Vol.2018-BIO-53 No.8
2018/3/10

クラウド上の分散GPU環境における タンパク質間相互作用予測計算フレームワークの開発

山本 悠生[1,a)]　大上 雅史[1,b)]　秋山 泰[1,c)]

概要：パブリッククラウドは必要な時に必要な分だけ計算資源を借りることができる点や，容易にスケールアウトすることができる点で既存の環境に比べて優れている．また近年クラウド上で GPU を使用できるサービスを各クラウド事業者が展開している．MEGADOCK はタンパク質間相互作用を高速に予測するソフトウェアであるが，多数の相互作用の予測には大規模な計算資源が必要である．本研究では MEGADOCK の GPU 版をパブリッククラウド上の分散 GPU 環境で並列に効率よく動作させ，計算機に精通していない潜在的な MEGADOCK の利用者にも使いやすいように，計算資源とソフトウェアを同時に提供する環境を開発した．MEGADOCK の効率的な並列計算を行うことができる条件について実験を行った結果次のことが分かった．a) 時間当たりの料金が同額の GPU 有りの VM のクラスタならば，大規模な VM を少数使用する場合よりも小規模な VM を多数使用する方が高速であること．b)GPU 有りの VM と無しの VM で時間当たりの料金が同程度のクラスタの場合，GPU を使用する方が 1.35 倍程度高速であること．

キーワード：パブリッククラウド，MEGADOCK，分散処理，GPU，タンパク質間相互作用，MapReduce，Apache Spark

Yuki Yamamoto[1,a)]　Masahito Ohue[1,b)]　Yutaka Akiyama[1,c)]

1. 序論

1.1 研究背景

1.1.1 パブリッククラウド

近年，計算資源を使用するための方法としてクラウド事業者のマシンの一部を仮想マシン（Virtual Machine; VM）の形で借りるパブリッククラウドが広まってきている．パブリッククラウドは従来の環境に比べて，使う時に使う分だけ借りられるため，利用する計算資源の量に波があるような場合に経済的である．さらにパブリッククラウドは VM の形で提供されるため，計算の規模に応じて台数を増やすことを容易に行うことができる．さらに物理的なマシンの管理はクラウド事業者が行うため，計算機の構成について詳しくない場合でも比較的運用が容易になる利点もある．

このようなパブリッククラウドを研究でも用いる流れが存在しており，本研究と同様のバイオインフォマティクスの分野でも利用され始めている [1–6]．

1.1.2 GPU

GPU（Graphics Processing Unit）は元々画像処理の用途のために作成された演算装置であるが，そのために行列演算を高速に行う機能が備わっている．近年の GPU では計算の流れを自由にプログラムできる機能が備わっているものがあり，科学技術計算の中でも行列演算などに活用されるようになっており，そのための GPU シリーズも発売されている [7]．パブリッククラウドのなかでも近年このような GPU 付きの VM を利用可能なサービスが現れている．

1.1.3 パブリッククラウドとバイオインフォマティクス

バイオインフォマティクス分野でも近年大規模で高速な情報処理環境が求められるようになってきており，その方法としてパブリッククラウド環境を利用するという研究が行われている．例えば，代表的な配列相同性検索ツールで

[1] 東京工業大学 情報理工学院 情報工学系
Department of Computer Science, School of Computing, Tokyo Institute of Technology
[a)] y_yamamoto@bi.c.titech.ac.jp
[b)] ohue@c.titech.ac.jp
[c)] akiyama@c.titech.ac.jp

ある BLAST をパブリッククラウド上で実行する研究 [1] がある．

1.1.4 MEGADOCK

MEGADOCK はタンパク質間相互作用を予測するソフトウェアである．このソフトウェアでは内部的に高速フーリエ変換を利用した畳み込み計算を行なっており，GPU を用いて高速に計算を行うことができる．同じ目的で使用されるソフトウェアに ZDOCK [8] が存在するが，MEGADOCK は ZDOCK と同程度の精度でおよそ 7.5 倍高速に計算を行うことができるとされている [9]．

1.1.5 タンパク質間相互作用

タンパク質間相互作用はしばしば Protein-Protein Interaction を略して PPI とも呼ばれる．PPI とは生物の中にあるタンパク質が相互作用することである．

PPI の予測が必要な理由としては，ヒトの体内には 10 万種ほどタンパク質が存在しており，それらの相互作用を全て実験で発見するためには時間的にも経済的にも莫大なコストがかかるため，計算機を用いてあらかじめ相互作用するかどうか予測することがあげられる．

1.2 並列分散処理におけるフレームワーク

分散処理を行うための技術は古典的な MPI（Message Passing Interface）を始めとしていくつかの技術が存在しているが，ここでは主に近年バイオインフォマティクスの分野でも用いられている MapReduce [10] や，その実装である Apache Hadoop [11]（以下 Hadoop とする），Hadoop の一部を置き換えて改良するソフトウェアである Apache Spark [12]（以下 Spark とする）について説明する．

