Vol. 51 No. 5 情報処理学会論文誌 May 2010

仮想マシン動的再配置による大規模データアクセスの高速化

佐藤賢斗[†] 佐藤 仁[†] 松岡 聡[†,††]

近年、科学技術計算で使用されるデータが大規模化しており、データインテンシブアプリケーションのファイルアクセス性能の向上がより重要となっている．既存研究では、ファイルキャッシュや複製を行うことにより向上を図っているが、消費ストレージ容量やファイル転送時間が大きいという問題がある．そこで我々は、これらの問題を解決するため、仮想計算環境を想定したファイルアクセスの高速化手法を提案する．具体的には、ファイルアクセス時間と仮想マシンの移動時間からなるファイルアクセスの性能モデルとファイルの依存関係を表すマルコフモデルから最短経路探索を利用した最適化問題を解くことにより、ファイルアクセスを行うために最適な仮想マシンの位置を決定し、アプリケーションの実行中に仮想マシンを動的に再配置する．提案手法のプロトタイプを実装し評価した結果、仮想マシンを移動させない手法に比べ最大 23%、ファイルの存在する場所へ毎回移動しローカルアクセスを行う手法に比べ最大 41% のアプリケーションの実行時間を削減し、性能モデル及びマルコフモデルに基づき仮想マシンを適切に再配置することにより、ファイルアクセスのスループットが向上されることを確認した．

Improving Large-Scale Data Access Performace Using Virtual Machine Migration

Kento Sato,[†] Hitoshi Sato [†] and Satoshi Matsuoka[†,††]

Federated storage resources in geographically distributed environments are becoming viable platforms for data-intensive cloud and grid applications. To improve I/O performance in such environments, we propose a novel model-based I/O performance optimization algorithm for data-intensive applications running on a virtual machine, which determines virtual machine(VM) migration strategies, i.e., to which site a VM should be migrated, while minimizing the expected value of file access time. We solve this problem as a shortest path problem of a weighted direct acyclic graph (DAG), where the weighted vertex represents a location of a VM and expected file access time from the location, and the weighted edge represents a migration of a VM and time. We confirmed that the prototype of the proposed technique can achieve higher performance of BLAST than simple techniques, such as ones that never migrate VMs by 23% or always migrate VMs onto the locations that hold target files by 41%.

1. はじめに

近年、高エネルギー物理学、生物学、天文学などの大規模データ処理を伴なう科学技術計算の分野では、広域に分散した計算資源を利用した e-サイエンスが実用的になりつつある．広域に分散した複数のサイトの資源を集約して用いることで、単一のサイトでは取り扱えないほどの大規模な計算環境、ストレージを実現することができる．このような環境でのデータ共有ではデータへのアクセスの高速化が重要な課題である．従来の手法ではデータの複製作成やキャッシュ [1)2)] を行うことで、ネットワークスループットに起因するアプリケーションの実行性能の低下を避けている．しかし、データインテンシブアプリケーションでは、アクセスの対象となるファイルの総サイズが非常に大きくなるため、複製作成のための転送時間や消費ストレージ容量などのオーバーヘッドが非常に大きくなるという問題がある．

このような難しさは、仮想マシンを用いることで解決できる．これは、動作中のアプリケーションの状態を維持したまま、仮想マシンをデータの所在するノード上へマイグレーションすることで、データへの高速なアクセスが実現できると期待されるためである．しかしながら、以下の理由により仮想マシンを現在の拠点から移動させるべきか、また、どのサイトへ移動させるべきかの判断は容易ではない。

† 東京工業大学
Tokyo Institute of Technology
†† 国立情報学研究所
National Institute of Informatics

- アクセス対象のファイルの総サイズ、仮想マシンに割り当てられたメモリサイズ、実行環境のノード間のネットワークスループットなどの様々なパラメタにより仮想マシンの移動時間やファイルアクセス時間が変化する.
- BLAST[3] や Montage[4] などの科学技術計算で用いられるデータインテンシブアプリケーションでは、個々のファイルサイズは小さくても、アクセスされるデータセットの総サイズが大きい場合があるので、ファイル間のアクセスの依存関係を考慮して、仮想マシンを移動させた方が有効である.

そこで我々はこれらの課題を考慮し、仮想マシン上で実行されるデータインテンシブアプリケーションを対象にした仮想マシンマイグレーションによるデータアクセスの高速化アルゴリズム手法を提案する. 具体的には、前者の課題に対し (1) 仮想マシンマイグレーションの時間とファイルアクセス時間の性能モデルを構築し、後者の課題に対し、(2) ファイルアクセスの依存関係をマルコフモデルとして構築する. そしてこれら 2 つのモデルから、最短経路探索を利用した最適化問題を解くことによりデータアクセスに最適な仮想マシンの位置を決定する.

