることで，BLAST の約 100 倍の高速化を達成している．また，DIAMOND は高速モードにおいて，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，約 20000 倍の高速化を達成している．

これらのツールは，単一ホスト上での実行を想定して開発されており，増大するゲノムデータ，特にメタゲノムデータのような大規模な入力に対して，メモリの枯渇や実行時間の肥大化が問題となってしまう．そこで，広域分散ファイルシステム Gfarm と動的なワークフローエンジンである Pwrake を用いて，クラスタ上にリファレンス DB とクエリ DNA を分散させ，相同性検索ツールを並列実行することによりこれらの問題の解決を目指すシステムの開発が行われている．[12].

本論文では，以下このシステムを GHOSTZ PW/GF と呼称し，システムのチューニングや関連研究との比較について述べる．

3. 関連研究

相同性検索を分散並列実行環境に拡張することで，処理の高速化とメモリの枯渇という問題を解決する研究がいくつか行われている．分散処理フレームワークとして有名なものに MPI と MapReduce があり，これらを用いた研究が多い．MPI を用いた実装では，その利点として細かな部分まで性能をチューニングすることができるため，比較的少量のデータセットに対する実行に関しては，一般に Hadoop や Spark における実行よりも高速に動作させることが可能である．しかしながら，MPI は通信を用いた分散処理インターフェースであるため，所謂ビッグデータと呼ばれるような大規模なデータセットに関して，低速な通信がボトルネックになってしまう可能性がある．Hadoop や Spark はファイルシステムとして HDFS を採用し，ファイルのローカリティを活かす設計となっているため，高速なローカルディスク I/O を生かすことが出来，通信がボトルネックとなり難いという利点がある．以下では，これらのフレームワークを用いた研究についてその概要を説明する．

3.1 MPI-BLAST[13]

MPI を用いて BLAST の処理を並列化することで高速化を達成している．並列化の方針として，BLAST の DB をセグメント化し，クラスタの各ノードに固有の DB の一部に対して検索を行う．DB を小さくセグメント化することにより，Disk I/O を削減し，BLAST の性能を改善している．また，DB のセグメンテーションは大量の通信を発生させないため，低コストで効果的な Linux クラスタのアーキテクチャを活かすことができる．
結果として，DB のセグメンテーション化は BLAST に対し線形に近い高速化を達成し，全体の実行時間に占める通信時間や結果のマージ及び出力時間が BLAST の検索時間に隠蔽されることにより，少なくとも 100 ノードに対して性能がスケールしている．

3.2 GHOST-MP[14]

GHOST-MP は GHOSTX のアルゴリズムを 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．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．この研究は，結果としてスケールアウトする性能を示したが，メモリ使用量の見積もりを課題としている．

3.3 HBLAST[15]

この研究は，BLAST を Hadoop を用いて分散並列処理するというものであり，クエリと DB ファイルの両方を分散する．また，データベースの virtual pertitioning を用いることで，データベースのパーティションサイズを Hadoop クラスタのノード数や入力クエリシーケンス数に従って調整する．CloudBlast との比較では，比較的大きいデータサイズの入力に対して約 2.5 倍速く動作することを示した．

3.4 SparkBLAST[16]

この研究は，BLAST を Spark を用いて分散並列実行するというものであり，クエリはクラスタ上に分散するが DB ファイルは分散しない．Spark の pipedRDD を用いることで BLAST を実行するジョブをクラスタ上の各ノードで実行する．Google Cloud と Microsoft Azure を用いて，Hadoop ベースの相同性検索ツールである Cloud BLAST と比較を行っており，どちらの環境においても Cloud BLAST よりも優れた性能を示している．この要因として，Spark の RDD によるインメモリ処理とそれに起因する I/O 時間の削減であると言及している．

3.5 HAMOND[17]

この研究は，DIAMOND を Hadoop を用いて分散並列処理するというものである．AWS の EMR と互換性があり，GUI での操作が可能であるため，専門知識のない生物学者向けの UI 設計がなされている．なお，DB は分散せずにクエリのみを分散する実験では，シングルホストでの DIAMOND の実行 (10 スレッド) と比較しており，HAMOND において 100 スレッドを用いた際に約 10 倍の高速化を達成している．

3.6 GHOSTMP MapReduce[18]

GHOST-MP を MPI ではなく MapReduce を用いて動

作するようにしたものがこの研究である．この研究は，大きなデータセットを用いた際に MPI の実装以上の性能が出ることを示しているが，ファイル配置やファイル分割・タスクの粒度などの最適化を課題としている．

4. 提案システム

本研究で提案する手法は，広域分散ファイルシステムである Gfarm[19] で構成されたクラスタ上の各ノードに入力ファイルを分散配置し，Gfarm と親和性の高い動的ワークフローエンジンである Pwrake[20] を用いてタスクをスケジューリングすることで，相同性検索ツールを並列実行するというものである．提案システムの概要を**図 1** に示す．