1.2.1 MapReduce

MapReduce [10] とは，Google によって提唱されたプログラミングモデルである．MapReduce では並列計算を単純化して，データをノードに配って計算する Map ステップとそれらの結果を集約する Reduce ステップのみで表現する．これによって大きなデータを容易に取り扱うことが出来るようになり，障害のあったノードが存在する場合でもリカバリーが容易になる．

1.2.2 Hadoop

Hadoop は MapReduce というプログラミングモデルを実装したソフトウェアで，The Apache Software Foundation [13] によって管理されているオープンソースソフトウェアである．現行のバージョンである Hadoop2 や 3 は以下の 4 つのモジュールから成り立っている．

(1) Hadoop Common: 共通ライブラリ群
(2) Hadoop Distributed File System (HDFS): Hadoop の動作が効率よくなるように設計された分散ファイルシステム
(3) Hadoop YARN: 計算ノードに関する管理を行うクラスタマネージャー
(4) Hadoop MapReduce: MapReduce 計算の実装

これらのモジュールを組み合わせることによって Hadoop では MapReduce による計算を簡単に行うことができる．

1.2.3 HDFS (Hadoop Distributed File System)

HDFS とは Hadoop の中のモジュールであり，分散ファイルシステムである．HDFS ではデフォルトで 3 つの異なるノードにファイルが保存され，いくつかのノードがダウンした場合でもデータが失われない仕組みになっている．

1.2.4 Spark

Spark とは，Hadoop MapReduce にかわる分散処理ソフトウェアとして開発されたソフトウェアである．Spark ではデータをメモリ上に保持することで，Hadoop MapReduce で必要であったディスク IO による処理速度の低下を防いでいる．

1.3 研究目的

PPI 予測ソフトウェアとしては比較的高速に動作する MEGADOCK であっても，多数の PPI を現実的に予測を行うためには大規模な計算資源が必要である．そのため十分な計算資源を持っていない研究者や計算機に精通していない研究者の利用にはハードルが高い．この問題を解決するために，本研究ではパブリッククラウド上で MEGADOCK での計算を GPU を使用して高効率で行なうことができ，そのようなシステムをユーザーが詳細な実装を意識することなく利用できるフレームワークの構築を目的とする．

2. 先行研究におけるパブリッククラウド

2.1 Lu らによる Microsoft Azure 上での BLAST

Lu らの研究 [1] では，パブリッククラウドである Microsoft Azure 上で BLAST を実行する AzureBlast を実装し，仮想マシンの数やサイズの違うインスタンスの様々な条件で性能の違いを調査している．その中では並列化性能は 1 つのタスクに 100 以上のクエリが含まれるようになると殆どスループットは変わらないことが示されている．また，高性能なインスタンスを用いると高速に計算できることは当然のことながら，高性能なインスタンスを用いたことによる利用料の上昇よりも計算速度の向上の方が大きく，同じだけ計算を行う場合では高性能インスタンスを用いた方が経済的であることも示されている．

2.2 Gunarathne らによる Amazon Web Service と Microsoft Azure の比較

Gunarathne らによる研究 [2] では，ゲノム断片のアセンブリを行なうソフトウェアである Cap3 を Amazon Web Service と Microsoft Azure を使用し，Hadoop と DryadLINQ を用いて動作させて比較を行っている．Cap3 はメモリよりも計算が計算時間に影響を与える支配的な要因となるソフトウェアである．この研究では Amazon Web Service を

使用した場合でも Microsoft Azure を使用した場合でも，また Hadoop を使用した場合でも DryadLINQ を使用した場合でも，並列化効率が 0.8 より大きいことが示されている．またローカルのクラウド環境を用いて HDFS と共有ディスク環境を比較したところ，共有ディスクは並列化においてオーバーヘッドとなることを示している．

2.3 Ekanayake らによる Hadoop·DryadLINQ·MPI の比較

Ekanayake らの研究 [3] では，遺伝子の類似度を調べるためにローカルのクラウド環境を用いて，Hadoop・DryadLINQ・MPI による並列化について比較を行っている．この研究によると，これらの 3 つの中では Hadoop による並列化が最も高速に動作するものの，大差ではないことが示されている．また自然なデータには殆ど無いことではあるものの長さ順にソートされているような偏ったデータである場合，DryadLINQ よりも Hadoop の方が性能が良いことが示されている．これは静的にタスクの割り当てを行う DryadLINQ に対して Hadoop は動的に行うことからこのような差が生まれているということが示されている．

2.4 Mrozek らによる Microsoft Azure 上での類似タンパク質の検索

Mrozek らの 2014 年の研究 [4] では，タンパク質の立体構造を用いた類似タンパク質の検索を行うソフトウェアを Microsoft Azure 上で動作させて，その性能について試験を行っている．この研究によると，計算に用いる VM の数を増やすとほとんど線形に計算性能が向上していることが示されている．検索のためのアルゴリズム 2 種類について比較を行っているが，ワーカーが 18 台のとき悪い方でも並列化効率が 0.84 あることが示されている．