我々はこの提案アルゴリズム基づき仮想マシン動的再配置を行うシステムのプロトタイプを実装し、広域に分散した 3 サイトを模した環境で評価を行った. その結果、BLAST の実行時間を仮想マシンを再配置しない手法に比べ最大で 23%、ファイルの存在するノード上へ常に移動させローカル I/O を行う手法に比べ最大で 41%の削減し、性能モデルに基づき適切に仮想マシンを動的に再配置することで、データインテンシブアプリケーションの実行時間を削減できることを確認した.

2. 関 連 研 究

広域分散環境でのデータへのアクセスの高速化において重要となるのは、”いかに必要なデータをアクセス要求元の近くに置くか” という点である. 従来手法では、ファイルの複製作成やキャッシュなどが行われる[1)2)]. また、この他にもファイルの存在するノードでジョブを実行し、データアクセスの局所化を行う手法[5] などがあるが、このような手法を用いた場合においても、ジョブがアクセスするファイルが複数あり、それらが広域に分散した異なるノードに存在する環境では、リモートアクセスの発生は避けられず、オーバーヘッドとなり、適時ファイルの複製やキャッシュを行う必要が発生する.

近年、仮想計算環境として、仮想クラスタの利用が数多く提案されている. 仮想クラスタ[6)7)] は、仮想化技術を用いたコンピューティングクラスタであり、仮想マシンを計算ノードとして利用することで、下位の物理マシン環境とは独立に柔軟な環境を構築することができる. このような環境で仮想マシンやプロセスをマイグレーションすることにより、性能向上を行っている研究がいくつかある. Timothy ら[8] は、CPU、ネットワーク、ディスクの使用率から負荷の高い物理マシン上の仮想マシンを低い物理マシン上へ再配置することにより負荷分散を行う手法を提案している. Tatezono ら[9] は、並列アプリケーションの通信量や通信パターンに応じて仮想マシンの負荷分散を行っている. しかし、これらはデータインテンシブアプリケーションのワークロードは考慮していない.

また、大規模データ処理を伴う科学技術計算の多くのワークロードでは、アクセスされるファイル間に依存関係があるため、個々のファイルを独立に扱うのではなく、関連したファイルの集合を考慮した最適化を行うことができる. 例えば、Doraimani ら[10] は高エネルギー物理学で実際に利用されているアプリケーションのファイルアクセスをトレースし、クラスタリングの結果からファイルのキャッシュを行うことにより、ファイルアクセスの性能向上を実現している.

3. 仮想マシンマイグレーションを用いたファイルアクセス高速化手法

提案手法は、仮想マシンからのリモートファイルアクセスとマイグレーションのコストをモデル化し最適化問題を解くことにより将来のファイルアクセスを考慮した仮想マシンの最適な位置を決定する. ここでは、そのアルゴリズムの詳細を述べる.

3.1 仮想マシン動的再配置アルゴリズム

我々はファイルアクセスを行うサイトを頂点、仮想マシンマイグレーションによるサイト間の遷移を辺として構築した有向非循環グラフ (DAG) に対し、頂点の重みをそのサイトにおけるファイルアクセスの時間の期待値とし、辺の重みをそのサイトへ移動する仮想マシンのマイグレーション時間として最短経路問題を解くことにより、仮想マシンの最適な移動先を決定する.

アルゴリズムの詳細を説明するために例を用いて説明する. いま 3 サイト (Site1, 2, 3) で構成される環境に 3 つのファイルが図 1 のように分散されているとする. このとき Site 1 上の仮想マシンが File 3 へアクセスした場合の提案アルゴリズムによる最適な仮想マ

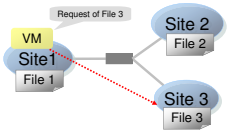

図 1 Site 1 上の仮想マシンが File 3 へアクセスする場合の例

シンの位置の決定方法を説明する．まず、仮想マシンが File 3 へアクセスする方法として 3 通り存在し、

- Site 1 に留まり、リモートからに File 3 へアクセスする方法 $(m_{11} + r_{13})$
- Site 2 へ移動し、リモートから File 3 へアクセスする方法 $(m_{12} + r_{23})$
- Site 3 へ移動し、ローカルに File 3 へアクセスする方法 $(m_{13} + r_{33})$

