レプリカ管理システムを利用したデータインテンシブアプリケーション向けスケジューリングシステム

町田悠哉[†] 滝澤真一朗[†]
中田秀基[††,†] 松岡聡[†,†††]

グリッド環境において既存のスケジューリングシステムはデータ入出力を共有ファイルシステムや単純なステージング機構を利用して行っている。しかしこれらの手法ではデータ保持ノードはアクセス集中によりパフォーマンスが低下、そして最悪の場合にはハングアップしてしまう。またユーザが同一のデータセットを利用する多数のタスクからなるジョブを実行した場合、スケジューリング後に毎回同じデータをステージングするのは非効率である。そこで本研究では複数ノードへ O(1) の転送時間でデータを複製できるスケーラブルなレプリカ管理システムをステージング機構として利用し、レプリカを再利用するような効率的なスケジューリングを可能とするシステムを提案する。プロトタイプシステム上でサンプルアプリケーションを実行したところ従来の共有ファイルシステムやステージング機構を利用したものより高い性能が確認できた。

A scheduling system coupled with a replica management system for data-intensive applications

Yuya Machida,[†] Shin'ichiro Takizawa,[†] Hidemoto Nakada[††,†] and Satoshi Matsuoka[†,†††]

Existing scheduling systems for the Grid mostly handle huge I/O via a shared file system or simple staging. However, when numerous nodes access a single I/O node simultaneously, major performance degradation occurs, or in a worst case, causes I/O nodes to hang. Moreover, when a user launches a job consisting of hundreds or even thousands of tasks which share the same data set, it becomes extremely inefficient to stage essentially the same data set to each compute node after every dynamic brokering and allocation of the compute nodes. Instead, we propose to utilize a replica management system that embodies a scalable multi-replication framework as a data staging mechanism, where multiple copies could be made in O(1) transfer time as well as make intelligent reuse of already-created replicas in scheduling for efficiency. A prototype executing a sample data-intensive application proved to be quite superior to shared files or traditional staging techniques.

1. はじめに

高エネルギー物理学や天文学、バイオインフォマティクスなどの諸分野において大規模なデータ処理を必要とするデータインテンシブアプリケーションの実行環境としてグリッド環境の利用が高まっている。例えばバイオインフォマティクス分野で広く利用されている BLAST[1] はクエリとなるヌクレオチドやタンパク質の配列に類似した配列を数ギガバイトクラスのデータベースから検索するアプリケーションであり、計算時間が数時間に及ぶこともある。ユーザはこのようなジョブをバッチジョブとしてサブミットし、スケジューリングシステムがそれらを実行するのに適したマシンを決定する。実行時に利用されるファイルは共有ファイルシステムやステージング機構を経由してスケジューリングシステムによって割り当てられたマシンから利用される。しかしユーザがサブミットしたジョブが同一データセットを利用する多数のタスクで構成されているような場合、多数のノードが同時に単一のデータ保持ノードにアクセスしてしまう。これによりデータ保持ノードにおいてアクセス集中が発生し、パフォーマンスの大幅な低下につながり、最悪の場合にはデータ保持ノードがダウンしてしまう可能性もある。ま

† 東京工業大学
Tokyo Institute of Technology
†† 産業技術総合研究所
National Institute of Advanced Industorial Science and Technology
††† 国立情報学研究所
National Institute of Infomatics

た計算ノードはジョブがスケジュールされると必要なデータを毎回ステージングしなければならないため同一データを利用するジョブが多数あるような場合には非効率的である。以上のように従来の手法ではデータインテンシブアプリケーションを実行する環境として不十分である。

そこで本研究ではレプリカ管理システムとスケジューリングシステムを連動させ、データのロケーションを考慮した最適なジョブスケジューリングを実現し、データインテンシブアプリケーションの効率的な実行環境を提供することを目的とする。プロトタイプシステムとしてバッチスケジューリングシステムを拡張し、複数ノードへ O(1) の転送時間でレプリカを作成することができるレプリカ管理システムと連携させた。これによりレプリカロケーションを考慮したスケジューリングを行うことが可能になった。本システムの有効性を示すため単純なデータインテンシブアプリケーションを実行し、従来の手法よりも高い効率とスケーラビリティが確認できた。