2.5 Mrozek らによる Microsoft Azure 上でのタンパク質構造予測

Mrozek らの 2015 年の研究 [5] では，タンパク質の構造予測に Microsoft Azure を使用し Cloud4PSP というシステムを開発，性能の評価を行っている．この研究によるとワーカーの数が 4 まではほぼワーカーが増えた分だけ性能が向上しているが，8 以上になると少しずつ線形な向上とはならなくなっていくことが示されている．研究中で実験されている最大の数である 18 台のワーカーでは 14.52 倍の高速化率が見られて，並列化効率は 0.81 程度であることが示されている．さらにこの研究では「スケールアップとスケールアウトではどちらが良いか」という問いに答えている．ソフトウェアなどの様々な条件によるとしながらも，Cloud4PSP を実行するに当たっては僅かながら同じ CPU コアの数を使用するならスケールアウトする方が高速であることを示している．またスケールアウトはスケールアップよりも実装が容易であることが述べられている．

2.6 Moghadam らによる Amazon Web Service 上での QSAR モデリング

Moghadam らの研究 [6] では，Amazon Web Service (AWS) 上での QSAR モデリングをローカルのスーパーコンピュータと比較して，性能やパブリッククラウドを使用することの利点について述べている．この研究によると，スーパーコンピュータなどが利用できるような環境であればスーパーコンピュータを使用した方が安いこと，そうでなければパブリッククラウドは魅力的な選択肢になること，スーパーコンピュータが利用できる場合でもキューで待たされたくない場合には良い選択肢となり得ることが述べられている．さらにパブリッククラウドに向いている計算について，単純に並列化できるような計算が挙げられている．例えばクロスバリデーションやグリッドサーチによるパラメータの探索などの場合，一般的にはそれらの個々の実行は独立なのでパブリッククラウドに向いていると述べられている．

3. フレームワークの設計と実装

作成したフレームワークの全体的な概要図を図 1 に示す．

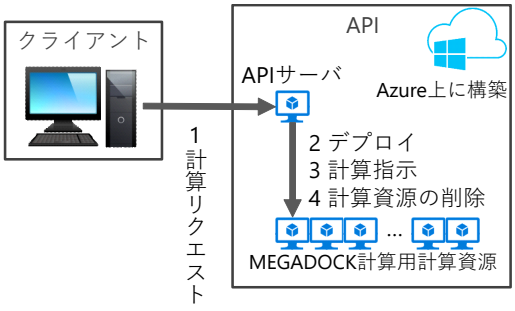

図 1 フレームワークの全体的な概要．「1 計算リクエスト」と「3 計算指示」については図 3 と第 3.2 節に，「2 デプロイ」については第 3.1.1 節に，「4 削除」については第 3.4 節にそれぞれ詳しく記述する．

3.1 Microsoft Azure へのデプロイの概要

デプロイとは VM などの計算資源を利用可能な状態にすることである．本研究における実装について説明する前に，一般的な Microsoft Azure へのデプロイと，デプロイした VM の削除の流れについて説明する．Microsoft Azure へは Azure CLI を使用してリクエストを送信する [14] ことでデプロイを行うことができる．Azure CLI によるデプロイ方法にはクラシック・ARM (Azure Resource Manager)・ASM (Azure Service Management) の 3 種類の方法が存在している．ここでは本研究で使用している ARM によるデプロイについて説明する．

3.1.1 デプロイ

この節の内容は図 1 中で「2 デプロイ」と記載されている部分の詳細である．ARM でのデプロイは Azure にデプロイしたい VM やそれに関連する計算資源の構成を記述したテンプレートを使用することで行うことができる．テンプレートは json 形式で記述されている．デプロイの際にリソースグループを指定しこのリソースグループに属する形でテンプレート中の計算資源はデプロイされる．ただしディスクはリソースグループによって管理されない．デプロイの概要について図 2 に示す．

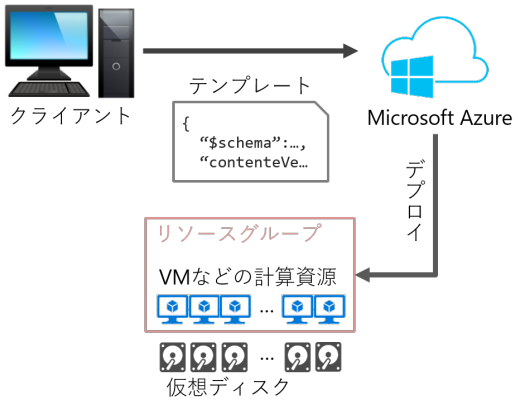

図 2 この図では ARM によるデプロイの概要について示している．デプロイにはテンプレートが使用され，ディスク以外の計算資源はリソースグループによって管理される．