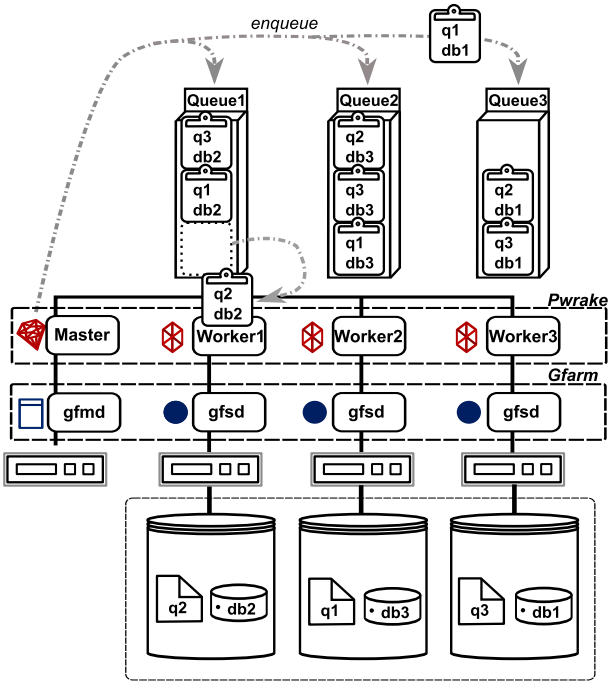

図 1 提案システムの概要

Gfarm ファイルシステムは，メタデータサーバである gfmd とファイルサーバである gfsd から構成される．クライアントはファイルシステム上のファイルアクセスする際にメタデータサーバからファイルの所在地を聞き，ファイルサーバと直接データのやり取りを行う．Pwrake は，マスターが Rakefile に記述されたタスクの依存関係を基に DAG を生成し，ワーカースレッドが動作する各ノードのタスクキューにそのノードの実行候補タスクをエンキューする．各ワーカスレッドはキューからタスクをデキューしタスクを実行する．Pwrake は ruby で実装されており，Rakefile と呼ばれるファイルにタスクの依存関係を記述することで，ssh 接続を用いて自動的にタスクを並列実行する．また，本研究では相同性検索ツールとして GHOSTZ を用いる．GHOSTZ は 2 入力 (クエリ・DB)1 出力 (相同性検索結果ファイル) の相同性検索ツールであり，OpenMP を用いたマルチスレッドによる高速化が施されている．

4.1 ワークフロー

本システムにおける相同性検索のワークフローの概要を**図 2** に示す．

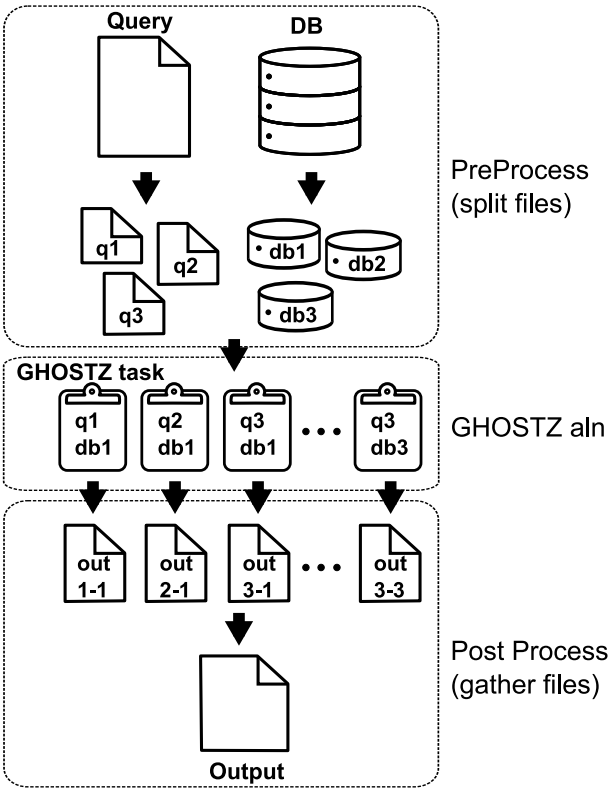

図 2 システムのワークフロー

4.1.1 前処理 (ファイル分割タスク)

システムの入力は，ゲノム配列が羅列された FASTA フォーマットのクエリファイルと DB の元となる FASTA ファイルである．初めに相同性検索の前処理としてこれらの入力に関して，ファイルを分割し各ノードに分散配置する．この際に用いるファイル分割プログラムは C++で実装されており，本研究における高速化の対象である．分割されたファイルのうち DB の元となるファイルは GHOSTZ を用いて DB にビルドされ，全てのファイルはノード間にバランス良く配置される．またこのステップにおいて，入力ファイルの複製を複数ノードに持たせることにより，後の相同性検索タスクにおいて通信の発生を抑え，高速なローカル I/O による read の割合を高めることが可能である．