2. 関 連 研 究

2.1 Stork

Stork[2] はデータ転送ジョブのためのスケジューリングシステムである。さまざまなストレージシステム、転送ミドルウェア、プロトコルに対応しているだけでなく、プラグインにより容易に拡張可能でグリッド環境の不均質性を隠蔽できる。転送ジョブ開始時には自動的に利用可能なプロトコルを調べ、適当なプロトコルで転送を開始するようになっている。Stork では、データ転送ジョブの記述言語として ClassAd[3] を利用しており、この中に利用したいプロトコルを記述することも可能である (図 1)。また Stork はリトライ機能によりネットワーク、ストレージシステム、ミドルウェア、ソフトウェア障害を隠蔽することも可能で、グリッド環境の不安定性にも対応している。データ転送ジョブが指定した時間を超過しても転送が終了しない場合、“kill and start” 機構によりジョブが再実行され、転送ジョブのハングアップを防ぐことも可能である。

Stork は DAGMan(Directed Acyclic Graph Manager)[4] を介して Condor[5)~7)] と連携して利用することによりデータインテンシブアプリケーション実行環境を提供している[8]。DAGMan は Condor および Stork のメタスケジューラであり、DAG ファイルに記述されたジョブの依存関係を解決し、計算ジョブを Condor に、データ転送ジョブを Stork にサブミットする。これにより単一のフレームワークでデータをステージングさせてから計算ジョブを実行することが可能になる。この手法はリモートサイトからローカルのストレージシステムにデータを転送し、そのファイル

```
[
dap_type = ‘‘transfer’’;
src_url  = ‘‘any://src.host.name/tmp/data’’;
dest_url = ‘‘any://dest.host.name/tmp/data’’;
alt_protocols = ‘‘gsiftp-file, http-file’’;
]
```

図 1 データ転送ジョブを記述した ClassAd の例

を少数のノードで共有して計算するような場合には有効であるが、データを共有するノード数が大きくなるとアクセス集中によりパフォーマンスが低下してしまう。また Stork はデータ転送ジョブを記述するファイルに転送元・転送先を指定しなければならないため、各ノードの負荷状況やネットワーク状況などに応じてデータ転送元・転送先を選択することができない。これは計算ジョブを Condor、データ転送ジョブを Stork がそれぞれ独立にスケジューリングしているためで、データインテンシブアプリケーションの効率的な実行環境は提供できていない。

2.2 並列・分散ファイルシステム

PVFS[9] や Lustre[10] などの並列ファイルシステムではストライピングによりファイルが分割され、複数のディスクに保存される。これによりファイルの読み書き時のディスクアクセスが複数ディスクに分散され、広帯域のディスク I/O が実現される。しかしこれらのファイルシステムはクラスタ環境での利用が想定されているためセキュリティや非対称なネットワーク環境などに対する考慮がなされておらず、グリッド環境でデータインテンシブアプリケーションを実行するために利用することはできない。

Gfarm[11] はネットワーク上に分散したストレージを統合し、仮想的に単一のファイルシステムを提供するグリッドファイルシステムである。owner-computes ルールによりローカルアクセスが積極的に利用されるようにプロセスをスケジュールできるが、同一データを利用するジョブが多数サブミットされた場合にはアクセス集中が発生してしまう。Gfarm ではファイルのレプリカを作成することによりアクセス集中を回避することが可能であるが、レプリカ作成のデータ転送では 1 対 1 転送が想定されているため非効率的である。

Batch-Aware Distributed File System(BAD-FS)[12] はストレージの制御機構を外部スケジューラにエクスポーズした分散ファイルシステムである。BAD-FS は内部制御を外部スケジューラにさらすことで広域データ転送を最小限に抑えて効率的に I/O インテンシブなワークロードを実行することを目的としている。しかしジョブのサブミット時にユーザは詳細なデータ利用パターンを記述する必要があり、複雑なワークフローの場合には大きな労力が求められる。また利用するデータはサブミット時にすでに決定されているため必ずしも最適なロケーションのデータが利用されると