3.1.2 計算資源の削除

デプロイした計算資源を削除するには，デプロイ時に作成したリソースグループを使用することができる．リソースグループを削除すると，リソースグループに属している計算資源は全て自動的に削除されるため，残りのディスクを個別に削除することで全てのデプロイされた計算資源を削除することができる．

3.2 実装の概要

クライアントからのリクエストに応じて Microsoft Azure 上で VM をデプロイして，その上でクライアントからアップロードされた PDB ファイルを用いて計算を行なう．このとき VM を複数デプロイするが，複数の VM を使用して計算を行ったときに可能な限り性能が低下しないように考慮する．

クライアントからの計算リクエストは MEGADOCK による計算とそれに必要なアップロードなどの処理を纏めたジョブという単位で管理され，以下のような手順で実行される．この流れについては図 3 にまとめる．図 3 の内容は図 1 中で「1 計算リクエスト」と記載されている部分の詳細である．

(1) ジョブを作成する
(2) PDB ファイルをアップロードする
(3) 計算の実行を指示する
(4) 結果を取得する

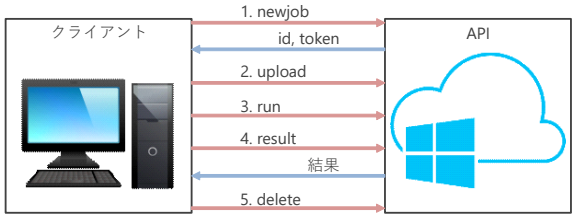

図 3 クライアントからの操作に着目したジョブの実行の流れを示している．赤の矢印はクライアントからのリクエスト，青の矢印は実装したフレームワークからのレスポンスである．

この手順とは別に，ジョブを削除する操作がある．

3.2.1 ジョブの作成

まず最初に計算を行うためにジョブを作成する．クライアントは 1 回 GET リクエストを行うことでこれを実行でき，この時 jobid と token を得る．これらは再発行できず，以降の操作ではこれらの組が必ず必要となる．

3.2.2 PDB ファイルのアップロード

相互作用の予測を行いたい PDB ファイルを zip または tar.gz 形式でアーカイブしてアップロードを行う．このときリクエストの形式として，multipart/form-data 形式・Base64 でエンコードして json でアップロードする形式・MessagePack 形式を使用でき，どの形式を使用するかはクライアントが選択できる．アップロードされたファイルはジョブごとに割り当てられたディレクトリに保存される．またこのアップロードの過程でどの組み合わせで相互作用の予測を行うか記述したファイルもアップロードする．そのファイルの形式が間違っていた場合はエラーとする．計算を行うまで組み合わせファイルが提供されなかった場合はアップロードされた PDB ファイルに対して全対全での予測を行う．

3.2.3 計算の実行を指示

PDB ファイルのアップロードが完了したら次は計算の実行の指示を行う．ここで，理論上は上限が無視できるほどの計算資源を用意できるパブリッククラウドでも，1 ユーザーに割り当てられる資源の量は限られており，同時に計算できないこともあるため，ただちに計算を開始するのではなく，一旦計算が可能が判断している．すぐに計算が開始できない場合，そのジョブは実行待機状態となって実行待ちのキューに入る．計算が開始されたら，まず VM をデプロイし，計算を行なう．クライアントが終了通知 URL を指定していた場合は，その URL に終了を通知する．使用した VM はすぐには削除せずに，削除可能としてマークする．このようにする理由については後述する．

3.2.4 結果の取得

計算が終了したら，結果をダウンロードする．このとき

も圧縮されたファイルを直接ダウンロードする，Base64 でエンコードした状態で json の中に入った形式でダウンロードする，MessagePack 形式でダウンロードすることを選択できる．いずれの場合でも，予期しないエラーが起こっていた場合はエラーが起こっていたことを知らせる結果を返す．

3.2.5 ジョブの削除

ジョブの削除は図 3 に示されているどの段階でも行うことができる．ジョブの削除を実行するとアップロードされたファイルは削除され，データベースのレコードは削除される．

3.3 並列化手法

MEGADOCK には MPI によって並列化を行うバージョンが存在している．しかしながら本研究では，個々の計算は完全に独立しており計算中に頻繁なデータのやりとりが必要ではないことから，VM の意図しない停止や計算の失敗などのエラーが起こった場合でも回復が容易な Apache Spark を用いた並列化手法が，必ずしもデプロイが安定して行われるとは言えないパブリッククラウド環境において有効であると考え，Apache Spark を用いた実装を行った．Apache Spark を用いた実装の概要について図 4 にまとめた．