4.1.2 相同性検索実行タスク

前処理が完了すると，GHOSTZ のアライメントを実行するタスクが並列に実行される．GHOSTZ の入力のうち DB ファイルに関しては事前にビルドされている必要があり，ビルドは前処理タスクにおいて実行される．このステップにおいて，ノード当たりの実行コア数やファイル複製数，タスクあたりの問題サイズ，1 タスクで用いるスレッド数等，様々なパラメータがシステム最適化の要因と

なる. また，本研究では，用いる相同性検索ツールである GHOSTZ の I/O を解析し，Direct I/O を用いて高速化を図った．詳細は後述する．

4.2 後処理 (ファイル集約タスク)

相同性検索で出力された断片的な出力ファイルを，クエリ毎に集約するタスクである．ファイルの集約タスクの依存関係は，**図 3** のような木構造になるため，並列に処理することが可能である．また出力ファイルは，2 つの入力ファイルサイズに依存して大きくなるため，必要に応じて圧縮を行う．

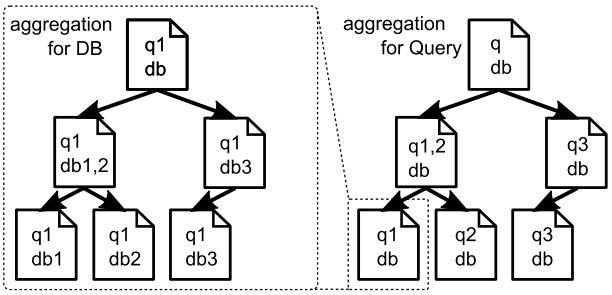

図 3 ファイル集約タスク

4.3 提案システムの特徴

提案システムの特徴として，Gfarm のスケジューリング機能によりファイルの出力先にはプロセスが実行されているノードが選択され，Pwrake のスケジューリングにより入力ファイルが存在するノードがタスク実行ノードとして選ばれるため，高速なローカル I/O を最大限生かすことができるという点が挙げられる．相同性検索に用いられるゲノムデータサイズは比較的大きいため，この特徴により通信の発生を最小限に留め，I/O 時間の削減が期待できる．

4.3.1 クエリと DB ファイルの両方を分割

提案システムは，増大するクエリと DB の両方を分割することで問題を解決する．分散並列実行のワークフローとして，Hadoop では MapReduce を用いるが，MapReduce の動作としては，1 つの入力ファイルを分割し，分割されたそれぞれの要素に対して Mapper が処理を実行，Mapper の出力を Reducer が集約といったプロセスをとる．両方を分割するには，あらかじめ前処理において分割しておき，Mapper において分割されたクエリと DB の両方のセグメントを取得するような処理が必要になる．そのため，Hadoop を用いた関連研究においては，DB ファイルに関しては分割せずクエリのみを分割して，それぞれのクエリに対して単一の DB との相同性検索を行うというアプローチが一般に取られる．しかし，2 章でも述べたようにクエリと DB ファイル両方の肥大化という問題に対する解決策としては有効ではない．

一方 Spark では，RDD と呼ばれるデータを抽象化した単位でそれらに対する処理の DAG を生成し，それを基にして，分散並列実行を実現する．それぞれの RDD は更に複数のパーティションで構成され，パーティション数が処理の並列度となる．そのため，理論的には DB とクエリの両方を分割させることが可能だが，実装コストが高いためか，関連研究の SparkBLAST では DB は分割せず，単一のものを用いている．

4.3.2 柔軟性

提案システムのもう一つの特徴として，相同性検索ツールをそのまま用いるという点が挙げられる．相同性検索ツールは，昨今様々なアルゴリズムを用いたものが開発されているため，MPI や Hadoop/Spark といったフレームワークを用いた実装では，用いる相同性検索アルゴリズムの変更に対する再実装のコストが比較的高い．それに対して，提案システムは既存の相同性検索ツールをそのまま用いて分散並列実行する設計となっているため，新たな相同性検索ツールが開発されたとしても，一般的なインターフェース (2 入力 1 出力) を満たしていれば，すぐにそのツールを用いた分散並列実行が可能である．
また，相同性検索ツールにおいて実行速度と正確性はトレードオフであり，様々なツールを用いて結果がどのように変化するか比較したいという需要に対しても，この設計は効果的であると考える．