が挙げられる (図 2 の 1^{st} Access)．ここで、m_{ij} を Site i から Site j への仮想マシンマイグレーション時間、r_{ik} を Site i から File k のファイルアクセス時間とすると、今、確率 1 で File 3 へアクセスすることが分かっているので、仮想マシンの移動時間を含めた全ファイルアクセス時間の期待値はそれぞれ、$m_{11}+r_{13}$、$m_{12}+r_{23}$、$m_{13}+r_{33}$ となる (ただし m_{ii} は移動しないことを意味するので $m_{ii} = 0$ となる) ☆．この時点で単一ファイルのみ (ここでは File 3) を考慮して仮想マシンの位置を決定する場合は、これら 3 つの時間の中で最小の時間となるサイトへ移動すればよいが、データインテンシブアプリケーションでは多数のファイルで構成されるデータセットへアクセスするため、File 3 の後の将来にアクセスされるファイルを考慮して仮想マシンの最適な移動先を決定する必要がある．

そこで、今、2^{nd} Access におけるファイルアクセス確率をそれぞれ (File1, File2, File3)=(1.0, 0.0, 0.0) とすると、2^{nd} Access における各サイトにおけるファイルアクセス時間の期待値は、それぞれ r_{11}、r_{21}、r_{31} となり、また 3^{rd} Access においても同様にファイルアクセス確率をそれぞれ (File1, File2, File3)=(0.0, 0.9, 0.1) とすると、$0.9 \times r_{12} + 0.1 \times r_{13}$、$0.9 \times r_{22} + 0.1 \times r_{23}$、$0.9 \times r_{32} + 0.1 \times r_{33}$ となる☆☆．

いま将来の 3 番目のファイルアクセス (3^{rd} Access) まで考慮する場合について考えると、構築される DAG は図 2 の様になる．これら $3 \times 3 \times 3 = 27$ 通りのファイルアクセス方法のうち、将来のファイルアクセスを

☆ 仮想マシンマイグレーション時間 (m_{ij}) とファイルアクセス時間 (r_{ik}) は 3.2 章で説明する性能モデルより求めることができる

☆☆ ファイルアクセス確率は 3.3 章で説明するファイルアクセスの推移確率行列から求めることができる

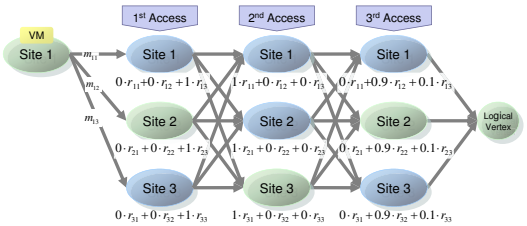

図 2 DAG の構築：頂点はファイルアクセスを行うサイトを表し、辺は仮想マシンの移動先を表す

考慮した上でファイルアクセス時間が最小となると期待される仮想マシンの位置は、構築された DAG の末尾にさらに論理的な頂点 (Logical Vertex) を重みが 0 の辺で結合した DAG の両端 (左端：Site 1、右端：Logical Vertex) の頂点の最短経路問題を解くことによって求めることができる．例えば Site 1 ⇒ Site 2 ⇒ Site 3 ⇒ Site 2 ⇒ Logical Vertex が最短経路であったとすると、今、File 3 をアクセスしている仮想マシンの最適な位置が Site 2 であると判断し、仮想マシンを Site 1 から Site 2 へ再配置する．

3.2 仮想マシンの性能モデル構築

3.1 節の提案アルゴリズムで使用される性能モデルについて説明する．ファイルアクセスの時間 (r) のモデルを式 (1) に示す．

$$r = \frac{file\_size}{\min\left(network, local\right)} \tag{1}$$

$file\_size$ はファイルサイズ $[MB]$、$network$ は仮想マシンが位置するサイトとアクセスするファイルが存在するサイト間のネットワークスループット $[MB/sec]$、$local$ はアクセスするファイルが存在するサイトのファイルシステムのスループット $[MB/sec]$ を表す．

次に、仮想マシンのマイグレーション時間 (m) のモデルを式 (2) に示す．

$$m = \frac{vm\_mem}{network} + init\_fin + iowait \tag{2}$$

vm_mem は仮想マシンに割り当てられたメモリサイズ $[MB]$、$network$ はマイグレーションを行うサイト間のスループット $[MB/sec]$、$init\_fin$ は前処理および後処理に要する時間 $[sec]$、$iowait$ はマイグレーション完了後ファイルアクセスを開始するまでの時間 $[sec]$ を表す．実際 $iowait$ はマイグレーション時間ではないが、便宜上この時間も仮想マシンマイグレーションの時間とみなす．