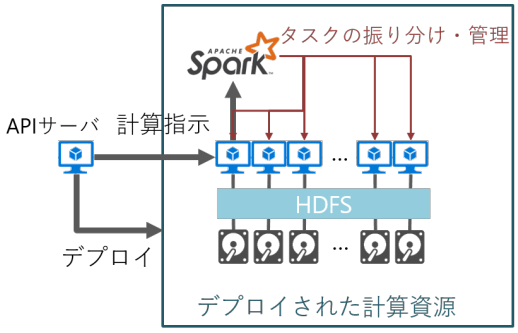

図 4 この図では Apache Spark を用いた並列計算の実装の概要を示している．API サーバが計算資源をデプロイした後，API サーバはデプロイされた VM に Spark クラスタと HDFS の起動を指示してから Spark クラスタに計算ジョブを投入する．Apache Spark はデプロイされた VM 上で動作し，分割されたジョブを動的にクラスタ内の計算機に割り振って計算を行う．計算時にエラーが生じた時は Apache Spark が再計算を行う．

3.4 VM の削除について

第 3.2 節でも示したように，VM の削除は直ちには行わない．削除可能としてマークされた VM は削除キューに入って実際に削除されるのを待つことになる．

これはジョブ中で削除を行ってしまうと，図 5 で示すように既に結果は出ているにもかかわらずクライアントから見た計算時間が数分伸びてしまうからである．

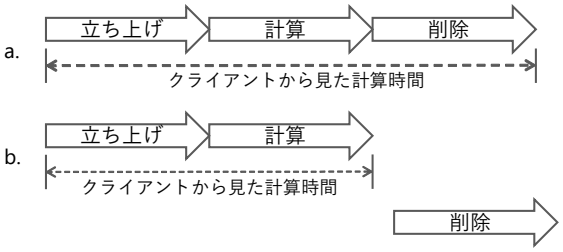

図 5 VM の削除を直ちに行わずに別のキューで行うことによりクライアントから見たみかけの実行時間が短くなることを示した図．a. は削除を別に行わない場合，b. は削除だけ別に行う場合を示している．

4. 実験

4.1 実験条件

`time` コマンドを用いて 3 回計測を行い中央値を求めた．測定に失敗した場合は回数には含めずやり直した．

Microsoft Azure では複数の種類の VM が用意されており [14]，今回使用した VM の概要を表 1 にまとめた．

4.2 データセット

データセットは protein-protein docking benchmark 1.0 [15] の中の Unbound PDB Set を用いて，全レセプター対全リガンドでドッキングを行った．ドッキングペア数はレセプター 59 個 × リガンド 59 個の計 3,481 組存在した．

4.3 VM 数と計算性能の関係を検証するための実験

F シリーズと NC シリーズの VM はそれぞれ使用料金に比例して理論的な計算性能が向上する仕様となっている．そこでそれらの VM のうちどれを使用するのが実際には最も高速または経済的であるか調べた．

4.3.1 実験条件

NC シリーズと F シリーズのそれぞれについて以下の 2 つの条件で実験を行う．

(1) それぞれの VM を 1 台使用してその速度を比較する．
(2) N シリーズでは CPU コア数が 720 になるように，F シリーズでは同様に CPU コア数が 320 になるようにして比較を行う．このときどちらのシリーズの場合も全ての VM 全体での価格は同一であり，理論上はほぼ同じ程度の時間となる．

4.3.2 結果

得られた結果を図 6, 図 7, 図 8, 図 9 にまとめた．これらの図を見ると，CPU のみを使用した場合と比べて GPU も使用した場合は，1 台あたりのスペックを高くすると計算時間が伸びる割合が大きいことが分かる．

4.3.3 考察

CPU 版では F8 と F16 では CPU のコア数が増えてもほぼ同等の性能が出ていることが分かる．F64 では少々低い性能となっているのは，MEGADOCK の実装が 64 個のコ

表 1 本研究で実験に使用した VM のリスト．F シリーズでは CPU に Intel Xeon E5-2673 v3 (Haswell) 2.4 Ghz が使用されている．NC シリーズに搭載されている CPU の詳細は不明である．NC や F の後の数字はその VM に搭載されている CPU コア数となっている．Standard_NC6 において Tesla K80 の搭載枚数が 1/2 枚となっているが，これは Tesla K80 には 1 枚のボードに GPU が 2 個搭載されており，Standard_NC6 インスタンスではそのうち 1 つのみが使用可能であるためである．すなわち Standard_NC12 では GPU が 2 個，Standard_NC24 では GPU が 4 個利用可能である．

VM の種類	CPU コア数	GPU	RAM（GiB）	料金（円/台/h）
Standard_F4s_v2	4	なし	8	18.93
Standard_F8s_v2	8	なし	16	37.86
Standard_F16s_v2	16	なし	32	75.83
Standard_F64s_v2	64	なし	128	303.08
Standard_NC6	6	Tesla K80 1/2 枚	56	100.80
Standard_NC12	12	Tesla K80 1 枚	112	201.60
Standard_NC24	24	Tesla K80 2 枚	224	403.20