5. 提案システムの最適化

5.1 ファイル分割タスクの高速化

本研究では，システムの前処理で用いるファイル分割プログラムの高速化を行った．ファイルの分割方法として，ファイルサイズによる分割が考えられるが，GHOSTZ の実行時間は，入力に用いる FASTA ファイルのサイズよりもそのシーケンス数に強く依存する．そのため，既存のファイル分割では，分割後の各ファイルシーケンス数が等しくなるようになっているが，シーケンシャルな方法でファイル分割を行っている．ここでシーケンス数とは，FASTA ファイルを構成する一まとまりの DNA 配列であるシーケンスの本数のことである．シーケンス数による分割では，一度全体のシーケンス数をカウントする必要があるが，この部分に関して mmap でファイルをメモリ空間にマップし OpenMP を用いてマルチスレッドでシーケンス数をカウントすることにより処理を高速化した．また，最初のシーケンス数のカウントの際にファイルのオフセットとシーケンス数の対応関係を保存しておき，それを基に正しい分割点を決定するようにすることでファイル分割処理に要する時間を削減している．一連のファイル分割の高速化の概要を**図 4** に示す．

また，高速化を施したプログラムとオリジナルの実装と

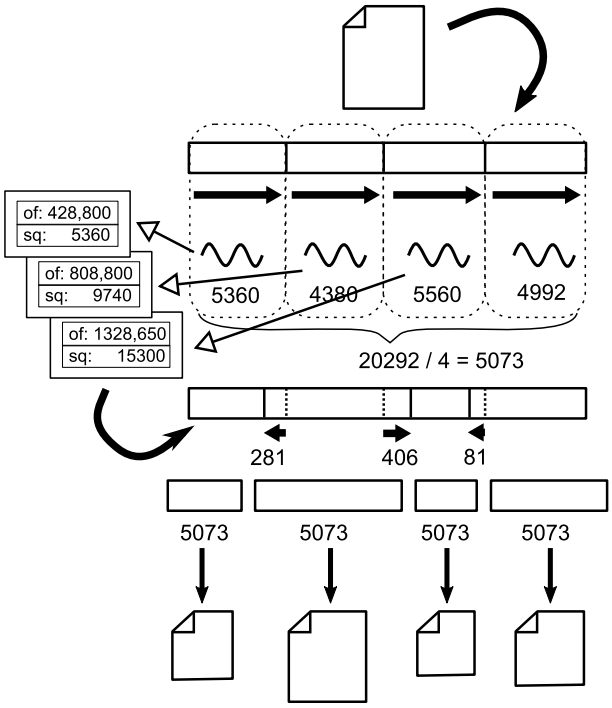

図 4 ファイル分割プログラムの概要

の比較を**図 5** に示す．

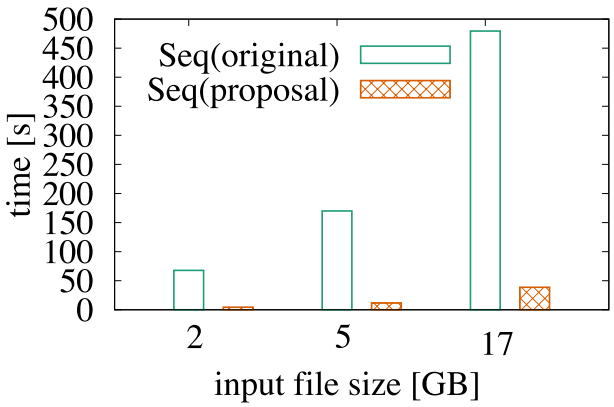

図 5 ファイル分割プログラムの高速化

結果より，オリジナルの実装よりも 10 倍以上高速であることがわかる．

5.2 GHOSTZ の高速化

提案システムは，ローカル I/O を活かす設計となっているため，ストレージデバイスに高速な SSD を用いることでファイル I/O のさらなる高速化が期待できる．NVMe SSD は近年注目されているフラッシュメモリベースのストレージであり，PCIe 接続により高バンド幅・低レイテンシを実現する．本研究では，提案システムのローカルストレージに NVMe SSD を使用するにあたって GHOSTZ の I/O を解析し，NVMe SSD の性能を発揮できるように GHOSTZ の I/O の高速化を検討した．I/O の解析にあたっては，ソースコードにおける I/O 部分を特定し，strace や HPC 向けの I/O モニタリングツールである darshan を用いた．解析の結果，クエリの読み込みに関して小さいサイズのシステムコールの発行回数が多く，NVMe SSD の高バンド幅を活かせていないことがわかった．その理由として，クエリの読み込みにおいては，C++の標準ライブラリの ifstream.getline を用いて 1 行ずつ読み込んでおり，かつ getline が read 命令に用いるブロックサイズが 8[KB] である点が挙げられる．一方，DB を構成する各ファイルの読み込みは，ファイルのバイトサイズ単位で読み込みを行っているため，比較的高いバンド幅を出せている．また，クエリの読み込みに関して，1 行読み込んだ後にファイルのオフセットを巻き戻す処理により，余分なシステムコールが呼ばれていたことが判明した．