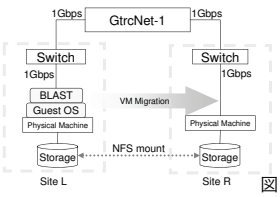

図 3 モデル構築環境

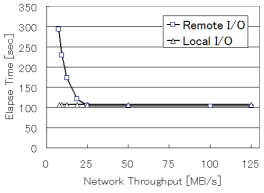

図 4 POSIX I/O による シーケンシャル読み出しの時間

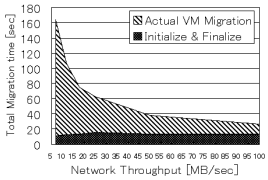

図 5 マイグレーション時間の内訳 VM Memory Size:1024MB

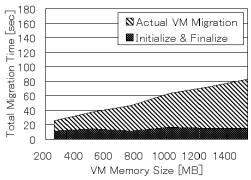

図 6 マイグレーション時間の内訳 Network Throughput:25MB/sec

次に、これらのモデルの構築方法について説明する．我々はこれらの性能モデルを構築するために、東京工業大学松岡研究室 PrestoIII クラスタ 2 ノードをネットワーク・エミュレータ GtrcNet-1[11] を介して接続した図 3 のような環境で予備実験を行った．一方のノードのローカル・ディスクに VM イメージを、他方のノードにアクセス対象データを用意し、NFS により仮想マシンがマイグレーション後も稼動できるようにした．クラスタの一ノードのメモリは 2GB、CPU は Opteron250(2.4GHz) × 2、NIC は Broadcom Corporation NetXtreme BCM5704 を搭載し、カーネルは 2.6.18-xen、Xen のバージョンは 3.1.0 を使用した．

まず、一方のノードから他方のノードに存在するファイルへアクセスした際の read アクセス時間のモデルを構築するため、2GB のファイルの全領域をシーケンシャルに read する実験をネットワークのスループットを変えて行った．図 4 にネットワークのスループットとそのときのファイルの読み出しに費やされた時間を表す．Local I/O はファイルが存在するノード上で同様の実験を行ったときの結果を表す☆．ネットワークのスループットが低い場合は、その部分がボトルネックとなり読み出しの実行時間に影響を与えているが、ネットワークのスループットの性能が高くなるにつれて逆にディスクやファイルシステム性能などに起因するローカルストレージへのスループットがボトルネックとなることがわかる．この結果からファイルアクセス時間のモデルを式 (1) とした．

☆ ネットワークを介していないのでネットワークスループットには無関係で一定

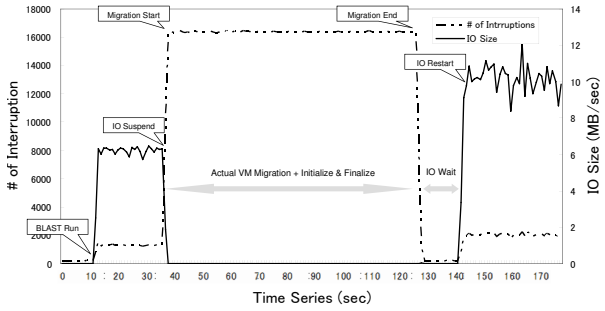

図 7 仮想マシンマイグレーション実行による IO 量と割り込み回数の変化

次に、仮想マシンのマイグレーションの時間のモデルを構築するための実験を行った．図 3 の環境において、バイオインフォマティクスの分野で広く利用されている相同性検索ツールである BLAST[3] を実行させ、実行途中で仮想マシンのマイグレーションさせる．データセットサイズは総サイズが 2.03GB であり、ネットワークのスループットと仮想マシンのメモリサイズを変化させて実験を行った．Xen では Stop-and-Copy Migration と Live Migration[12] という 2 種類の仮想マシンマイグレーション機能をサポートしているが、ここでは、モデル構築の容易な Stop-and-Copy Migration を対象とする．図 5 は、仮想マシンのメモリサイズを 1024MB に固定し、ネットワークのスループットを変化させたときの仮想マシンのマイグレーションに要した時間、図 6 は、ネットワークのスループットを 25MB/sec に固定し、仮想マシンのメモリサイズを変化させたときの仮想マシンのマイグレーションに要した時間、図 7 は、仮想マシンのマイグレーションを行ったときの、対象ファイルが格納されているノードの I/O 量と割り込み回数を表す．この測定には *vmstat* を用いた．図 5、図 6 から、仮想マシンのメモリイメージの転送に要する時間 (Actual VM Migration) は、ネットワークスループットと仮想マシンのメモリサイズに依存する項目であり、仮想マシンのメモリイメージに比例し、ネットワークスループットに反比例することがわかる．また、前処理や後処理に要する時間 (Initialize & Finalize) はネットワークスループットや仮想マシンのメモリサイズに依存せず一定であることがわかる．また同様の評価をアイドル中の仮想マシンで行った場合も同様の結果が得られた．さらに、図 7 から、マイグレーション開始直後、割り込み回数が増加し I/O はサスペンドされ、前処理、メモリイメージ転送、後処理を経てマイグレーションを完了するが (Actual VM Migration + Initialize &

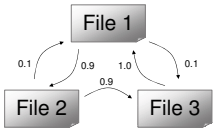

図 8 ファイルアクセスの依存関係を記述する推移図

Finalize)、しかし、完了後もしばらくの間 I/O が再開されないことがわかる (IO Wait). したがって、我々はこれらの実験の結果から、仮想マシンのマイグレーションに要する時間のモデルを式 (2) とした.