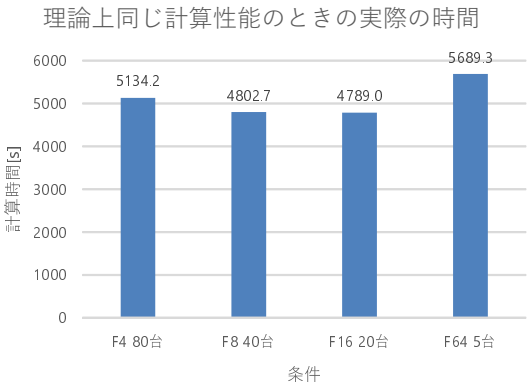

図 6 F シリーズ（GPU 無）の VM による計算時間測定結果．総 CPU コア数が 320 になるように作成したクラスタでの計算時間．

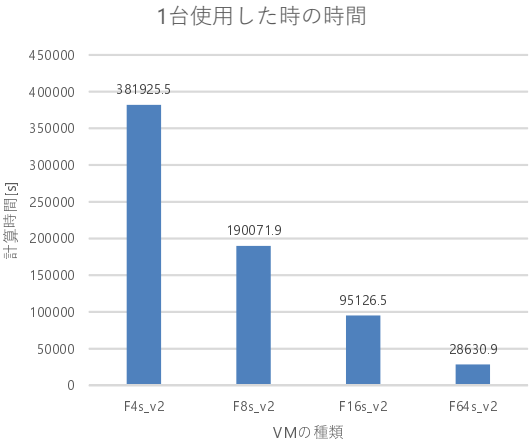

図 7 F シリーズ（GPU 無のみ）の VM を 1 台のみ使用した場合の計算時間．

アに対して十分並列に計算できるようになっていないからであると考えられる．また F4 で低い性能となっているのは，VM の数が多くなったことで計算時間に対して通信に要する時間が増したからであると考えられる．

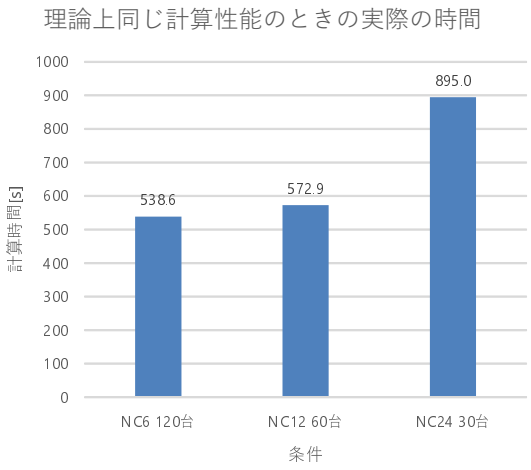

図 8 NC シリーズ（GPU 有）の VM による計算時間測定結果．総 CPU コア数が 720 になるように作成したクラスタでの計算時間．

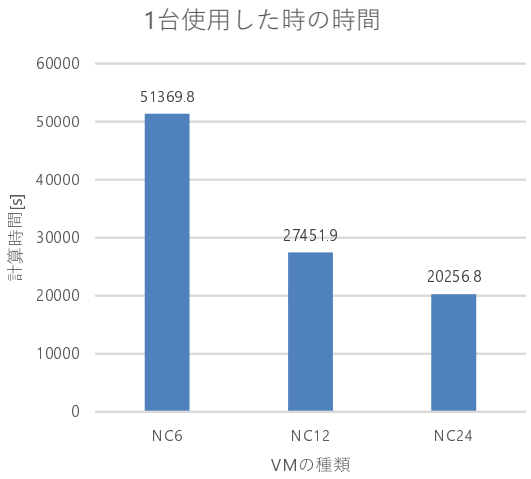

図 9 NC シリーズ（GPU 有）の VM を 1 台のみ使用した場合の計算時間．

GPU 版では VM を 1 台使用した場合の高速化率が，表 2 に示されるように VM が備えている計算能力から期待される値よりも低くなっている．さらに複数の VM を使用した実験での計算時間が，1 台の計算時間から線形に計算能力が増加した場合と大きく変わらないことから，GPU を複数使用することによる非効率化が台数の増加よりも性能の低下要因として支配的であることが分かる．

4.4 GPU の有効性を検証するための実験

CPU のみを用いた場合と GPU を用いた場合の性能を比較し，GPU を使用することがどの程度有効か調べる．

4.4.1 実験条件

CPU のみの VM と時間あたりの料金が同程度となる GPU 付き VM を用いて比較を行う．CPU 専用で最速であった構成は F16 の 20 台であり，この構成の 1 時間あたりの料金は $75.83 \times 20 = 1516.6$ 円である．これに最も近い GPU 使用時の条件は NC6 を 15 台使用する時であり，この時の料金は 1 時間あたり $100.8 \times 15 = 1512$ 円である．今回の実験ではこの条件での比較を行う．

4.4.2 結果

結果を図 10 にまとめた．この図を見ると GPU を使用した方が 1.35 倍程度高速であり，CPU のみの場合の 75%程度の料金で計算を行うことができることが分かる．