そこで，このファイルオフセットの巻き戻し処理を無くし，また，NVMe SSD の高バンド幅を活かすためにバッファサイズを変更可能な一行読み込み関数を実装することにより，クエリの読み込みに関して高速化を施した．関数内で，次のシーケンス部のヘッダを検知し，次の関数呼び出し時にその場所から処理が再開するようにすることで，ファイルオフセットの巻き戻しによるオーバヘッドを無くしている．なお，提案システムにおける I/O の効率化のために，ファイルの read には Direct I/O を用いた．オリジナルの GHOSTZ と高速化を施したものとの比較を，**図 6** に示す．また，図 6 における，I/O 時間のみを抜き出したものを**図 7** に，全体の実行時間に対する I/O の割合を**図 8** に示す．入力ファイルサイズはクエリ・DB 共に 1[GB] であり，I/O 時間の取得には，Darshan を利用した．GHOSTZ の比較は**表 1** に示す環境で実施され，NVMe SSD 上のファイルに対して GHOSTZ が実行される．なお，比較においては CUDA を用いた GPU による GHOSTZ の高速版である GHOSTZ-GPU[21] を用いて，東工大のスーパーコンピュータである TSUBAME3.0 においても実験を行った．こちらも NVMe SSD 上のファイルに対して GHOSTZ-GPU が実行される．

結果より，GHOSTZ に関して全体の実行時間の減少割合は少ないが，read の時間が減少していることが見て取れる．また，GHOSTZ-GPU については，全体の実行時間に占める I/O の割合が約 8%であることがわかった．興味深いのは，TSUBAME における GHOSTZ-GPU に関して，改善後の性能向上が著しい点である．これは，TSUBAME の計算ノードの NVMe SSD の特性的にファイルオフセットの巻き戻し処理が全体の実行時間に多大な影響を及ぼしていると推測できる．