は限らない。

2.3 レプリカ管理システムを利用した複製手法

データのレプリカを作成することによりアクセス遅延やデータローカリティ、ロバストネスなどを改善できる。そのためデータインテンシブアプリケーションの実行環境においてもデータのレプリカ作成による性能向上が図られている。レプリカデータの管理サービスは Globus Toolkit[13] や EU DataGrid(EDG)[14] などによって提供されている。Globus Toolkit のデータ管理コンポーネントは Replica Location Service(RLS)[15]、GridFTP[16]、RFT(Reliable File Transfer) によって構成されている。RLS は論理ファイル名と物理ファイル名のマッピングを階層的に管理し、GridFTP や RFT などを用いてファイルの転送を行う。RLS は論理ファイル名と物理ファイル名のマッピングを管理する Local Replica Catalog(LRC) と、論理ファイルと LRC のマッピングを管理する Replica Location Index(RLI) で構成される。RLS ユーザは RLI の数や冗長度などを変更することによりパフォーマンスや信頼性といったシステムの特性を調節できる。EDG のレプリカ管理サービス Reptor[17] はさまざまなコンポーネントをプラグインできるように設計されており、Replica Location Service(RLS) や Replica Metadata Catalog Service(RMC)、Replica Optimization Service(ROS) などのコンポーネントで構成される。RLS は Globus の RLS の構造をベースにしており、GUID(Grid Unique Identifier) と PFN(Physical File Name) のマッピングを管理する。RMC は GUID で表されるオリジナル物理ファイルのメタデータおよび LFN(Logical File Name) と GUID のマッピングを管理する。ROS はアクセスコストをもとに最適なレプリカを選択する。

しかしこれらのデータ管理サービスは 1 対 1 の通信・転送プロトコルを想定しているため、現在のアーキテクチャでは複数ノードが I/O ノードに同時アクセスした場合、I/O ノードのパフォーマンスの低下は避けられない。またさまざまなデータレプリケーションアルゴリズムが提案されている[18)~20)] が、単一データへアクセスが集中する場合は想定されておらず、それを回避するための効率的な並列レプリケーションアルゴリズムは考えられていない。またジョブスケジューリングとデータ配置はそれぞれが独立に行われているため、アクセス集中が発生してしまう上にジョブがスケジューリングされるマシンのロケーションが全く考慮されないレプリケーションが行われてしまう。

3. 設　　計

同一データセットを使用するタスク群で構成されたデータインテンシブなジョブを従来のシステムで実行する場合には下記のような問題が発生する。

- ジョブスケジューリングとは独立にシステムまたはユーザがデータの転送元・転送先を決定するため、最適な決定がなされない。
- 同一データセットを使用するタスクが同一実行マシンに複数回割り当てられた場合、割り当てられるたびに毎回同一データの転送が行われる。
- 1 対 1 転送が想定されており、同一データセットに多数のノードから同時にアクセスされる状況については考慮されていないためデータ転送のパフォーマンスが低下してしまう。

これらの問題を解決するためスケジューリングシステムとレプリカ管理システムの統合を提案する。スケジューリング時にデータのロケーション情報を考慮することにより、コストの高いステージングが回避できると同時に高速なローカル I/O が利用できるようになり、データインテンシブアプリケーションを効率的に実行可能となる。以下では本システムの設計方針について述べる。