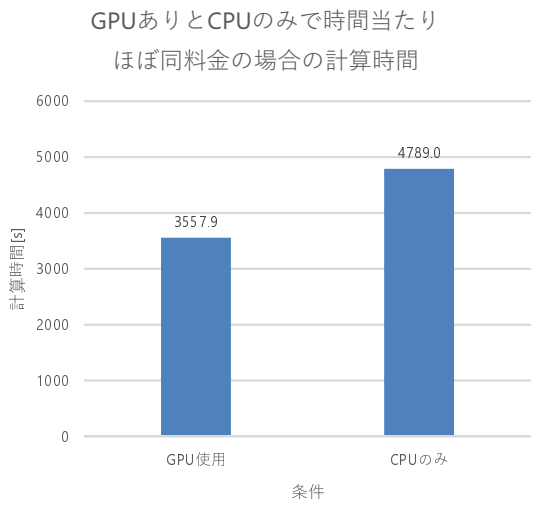

図 10 CPU と GPU で時間当たりの利用料金がほぼ等しい時の計算時間の差を示す．GPU を使用した方が 1.35 倍程度高速であることが示されており，これはほぼそのまま計算にかかる料金の差を示している．

4.4.3 考察

GPU を使用すると CPU を使用する場合よりも高速であるだけでなく，経済的に計算を行うことができることが示された．したがって MEGADOCK の計算においては常に GPU を使用した計算を行う方が良いと考えられる．

4.5 HDFS の有効性と並列化効率を調べる実験

このタスクで HDFS を使用することが有効か，また並列化効率はどの程度になるか検証するため，HDFS を使用しない実装と使用する実装を行い，計算にかかる時間を比較した．

4.5.1 実験条件

Standard_NC6 を 1, 5, 10, 15, 20, 30, 40, 50, 60, 80, 100, 120, 150 台使用して比較を行い，HDFS を使用する場合と使用しない場合についてそれぞれ 1 台の場合に対する高速化率と並列化効率を調べた．

4.5.2 結果

高速化率を図 11 に，並列化効率を図 12 にまとめた．HDFS を使用した場合と使用していない場合で，1 台の場合の計算時間は 10 秒程度（計算時間の 1/5000 程度）しか差が無いので，高速化率や並列化効率の差はほぼそのまま HDFS の有無による計算時間の差と考えることができる．

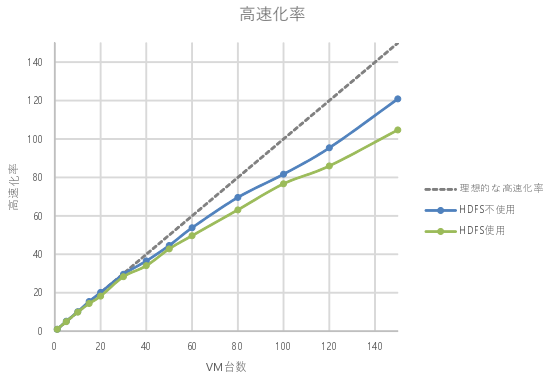

図 11 HDFS を使用した場合と使用していない場合の高速化率を示す．HDFS を使用した場合と使用していない場合で 1 台の時の計算時間はほぼ変わらないので，高速化率の差はほぼそのまま HDFS の有無による差と考えられる．

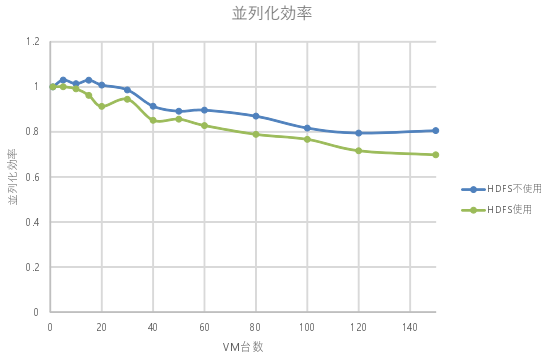

図 12 HDFS を使用した場合と使用していない場合の並列化効率を示す．HDFS を使用した場合と使用していない場合で 1 台の時の計算時間はほぼ変わらないので，並列化効率の差はほぼそのまま HDFS の有無による差と考えられる．

表 2　1 台使用した時の NC12・NC24 での NC6 に対する高速化率から，使用台数分線形に性能が向上したと仮定した時に予測される計算時間と，実際の計算時間とのずれを表わる．

VM の種類	1 台時の高速化率	予想計算時間	実際の計算時間	予測との差
NC12	1.87	576.0 s	572.9 s	-3.1 s
NC24	2.54	848.0 s	895.0 s	47.0 s

4.5.3 考察

HDFS なしで VM が 5 台・10 台・15 台のケースでは並列化効率が 1.0 を超えている．さらに VM が 10 台のケースに比べて 5 台と 15 台のケースで明らかに高い並列化効率となっている．これは以下のような原因によって引き起こされていると考えられる．