5.3 相同性検索実行タスクのワークフロー改善

5.3.1 最適なパラメータ

提案システムの相同性検索実行タスクにおいては，様々なパラメータに対して，全体の実行時間が最小になるよう

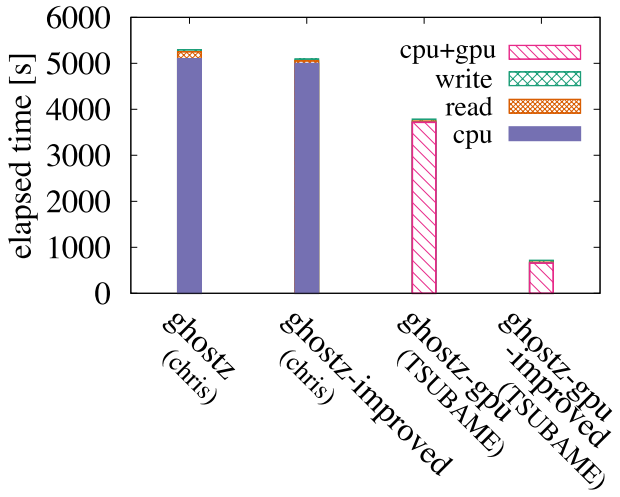

図 6 GHOSTZ の実行時間

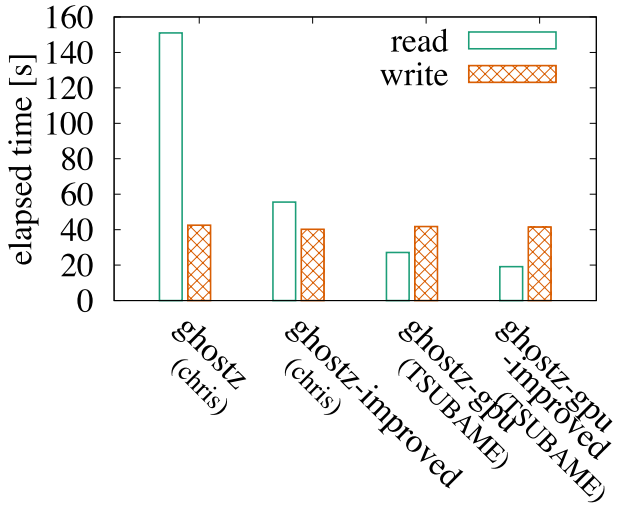

図 7 GHOSTZ の I/O 時間

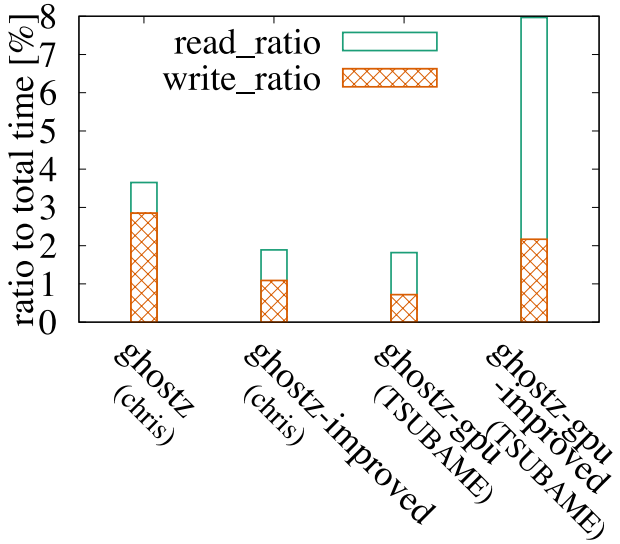

図 8 全体の実行時間に対する read/write 時間の割合

にパラメータを最適化することが重要である．パラメータとしては，ノードあたりのコア数・1 タスクあたりの入力ファイルサイズ (シーケンス数)・ファイル複製数などが考えられる．これらのパラメータを用いて実行時間をモデル化し，これを解析することによってパラメータ推定を行うのが理想であるが，実際にはリソースの競合や実行時環境が実行時間に与える影響は大きく，予想が困難であるため，実験的に求める必要がある．本研究においては，ノードあたりコア数に関して，実行環境における最適なパラメータを求めた．実験結果を**図 9** に示す．なお，実験は表 1 に示す計算ノードを 5 ノード用いて行った．実験結果より，実験環境においてはノードあたりのプロセス数が 8 のとき最も効率が良いことがわかった．1 ノード当たりの最適なコア数は，実行環境によって差異が生まれる可能性があるが，本システムでは使用アプリのワークロードに依存する 1 ノード内におけるリソースの取り合いが最も実行時間に影響を与えるため，この結果は 1 つの最適化指標として有用であると考える．

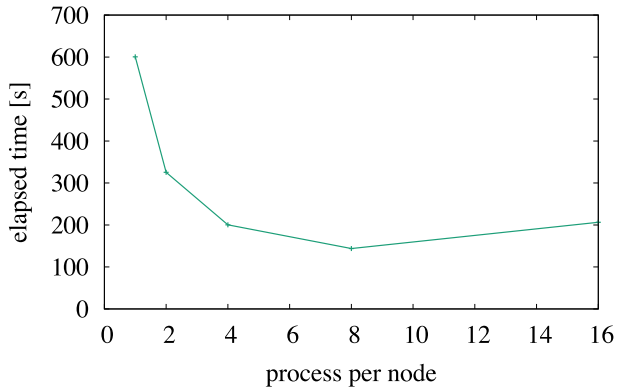

図 9 ノードあたりプロセス数による実行時間の減少

5.3.2 I/O の効率化

GHOSTZ の I/O を解析した結果，クエリの読み込みは 1 ファイルあたり 1 回であるが，DB ファイルの読み込みは複数回であることがわかった．これは，クエリに対するイテレーションの中で，DB のイテレーションが発生するためである．そのため，DB ファイルに関して，高速なローカル I/O を活かし，かつメモリにキャッシュすることで DB の読み込みをより高速にすることができる．さらに DB は全ノードにバランス良く分散配置されるため，相同性検索実行タスクにおいて，ある DB を入力とするタスクは常にその DB が存在するファイルサーバ上で実行される．DB ファイルのキャッシュを保持しつつ，DB ×クエリ数のタスクが実行される事を考えると，クエリに関してはキャッシュを用いる必要性が低い．そのため，章 5.2 の GHOSTZ の高速化で述べたように，クエリの読み込みに関しては，Direct I/O を用いてキャッシュをバイパスすることによって，システム全体としてのさらなる高速化が期待できる．また，結果ファイルの出力に関しては，基本的にローカル書き込みが行われるため高速であるが，オリジナルの実装では，メモリに出力ファイルのページをキャッ

シュする．しかし，出力ファイルはその後の処理において用いられることは無いため，これはオーバーヘッドとなり，更には出力ファイルサイズは一般に大きいため，キャッシュされている DB ファイルが退避されてしまう可能性がある．出力の Direct I/O による高速化の検討は今後の課題に含まれる．

6. 性能評価

本章では，提案システムを関連研究である SparkBLAST 及び HAMOND と比較した結果を示す．比較対象のプログラムは，それぞれ相同性検索ツールとして，BLAST・DIAMOND を用いているため，それぞれとの比較において提案システムで同じ相同性検索ツールを用いるようにした．SparkBLAST と HAMOND においては，ファイルシステムに HDFS を，クラスタマネージャに YARN を用いた．Hadoop のバージョンは 2.7.7，Spark のバージョンは 2.2.0，BLAST のバージョンは blastall2.2.26，DIAMOND のバージョンは 0.9.14 のものを用いた．実験環境を表 1 に示す．

表 1 実験環境

	chris
OS	CentOS 6.9
CPU	Intel Xeon E5-2665
memory	64GB
Storage	NVMe SSD (Ultrastar SN260)

また，Hadoop クラスタの基本設定は**表 2**, **表 3**, **表 4** のとおりである．また，Spark の基本設定を**表 5** に示す．なお，クラスタのチューニングは CloudEra の記事を参考に行った [22].