- ジョブスケジューリングとレプリカ管理の連動
  従来は独立に行われてきたジョブスケジューリングとレプリカ管理を相互に連携させることで、ジョブスケジューリングにデータのロケーション情報が、レプリカの作成にスケジューリング情報が加味される。これにより最適な転送元からデータを必要とするノードにレプリカが作成され、低コストでデータがステージングされたノードでジョブが実行される。これはネットワークのバンド幅やレイテンシなどから各実行ノードへのレプリカ作成コストを見積もっておき、ジョブの要求を満たすノードの中でこのコストが低いノードへの転送およびジョブスケジュールを決定することで実現できる。
- データの再利用性を意識したスケジューリング
  対象とするデータインテンシブアプリケーションは概して扱うデータサイズが大きいため、各タスクを効率的に実行するためには実行マシンにすでにデータが格納されているかどうかがスケジューリング時に考慮すべき重要な指標となる。そしてデータの格納情報を利用することにより、同一データを使用するタスクが複数回スケジュールされた場合にもデータを毎回転送することを回避できる。ローカルにデータが格納されているような場合はスケジューリング時に考慮される転送コストを低く見積もっておくことでそのデータを使用するジョブが割り当てられやすくなる。
- 同一ファイルに対するレプリケーションリクエストの集約
  ユーザが同一データセットを使用する複数のタスクで構成されるジョブを実行した場合、既存のスケジューリングシステムではデータ保持ノードにおいてデータへのアクセス集中が発生してしまい、

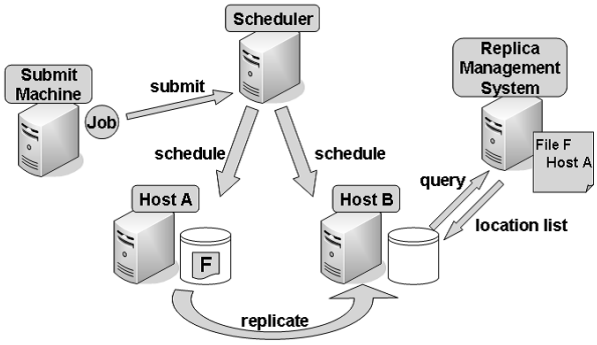

図 2 システムの概要

パフォーマンスの低下につながる。このような状況に対処するため同一ファイルに対する同時アクセス数を低く抑える必要がある。そこで近傍ノード群の転送要求を集約し、1 ノードのみがデータ保持ノードから転送を行い、その後他ノードへ並列に転送することで同時アクセスを避けることができる。

本システムの概要を図 2 に示す。レプリカ管理システムはレプリカのロケーション情報を管理する。各マシンはレプリカ管理システムで管理している各ファイルについてローカルにレプリカを作成するのに必要なコストを見積もっておき、OS やメモリ容量などのマシン情報とともにスケジューラに送信する。レプリカ管理システムに登録されているファイル F を必要とするジョブをユーザがサブミットするとスケジューラは受信したマシン情報やデータ転送に必要なコストを考慮して適当なマシンへスケジュールする。すでにローカルディスクにファイル F が格納されている A が遊休状態の場合には A にジョブがスケジュールされる。すでに A でジョブが実行されている場合は B にスケジュールされ、ファイル F をローカルに転送するのに最適なマシンをレプリカ管理システムに問い合わせ、転送を開始する。ファイルの転送が終了するとそのファイルのロケーションが新たにレプリカ管理システムに登録され、ジョブが実行される。複数のジョブが同時にファイル F の転送要求を送信した場合はシステムがそれを検知し、自動的に並列レプリケーションを行う。

4. プロトタイプシステムの実装

上記の設計をもとにスケジューリングシステムとレプリカ管理システムを連携させ、プロトタイプシステムを実装した。スケジューリングシステムとして Jay[21] を用いた。Jay は Condor の各コンポーネントを Java 実装したバッチスケジューリングシステムであり、GSI を認証機構として利用することでよりセキュアなシステムとなっている。Jay を拡張し、スケジューリング機構そのものにファイルレプリカを意識した動作を組み込んだ。レプリカ管理システムとしては Globus の RLS サービスをベースとし、高速ファイル転送ツール Dolly+[22] を組み合わせたマルチレプリケーションフレームワーク[23] を利用した。以下ではマルチレプリケーションフレームワークの概要およびレプリカ管理システムと連携したスケジューリングの流れについて説明する。