3.3 ファイルの依存関係を記述するマルコフモデル

提案する最適化アルゴリズムではファイルの依存関係を記述するために状態空間及び時間空間が共に離散的なマルコフモデルを用いる. マルコフモデルを構築するために、まずファイルアクセスの履歴からあるファイルから次にどのファイルにアクセスされたかを、カウントすることにより、あるファイルから次のファイルへどのくらいの確率でアクセスするかを求める. いま、対象とする環境にファイルが 3 つのみ存在し推移図が図 8 のようであったとすると、この推移図から以下のように推移確率行列と呼ばれる行列を記述することができる.

$$P = \begin{array}{c} \\ File1 \\ File2 \\ File3 \end{array} \begin{array}{c} \begin{array}{ccc} File1 & File2 & File3 \end{array} \\ \begin{pmatrix} 0.0 & 0.9 & 0.1 \\ 0.1 & 0.0 & 0.9 \\ 1.0 & 0.0 & 0.0 \end{pmatrix} \end{array}$$

この行列の i 行 j 列の要素 $p_{ij}^{(1)} (= P^1 = P)$ はある時点 n (n^{th} Access) において File i にアクセスしたとき、時点 $n+1$ ($n+1^{th}$ Access) に File j にアクセスする確率 $P\{X_{n+1} = File\ j | X_n = File\ i\}$ が $p_{ij}^{(1)}$ であることを表す. また、この推移確率行列の P^2 を計算することにより、時点 $n+2$ におけるファイルアクセス確率 $P\{X_{n+2} = File\ j | X_n = File\ i\} = P^2 = p_{ij}^{(2)}$ を求めることができる. 一般に推移確率行列は $P\{X_{n+k} = File\ j | X_n = File\ i\} = P^k = p_{ij}^{(k)}$ を満たす. したがって、ファイルアクセスのパターンがマルコフモデルに従うとすると、ファイルアクセス履歴から任意の時点 n におけるファイルアクセス確率を求めることができる.

4. 提案手法のプロトタイプ

我々は、3 章の提案アルゴリズムに基づいて仮想マシンを動的に再配置するシステムのプロトタイプを実装した. アーキテクチャの概要を図 9 に示す. プロトタイプシステムは、以下の 3 つの要素からなる.

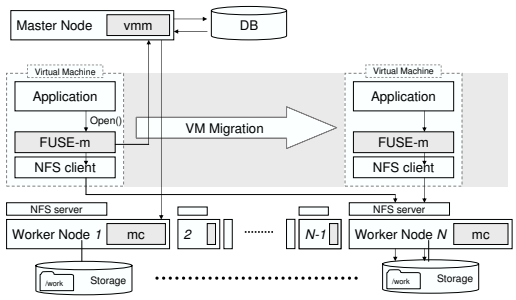

図 9 プロトタイプのアーキテクチャ

FUSE Monitor(FUSE-m) FUSE[13] を用いて実装されたファイルシステム. VM は NFS を用いて各 Worker Node のデータセットが保存されている特定のディレクトリ (/work) をそれぞれ異なるディレクトリにマウントし、どの Worker Node 上のデータセットへもアクセスができるようになっており、そのマウントされたディレクトリに対し FUSE を用いてファイルアクセスを監視する. そのディレクトリ内のファイルに対するシステムコール open() が呼ばれると、open されたファイルを特定し、その情報を vmm に通知する. その後は通常通りのファイルアクセスが継続される.

Virtual Machine Manager(vmm) FUSE-m からオープンされたファイルの通知を受けると、過去のファイルアクセス履歴、及び、3 章で説明した最適化アルゴリズムに基づいて仮想マシンの最適な位置を決定し、移動させる必要がある場合には mc に対し移動させる仮想マシンとその移動先を通知する. もし移動させる仮想マシンが既に仮想マシンマイグレーションの実行中である場合は、時間的優先に基づいてその通知を破棄する.

Migration Controller(mc) vmc から仮想マシンマイグレーションの要求があると、通知された移動先へ仮想マシンマイグレーションを実行する. プロトタイプでは Stop-and-Copy Migration を使用する.

vmm、mc は Python を用いて実装し、仮想マシンモニタとして Xen[14] を使用した.