表 2 yarn-site.xml, core-site.xml

io.file.buffer.size	128[KB]
dfs.replication	3

表 3 mapred-site.xml

yarn.app.mapreduce.am.resource.cpu-vcores	2
yarn.app.mapreduce.am.resource.mb	2048
mapreduce.map.cpu.vcores	3
mapreduce.map.memory.mb	2048
mapreduce.task.io.sort.mb	2048

表 4 yarn-site.xml

yarn.nodemanager.resource.cpu-vcores	12
yarn.nodemanager.resource.memory-mb	10240
yarn.scheduler.minimum-allocation-vcores	1
yarn.scheduler.maximum-allocation-vcores	12
yarn.scheduler.minimum-allocation-mb	1024
yarn.scheduler.maximum-allocation-mb	10240

SparkBLAST との比較を**図 10** に示す．なお，クエリと DB には 100[MB] のファイルを用いた．

表 5 spark-defaults.xml

spark.executor.instances	5
spark.driver.cores	5
spark.executor.cores	5
spark.driver.memory	20g
spark.executor.memory	20g
spark.serializer	KryoSerializer

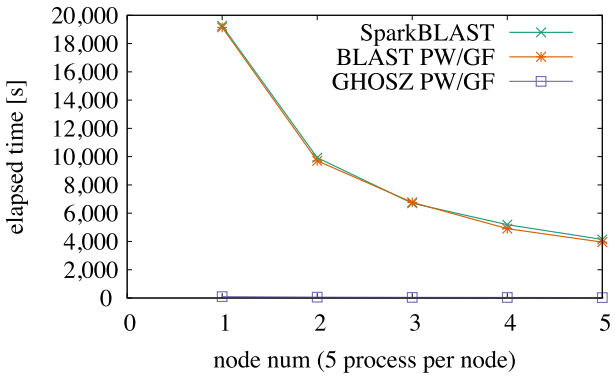

図 10 SparkBLAST との比較

SparkBLAST との比較では，クエリの分割数 (50)・DB の分割数 (1)，Spark の 1 ノードあたりの Executor 数 (5) という条件にあわせて提案システムの評価を行っている．結果より，提案システムは全てのノード数において SparkBLAST 以下の実行時間であることがわかった．なお，評価に際して，SparkBLAST では結果ファイルの集約に関して hdfs コマンドで別途集約する必要があるため，提案手法におけるファイル集約タスクは除外している．SparkBLAST との性能が同等なものになった理由としては，Spark において Map 処理のみが存在するためシャッフルが発生せず，余分なオーバーヘッドが少ないからであると考えられる．また，SparkBLAST において BLAST の処理はシェルコマンドとして各ノード上で実行するような実装であり，これは提案手法と同様の手法であるため，BLAST の実行において両者における差異はあまりない．今後の課題として，ファイル数や用いるデータセットのサイズを大きくした場合に関して結果がどのように変化するか確認したい．

次に，HAMOND との比較を**図 11** に示す．なお，クエリと DB には 100[MB] のものを用いた．

実験では，1 つの DIAMOND 実行タスクが処理の際に 3 スレッドを用いるようにしている．また，HAMOND の実装により入力ファイルの分割サイズは 2[MB] となっているため，Mapper の数が 50 になっている．提案手法においては，クエリの分割数をこれらの値に設定し，またノード当たり実行コア数を 1 ノード当たりで実行される Mapper 数である 1 に合わせた．HAMOND は Hadoop クラスタのさらなるチューニングや適切な Mapper 数の割当等による性能向上の余地があるが，結果は提案手法が HAMOND より

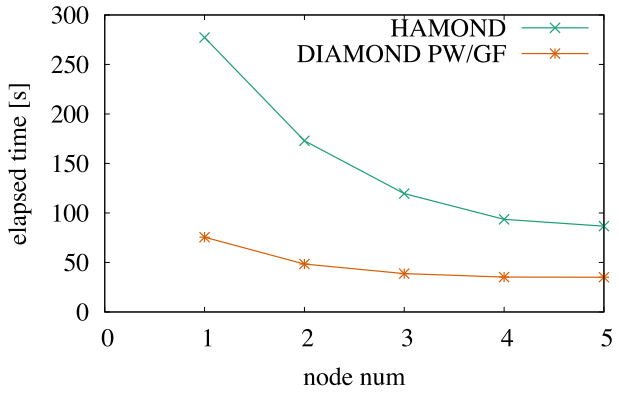

図 11 HAMOND との比較

も高速に相同性検索を実行するということを示している．

7. まとめと今後の課題

本研究では，Gfarm と Pwrake を用いた相同性検索システムについて，前処理タスクにおけるファイル分割プログラムの高速化と GHOSTZ の高速化を行った．さらに，相同性検索タスクにおける最適なパラメータを実験的に求め，全体としてシステムの高速化を達成した．また，クエリごとにファイルを集約するタスクを加えることによりユーザビリティを向上させた．