4.1 マルチレプリケーションフレームワークの概要

マルチレプリケーションフレームワークはレプリカの位置情報を管理する Replica Location Service(RLS) とマルチキャスト転送技術を利用したデータ転送サービスで構成されるレプリカ管理システムである。ファイルのレプリケーションリクエストを送信するとファイルの転送元として最適なホストを自動的に選択され、転送完了後はそれ以降のファイル転送のためロケーション情報が登録される。さらに同一ファイルに対するレプリケーション要求は集約され、データ転送時にアプリケーションレベルマルチキャストが利用される。以下ではデータ転送元ホスト選択機構およびデータ転送機構について述べる。

4.1.1 転送元ホスト選択機構

図 3 に示したように転送元ホストの選択はレプリカのロケーション情報を管理する Replica Location Service(RLS) と複数あるレプリカの中から転送に適したレプリカを選択する Replica Selector の 2 つのコンポーネントが使用される。

RLS はレプリカカタログというデータベースでファイルの位置情報を管理し、レプリカカタログへの操作要求を処理する RLS サーバと、RLS サーバに対してレプリカの登録、削除、位置情報取得といった要求を出す RLS クライアントから構成されるサービスである。レプリカカタログではファイルは論理名で管理され、各論理ファイルに対して複数のサイトに分散した複数の物理実体の位置情報が登録される。Replica Selector は登録されたロケーションの中からネットワーク情報などをもとに転送に適したものを選び出す。現在の実装ではレプリカの選択基準として RTT 値を採用しており、その値が最小のものを最適な候補として選択する。

4.1.2 データ転送機構

データ転送サービスの概要を図 4 に示した。サイト内の複数ホストがサイト外にある同一レプリカを要求する際に、サイト内で要求を集約し、代表ノードのみがそのレプリカを取得し、その後サイト内でマルチキャスト転送を行うという設計である。そこでマルチレプリケーションフレームワークでは転送サービスを次の 3 つのコンポーネントから構築している。

転送サーバ 同じサイト内のホストのみに対して、アプリケーションレベルのマルチキャストファイル転送ツールである Dolly+を用いてファイル転送

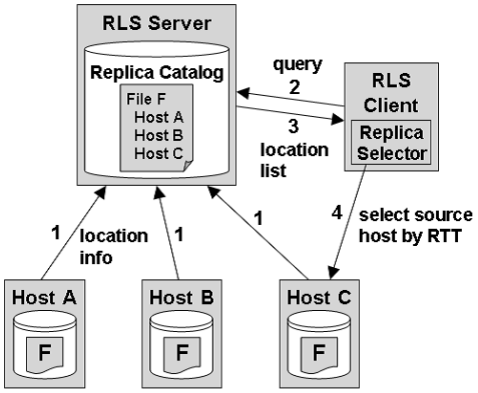

図 3 転送元ホスト選択機構

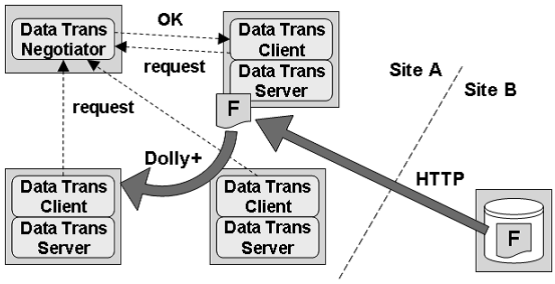

図 4 データ転送機構

を行う。

転送クライアント 同じサイト内のホストからは Dolly+で、サイト外のホストからは HTTP を用いてファイルを取得する。

転送ネゴシエータ サイト外ホストへのファイル要求を 1 つに集約するサービスである。サイト内の複数ホストがサイト外の同一レプリカを要求した際には、1 つのホストにのみそのレプリカの取得を許可し、他のホストには許可を得られたホストから転送するように返信する。

4.2 スケジューリングシステムとマルチレプリケーションフレームワークの連携

Condor と同様に Jay はセントラルマネージャにおいてマッチメイキング[24] を行い、ClassAd の rank の値に応じてジョブのスケジューリングを決定する。このマッチメイキング時にジョブが利用するデータのレプリカロケーション情報を考慮に入れるため、rank 値にその情報を反映させなければならない。そこで本システムではセントラルマネージャがロケーション情報 (現在の実装では RTT) に応じた値を rank 値に付加している。そして最終的に最大 rank 値となったマシンにジョブをスケジュールする。ディスクスペースの管理もマッチメイキング時に行われており、データセットを格納するためのディスクスペースを保持しないマシンへはスケジュールされないようになっている。