5. 実　　験

仮想マシンを動的に再配置することによるデータインテンシブアプリケーションの性能向上を確認するための実験をプロトタイプを用いて行った。

5.1 実 験 環 境

評価では、東京工業大学松岡研究室 PresotIII クラ

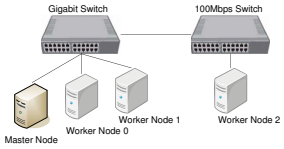

図 10 実験環境

表 1 使用ノードの仕様 (PrestoIII)

CPU	Quad-Core AMD Opteron 2346 HE × 2
Chipset	nVIDIA MCP55V-Pro
Memory	DDR2 SDRAM DIMM (4GB)
Local Disk	320GB Serial ATA (16MB Cache)
Network	nVIDIA MCP55V-Pro Dual-port Ethernet

スタノードを使用し、仮想マシンが自由に移動可能な Worker Node を 3 台と提案アルゴリズムを計算して仮想マシンを再配置する Master Node 1 台を図 10 のように接続した．使用した各ノードのハードウェア仕様は表 1 の通りである．仮想マシンモニタとして Xen3.1.0[14]、カーネルは xen kernel (2.6.18-xen) を使用した．また、スループットの低い環境を構成するためにスループットが 11MB/sec の 100Mbps のスイッチを使用した．性能モデルで使用するネットワークスループットは iperf、ローカルファイルシステムのスループットは hdparam を用いて計測した．

5.2 アプリケーション設定

仮想マシン上で実行するアプリケーションは BLAST を対象した．使用した 7 つのデータセットのサイズとデータセットを構成するファイルの数を図 11 に示す．今回の評価では NCBI BLAST[3] からダウンロード可能な 26 種類データベースのうち 7 種類のデータベースを使用した．アクセス対象ファイルの依存関係を記述するマルコフモデルの構築では、予め BLAST を実行し各データセットを構成するファイルのアクセス順序を抽出し、その結果からデータセット”内”のファイル間のマルコフモデルを構築した．今回は 7 つデータセットへはランダムにアクセスされることを想定してデータセット”間”の推移確率はどのデータセットに対しても等確率で推移するように記述したマルコフモデルを使用した．また、考慮する将来のファイルアクセスは対象ファイルを含め 5 アクセス分を考慮して仮想マシンの最適な位置を決定する．これは事前にシミュレーションをしたところ 5 アクセス以上考慮しても、同じ経路を決定したためである．仮想マシンのメモリサイズは 1024MB とし、仮想マシンのディスクイメージは Worker Node 1 に配置し、各 Worker Node が NFS を用いることにより仮想マシンが Woker Node 1 以外のノードへ移動したとしても仮想マシンがディスクイメージへアクセスできるようにした．

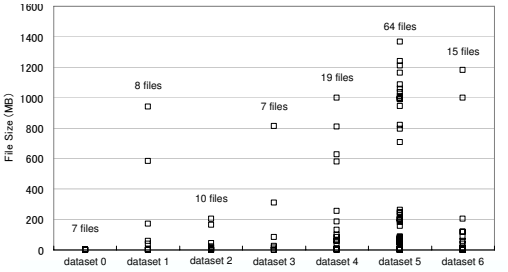

図 11 評価で使用した 7 種類のデータセットを構成するファイル数とサイズ

5.3 BLAST 実行時間の評価

単一の仮想マシンの再配置による BLAST の実行時間の影響を調べる実験を行った．BLAST の実行では、同一の塩基配列をクエリシーケンスとして各ノード上に保存されているデータベースの dataset 0 から 6 まで順に相同性検索を行った．この実験では、データセットへのアクセス順序とデータセットの保存ノードを変えた 2 つの評価を行った．その設定を表 2 に示す．これらは以下のことを想定した．

- 評価 **1**:ファイルサイズの大きなデータセットがネットワークのスループットが低いノード (Worker Node 2) 上に集中している場合
- 評価 **2**:ファイルサイズの小さいデータセットがスループットの低いノード上に存在した場合

表 2 データセットのアクセス順序 (dataset0 から 6 へ順にアクセスする) と格納されている Worker Node 番号

dataset #	評価 1	評価 2
dataset 0	Worker Node 0	Worker Node 2
dataset 1	Worker Node 1	Worker Node 0
dataset 2	Worker Node 0	Worker Node 1
dataset 3	Worker Node 1	Worker Node 0
dataset 4	Worker Node 2	Worker Node 1
dataset 5	Worker Node 2	Worker Node 2
dataset 6	Worker Node 2	Worker Node 2

我々は提案手法の有効性を示すため、以下の 2 つの手法と比較評価を行った．

- **No Migration**: 初期位置 (Worker Node 1) から全てのファイルへアクセスし、仮想マシンの再配置を行わない手法
- **Migration**: ファイルが保存されているノード上へ毎回移動させる手法