関連研究との比較では，相同性検索ツールである BLAST と DIAMOND の Spark/Hadoop による拡張版と比べて同等かそれ以上の性能を示し，提案システムのスケーラビリティ及び高速性を確認した．

今後の課題として，GHOSTZ の出力の高速化の検討や，実行時間のモデル化による最適なパラメータ推定等が挙げられる．また，大規模実行環境における実験により，本システムのスケーラビリティの調査やさらなる課題発見も行いたい．

謝辞 本研究の一部は，JSPS 科研費 17H01748，国立研究開発法人新エネルギー・産業技術総合開発機構 (NEDO)，JST CREST JPMJCR1414 および富士通研究所との共同研究による

参考文献

[1] イルミナ社：ヒトの健康における細菌およびメタゲノム，入手先 ⟨https://jp.illumina.com/content/dam/illumina-marketing/apac/japan/documents/pdf/publicaton_metagenome_forhumanhealthj.pdf⟩ (2015?)
[2] ヒト腸内メタゲノム解析が広げる医療展開，入手先 ⟨https://www.jstage.jst.go.jp/article/kagakutoseibutsu51/12/51_802/_pdf/char/ja⟩ (2013)
[3] Gaspar Taroncher-Oldenburg, Susan Jones, Martin Blaser, Richard Bonneau, et al, Translating microbiome futures, Nature Biotechnology volume 36, pages 10371042 (2018)
[4] Metagenomics Market Size, Share, Industry Trends Report, 2018-2025, GVR-2-68038-452-9 (2018)
[5] Kanehisa M, Goto S, KEGG: Kyoto Encyclopedia of Genes and Genomes, Nucleic Acids Research. 2000;28(1):2730
[6] Kanehisa M, Goto S, Sato Y, Furumichi M, Tanabe M, KEGG for integration and interpretation of large-scale molecular data sets, Nucleic Acids Research. 2012;40(D1):D109D114
[7] Tatusov R, Fedorova N, Jackson J, Jacobs A, Kiryutin B, Koonin E, et al. The COG database: an updated version includes eukaryotes, BMC Bioinformatics. 2003;4(1):41. pmid:12969510
[8] Stephen F.Altschul, Warren Gish, Webb Miller, Eugene W.Myers , David J.Lipman1Basic：local alignment search tool, Biol. 1990,vol.215(pg. 403-410)
[9] 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.
[10] Shuji Suzuki, Masanori Kakuta, Takashi Ishida and Yutaka Akiyama：Faster sequence homology searches by clustering subsequences, Bioinformatics (2014) doi: 10.1093/bioinformatics/btu780
[11] Benjamin Buchfink, Chao Xie & Daniel H. Huson：Fast and Sensitive Protein Alignment using DIAMOND, Nature Methods, 12, 5960 (2015) doi:10.1038/nmeth.3176.
[12] Machida Kenta, Osamu Tatebe: Pwrake/Gfarm による分散並列相同性検索システムの提案, 第 162 回ハイパフォーマンスコンピューティング研究発表会 (2017)
[13] AE Darling, L Carey, WC Feng, The design, implementation, and evaluation of mpiBLAST, ClusterWorld Conference&Expo 2003
[14] Yutaka Akiyama, GHOST-MP: an ultra-fast and high-sensitive homology search tool for metagenome analysis, ADAC Workshop, 2016.
[15] A O'Driscoll, V Belogrudov, J Carroll et. al., HBLAST: Parallelised sequence similarity A Hadoop MapReducable basic local alignment search tool, Journal of Biomedical Informatics Volume 54, April 2015, Pages 58-64
[16] Marcelo Rodrigo de Castro, Catherine dos Santos Tostes, et. al., SparkBLAST: scalable BLAST processing using in-memory operations, BMC BioinformaticsBMC series open, inclusive and trusted201718:318
[17] Jia Yua, Jochen Blomb, Alexander Sczyrbac, Alexander Goesmann, Rapid protein alignment in the cloud: HAMOND combines fast DIAMOND alignments with Hadoop parallelism, Journal of Biotechnology Volume 257, 10 September 2017, Pages 58-60
[18] 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)
[19] 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
[20] 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
[21] Suzuki Shuji, Kakuta Masanori, Ishida Takashi and Akiyama Yutaka: GPU-Acceleration of Sequence Homology Searches with Database Subsequence Clustering, PLoS ONE 11(8): e0157338.(2016)

https://doi.org/10.1371/journal.pone.0157338
[22] Tuning YARN,
入手先 ⟨https://www.cloudera.com/documentation/ enterprise/latest/topics/cdh_ig_yarn_tuning.html⟩