ここで本システムがどのように動作するかについて説明する (図 5)。ユーザはジョブが利用するデータをあらかじめ論理名を指定して RLS サーバに登録しておく必要がある。サブミットファイルには登録したデータの論理名を `transfer_replica_files` に記述する必要がある。論理名で指定されたデータの物理パスは `$(Replica_Files)` で参照できる。

(1) Startd はマシン情報を記述した ClassAd を定期的にセントラルマネージャに発行する。このマシン情報には OS やメモリ容量だけでなくレプリカに関する情報も含まれている。具体的には RLS サーバに登録されている各ファイルをローカルに転送してくるのに必要なコスト (現在の実装では RTT) に応じた値である。Schedd も同様にジョブ情報を定期的にセントラルマネージャに発行する。

(2) セントラルマネージャは受信した ClassAd をもとにマッチメイキングを行い、どのマシンでジョブを実行するかを決定する。この際に ClassAd に記述されたレプリケーションコストに応じた値を rank に加え、レプリカのロケーションが考慮されたスケジューリングを行う。最大 rank 値を持つマシンにジョブをスケジュールすることを決定し、実行マシンおよびサブミットマシンに通知する。

(3) セントラルマネージャから通知が来るとサブミットマシンで稼働している Schedd は Shadow プロセスを起動する。Shadow はジョブ情報を記述した ClassAd をスケジュールされた実行マシンの Startd に送信する。Startd は Shadow から ClassAd を受け取るとそのジョブを実行できるか検証を行い、実行可能な場合は Starter プロセスを起動し、OK メッセージを返信する。Shadow は OK メッセージを受信すると Starter にジョブの実行を依頼する。

(4) 実行を依頼されたジョブが RLS サーバに登録されたデータを利用する場合、Starter はマルチレプリケーションフレームワークの転送クライアントにデータ転送を要求する。転送元ホストがサイト外の場合、転送クライアントは転送ネゴシエータにデータ転送開始の許可をもらう。転送元ホストが同一サイト内の場合もしくは転送ネゴシエータに転送開始を許可されなかった場合、Dolly+を用いてデータ転送を行う。データ転送完了後、ジョブ実行を開始する。

5. 評 価 実 験

5.1 評価実験の概要

本システムの有効性を示すためサンプルアプリケーションを実行した。実験では提案手法とともに従来手

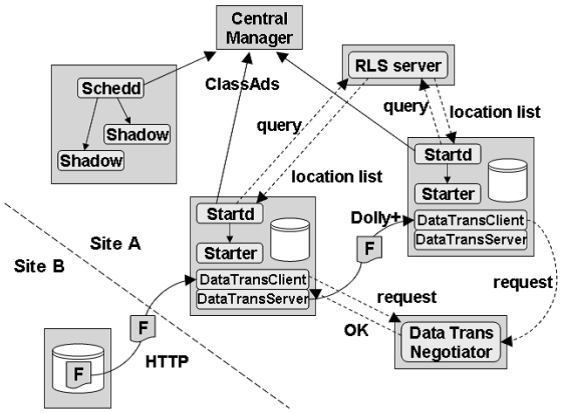

図 5 データインテンシブアプリケーション実行の流れ

```
executable = application
input      = input.$(Process)
output     = output.$(Process)
error      = error.$(Process)
log        = application.log
arguments  = $(Replica_Files)
transfer_replica_files = data1,data2
queue 100
```