図 12 に、評価 1 における仮想マシンマイグレーショ

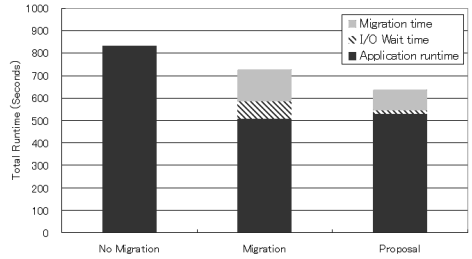

図 12 評価 1: ネットワークのスループットが低いノード (Worker Node 2) 上に集中している場合

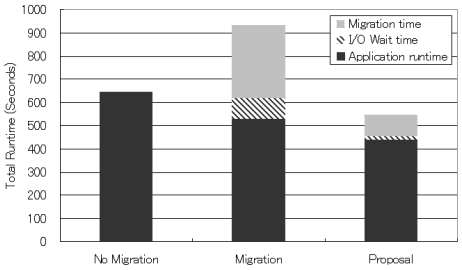

図 13 評価 2: 一部の小規模のデータセットがスループットの低いノード上に存在した場合

ン時間を含めた 7 種類のデータベースへアクセスする BLAST の実行時間を示す．Migration time は仮想マシンマイグレーションを行っている時間の合計、I/O Wait time は仮想マシンマイグレーション後、実際に I/O が再開されるまでの時間の合計、Application runtime はそれ以外の時間の合計を表す．サイズの大きいファイルが単一ノードに集中している場合は、ファイルの存在するノードへ仮想マシンを移動させることにより、移動させない場合に比べ BLAST の実行時間を 12.4%削減できることを確認した．さらに、提案手法を用いることで、ファイルのサイズやネットワークスループットの状況に応じて、ファイルアクセスに最適な仮想マシンの位置を決定することができ、23.4%削減できることを確認した．これは提案手法により不必要な仮想マシンのマイグレーションを防いでいるためである．Migration 手法では全てのファイルに対しローカルにアクセスしているため、BLAST の実行時間のみを考えると他の手法に比べると最小の時間であるが, 高いネットワークスループットを確保できる環境でも、仮想マシンを移動させ、ローカルファイルアクセスを行うため、全体の実行時間は提案手法より長くなってしまう．

次に、評価 2 における BLAST の実行時間を図 13 に示す． Migration 手法による全体の実行時間が一番長くなっている．これは、スループットの低いネットワークで接続された Worker Node 2 へ不必要な仮想マシンマイグレーションを行ったためである．この場合、Worker Node 2 上に置かれた dataset 0 の総ファイルサイズは小さく、かつ、Worker Node 2 がスループットの低いネットワークを介して接続されているため、仮想マシンを移動させることが逆に全体の実行時間の増加を引き起こしてしまう．このため仮想マシンを再配置しない手法の方が全体の実行時間が短くなった．一方、提案手法ではマイグレーションによるオーバーヘッドを最小限に抑えている．これはファイルサイズや仮想マシンのメモリサイズなどの情報から仮想マシンマイグレーションを実行すべきか否かを決定しているからである．

5.4 議　　論

5.4.1 ファイル間の依存関係の考慮

仮に今回の評価に対して単一ファイル、つまり将来のファイルアクセスは全く考慮せず、これから仮想マシンがアクセスするファイルのみを対象とした場合 (DAG は 1^{st} Access のみ構築する場合) について考える．この場合、仮想マシンは現在の位置に留まるか、もしくはファイルの存在する場所へ移動するかの 2 択しかない．いま対象ファイルのサイズを xMB とすると、今回の評価環境では、Worker Node 2 から他の Worker Node 間のネットワークスループットが $11MB/sec$、ローカルファイルシステムのスループットが $55MB/sec$、仮想のメモリサイズが $1024MB$、仮想マシンの前処理、後処理と IO Wait の時間はそれぞれ $13sec$、$15sec$ であったので、前者の時間は $\frac{x}{\min(11,55)}$ 秒、後者の時間は仮想マシンマイグレーション時間とローカルアクセスの時間の和になるので、$\left(\frac{1024}{11}+13+15\right)+\frac{x}{55}$ 秒となる．したがって、この場合 $x = 1665MB$ より大きなサイズのファイルに対して仮想マシンマイグレーションを行い、小さなサイズのファイルに対して仮想マシンを移動させない方がよいということを意味する．しかし、図 11 を見ると分かるように今回使用したデータセットの個々のファイルのサイズは全て $x = 1665MB$ 未満である．これは単一ファイルのみ着目しただけでは、仮想マシンマイグレーションは実行されず No Migration 手法と同じ結果となることを意味する．このため、ファイルアクセスパターンをマルコフモデルとしてモデル化しそれに基づいてデータセット単位で仮想マシンの位置を決定することが非常に重要であることがわかる．

また、今回の実験ではファイルアクセスのマルコフモデルは予め用意したものを用いたが、アプリケーション実行中にもアクセス履歴を記録し、一定のタイ

ムインターバルでマルコフモデルを更新することにより、より正確な依存関係を抽出することが可能である.