- VM の増加に伴い，1 台あたりの Apache Spark による処理が減少した
- 測定ごとにばらつきが存在していたために VM を 10 台利用したケースで 15 台利用したケースよりも並列化効率が低調であった．

HDFS を使用すると VM150 台の時に 15%程度性能が低下するが，時々ノードが停止してしまうことを考慮に入れると，HDFS を使用することは，性能の低下分を補うだけの価値があると考えられる．

5. 結論

本研究では，MEGADOCK によるタンパク質間相互作用の計算をパブリッククラウド上の分散 GPU 環境で高速に行う環境を構築した．その環境を計算機に精通していない研究者にも使用しやすくするために，PDB ファイルをアップロードするとパブリッククラウドを使用した MEGADOCK 計算を自動的に行う API を開発した．計算を高速に行うために実験を通して最適な条件を検討し，小規模な GPU 付き VM を多数使用する構成が高速であることが分かった．

謝辞　本研究の一部は，JSPS 科研費 (15K16081, 16K00388, 17H01814)，JST CREST「Extream BigData」(JPMJCR1303)，JST リサーチコンプレックス推進プログラム川崎拠点，AMED 創薬等ライフサイエンス研究支援基盤事業，日本マイクロソフト株式会社の支援を受けて行われた．

参考文献

[1] Lu, W., Jackson, J. and Barga, R.: AzureBlast : A Case Study of Developing Science Applications on the Cloud, *Design*, pp. 413–420 (online), DOI: 10.1145/1851476.1851537 (2010).
[2] Gunarathne, T., Wu, T. L., Choi, J. Y., Bae, S. H. and Qiu, J.: Cloud computing paradigms for pleasingly parallel biomedical applications, *Concurrency Computation Practice and Experience*, Vol. 23, No. 17, pp. 2338–2354 (online), DOI: 10.1002/cpe.1780 (2011).
[3] Ekanayake, J., Gunarathne, T. and Qiu, J.: Cloud technologies for bioinformatics applications, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 22, No. 6, pp. 998–1011 (online), DOI: 10.1109/TPDS.2010.178 (2011).
[4] Mrozek, D., Malysiak-Mrozek, B. and Kłński, A. K.: Cloud4Psi: Cloud computing for 3D protein structure similarity searching, *Bioinformatics*, Vol. 30, No. 19, pp. 2822–2825 (online), DOI: 10.1093/bioinformatics/btu389 (2014).
[5] Mrozek, D., Gosk, P. and Małysiak-Mrozek, B.: Scaling Ab Initio Predictions of 3D Protein Structures in Microsoft Azure Cloud, *Journal of Grid Computing*, Vol. 13, No. 4, pp. 561–585 (online), DOI: 10.1007/s10723-015-9353-8 (2015).
[6] Moghadam, B. T., Alvarsson, J., Holm, M., Eklund, M., Carlsson, L. and Spjuth, O.: Scaling predictive modeling in drug development with cloud computing, *Journal of Chemical Information and Modeling*, Vol. 55, No. 1, pp. 19–25 (online), DOI: 10.1021/ci500580y (2015).
[7] NVIDIA: サーバー用 Tesla GPU アクセラレータ |NVIDIA，入手先 ⟨http://www.nvidia.co.jp/object/tesla-servers-jp.html⟩ (参照 2018-01-22).
[8] Chen, R., Li, L. and Weng, Z.: ZDOCK: An initial-stage protein-docking algorithm, *Proteins*, Vol. 51, No. 1, pp. 80–87 (online), DOI: 10.1002/prot.10389 (2003).
[9] Ohue, M., Matsuzaki, Y., Uchikoga, N., Ishida, T. and Akiyama, Y.: MEGADOCK: An All-to-All Protein-Protein Interaction Prediction System Using Tertiary Structure Data, *Protein & Peptide Letters*, Vol. 21, No. 8, pp. 766–778 (online), DOI: 10.2174/0929866511320999005O (2014).
[10] Dean, J. and Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, *Communications of the ACM*, Vol. 51, No. 1, p. 107 (online), DOI: 10.1145/1327452.1327492 (2008).
[11] The Apache Software Foundation: Hadoop, available from ⟨http://hadoop.apache.org/⟩ (accessed 2018-01-22).
[12] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S. and Stoica, I.: Spark: Cluster Computing with Working Sets, *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, (HotCloud'10), Berkeley, CA, USA, USENIX Association, pp. 10–10 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1863103.1863113⟩ (2010).
[13] Foundation, T. A. S.: The Apache Software Foundation, available from ⟨https://www.apache.org/⟩ (accessed 2018-01-22).
[14] Microsoft: Microsoft Azure のドキュメント，入手先 ⟨https://docs.microsoft.com/ja-jp/azure/index⟩ (参照 2018-01-22).
[15] Chen, R., Mintseris, J., Janin, J. and Weng, Z.: A protein-protein docking benchmark, *Proteins*, Vol. 52, No. 1, pp. 88–91 (online), DOI: 10.1002/prot.10390 (2003).