5.4.2 提案アルゴリズムのオーバヘッドとスケーラビリティ

FUSE 内で 285 回の open() を監視し、それを vmm へ通知するために要する時間の合計は約 9 秒であった. これは 1 アクセスあたり 31 ミリ秒にあたり、広域分散環境上でのファイルアクセス時間と比べると小さい.

また、スケーラビリティに関して、最短経路探索に要する時間は、考慮する将来のファイルアクセス数とサイト数によって増加する. しかし、前者の数の増加に伴い、ファイルアクセス確率は収束するため、考慮するファイルアクセス数が一定数以上では、仮想マシンの移動経路に変化はない. したがって、考慮する将来のファイルアクセス数を多くする必要はない. 一方、後者において、今回の実験では 3 サイトを想定したが、考慮する未来のファイルアクセス数が 5 アクセス分の場合、例えば Intrigger(13 サイト) のような大規模な環境でも最短経路探索の計算時間は約 17 ミリ秒、仮に 100 サイトの場合も約 980 ミリ秒で完了する. さらに、提案アルゴリズムの計算は、仮想マシン上の I/O と並行して行うため、I/O がブロックされることない.

6. おわりに

我々は、仮想マシン上で実行されるデータインテンシブアプリケーションを対象にした仮想マシンのマイグレーションによるデータアクセスの高速化手法を提案した. 提案手法のプロトタイプを広域に分散した 3 サイトを模した環境で評価した結果、仮想マシンを移動させない手法やファイルの存在する場所へ毎回移動しローカルアクセスを行う手法に比べデータインテンシブアプリケーションの性能向上を確認した.

今後の課題として、今回の実験では 1 つ仮想マシンのみを想定して評価したが、仮想マシンが複数存在するような仮想クラスタの場合は、同一ファイルへのアクセス集中によるコンテンションが生じる可能性がある. このため、仮想マシン間のロードバランスを考慮する必要がある.

謝辞 本研究の一部は科学研究費補助金特定領域研究 (18049028) の支援によって行われた.

参考文献

1) Osamu Tatebe, Youhei Morita, Satoshi Matsuoka, Noriyuki Soda, and Satoshi Sekiguchi. Grid datafarm architecture for petascale data intensive computing. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, pp. 102 – 110, 2002.
2) Yuya Machida, Shin'ichiro Takizawa, Hidemoto Nakada, and Satishi Matsuoka. Multi-replication with intelligent staging in data-intensive grid applications. In *In Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, pp. 88–95, 2006.
3) NCBI BLAST. `http://www.ncbi.nlm.nih.gov/BLAST`.
4) Montage: An Astronomical Image Mosaic Engine. `http://montage.ipac.caltech.edu/`.
5) Srinath Shankar and David J. DeWitt. Data driven workflow planning in cluster management systems. In *Proceedings of the 16th IEEE International Symposium on High Performance Distributed Computing*, pp. 127–136, New York, NY, USA, 2007.
6) I.Foster, T.Freeman, K.Keahey, D.Scheftner, B.Sotomayor, and X.Zhang. Virtual Clusters for Grid Communities. In *IEEE International Symposium on Cluster Computing and the Grid*, pp. 513– 520, 2006.
7) Amazon EC2. http://aws.amazon.com/ec2/.
8) Timothy Wood, Prashant Shenoy, and Arun Venkataramani. Black-box and Gray-box Strategies for Virtual Machine Migration . In *4th USENIX Symposium on Networked Systems Design & Implementation, NSDI2007*, 2007.
9) Masaki Tatezono, Hidemoto Nakada, and Satoshi Matsuoka. Mpi environment with load balancing using virtual machine. In *Symposium on Advanced Computing Systems and Infrastructures SACSIS*, pp. 525–532, 2006.
10) Shyamala Doraimani and Adriana Iamnitchi. File grouping for scientific data management: lessons from experimenting with real traces. In *Proceedings of the 17th international symposium on High performance distributed computing*, 2008.
11) Y. Kodama, T. Kudoh, R. Takano, H. Sato, O. Tatebe, and S. Sekiguchi. GNET-1:Gigabit Ethernet Network Testbed. `http://projects.gtrc.aist.go.jp/gnet/`.
12) Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hanseny, Eric July, Christian Limpach, Ian Pratt, and Andrew Wareld. Live Migration of Virtual Machines. In *NSDI*, 2005.
13) FUSE: Filesystem in Userspace. `http://fuse.sourceforge.net/`.
14) Xen Community. http://xen.xensource.com/.