図 6 サブミットファイルの例

表 1 PresotIII クラスタのスペック

CPU	Opteron 242
Memory	2GBytes
OS	Linux 2.4.30
Network	1000Base-T

法においてアプリケーションを実行し、データ再利用性の向上およびデータへのアクセス集中の回避を確認した。サンプルアプリケーションとしてバイオインフォマティクスの分野において広く利用されている相同性検索ツール BLAST を利用した。実験には本研究室の PrestoIII クラスタを用いた (表 1)。セントラルマネージャ、サブミットマシンを担当するマシンをそれぞれ 1 ノードずつ用意した。実行マシンは $n(n = 4, 8, 16, 32)$ ノード用意し、$5n$ 個のジョブをサブミットした。ジョブは 5 個のヌクレオチド配列をクエリとしてヌクレオチドデータベース nt に対して検索を行うというものである。実験では以下で述べる 4 つの手法を比較した。

ファイル共有手法 (NFS) データベースファイルを共有するため広く利用されている NFS(Network File System) を利用した。1 ノードを NFS サーバとして用意し、そのノードに格納されたデータベースファイルを実行マシンで共有する。

ステージング手法 (Staging) データベースファイルをサブミットマシンのローカルディスクに格納しておき、セントラルマネージャにスケジュールされたマシンへ scp によりステージングする。

提案手法 (Replica) RLS サーバとデータ転送ネゴシエータが稼働するノードを用意し、そのノードのローカルディスクに格納されたデータベースファイルのロケーションを登録しておく。

理想状態 (Ideal) すべての実行マシンのローカルディスクにデータベースファイルを格納し、ステージングの必要がない。

5.2 評価実験結果

実行マシン 16 ノードでファイル共有手法を用いた際のジョブをサブミットしてからすべてのジョブが終了するまでの稼働中ジョブ数の推移を図 7 に示した。ステージング手法、提案手法を利用した状況でのジョブ数の推移についてもそれぞれ図 8, 9 に示した (比較のため図 8 には共有手法のグラフ、図 9 には共有手法およびステージング手法のグラフも重ねて表示している)。図 8, 9 の斜線部分はデータ転送を行っていることを示す。これらの結果よりファイル共有手法では NFS サーバへのアクセスが集中し、パフォーマンスが大幅に低下していることがわかる。ステージング手法においても特に最初のステージング時にアクセス集中が生じていることがわかる。また同一データを反復して転送しており、非効率なデータ転送が行われていることもわかる。一方、本システムでは Dolly+によりアクセス集中を回避して高速なデータ転送が行われていることが確認できる。さらにレプリカが効率的に再利用されることにより非効率な転送が回避できていることもわかる。本手法はレプリケーションコストを大きく低減することによりステージング手法に対して 44.3%、共有手法に対して 57.5%の性能向上が得られた。

ジョブの平均実行時間 (転送時間+計算時間) を図 10 に示した。図 10 より本システムがファイル共有手法やステージング手法と比較して非常に高い性能を示しており、理想状態とほぼ同等の性能を達成していることがわかる。さらにノード数に関係なくほぼ同等の性能を示しており、非常にスケーラブルなシステムであることがわかる。

次にマルチレプリケーションの効率を評価するため実行マシンが 2 ノードの場合と 16 ノードの場合におけるレプリケーション時間を比較した。レプリカが作成されるデータベースファイルのサイズは約 3GB である。レプリケーション時間は 2 ノードの場合は 3.23 分 (30MB/s)、16 ノードの場合は 6.4 分 (120MB/s) で効率的に転送されていることがわかる。

6. おわりに

本研究ではデータインテンシブアプリケーションを効率的に実行するためレプリカ管理、並列レプリケーション、ジョブスケジューリングを連携したシステムを提案した。プロトタイプシステムとしてバッチスケ

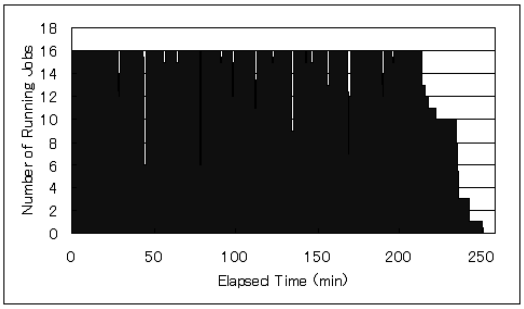

図 7 ファイル共有手法におけるジョブ数の推移

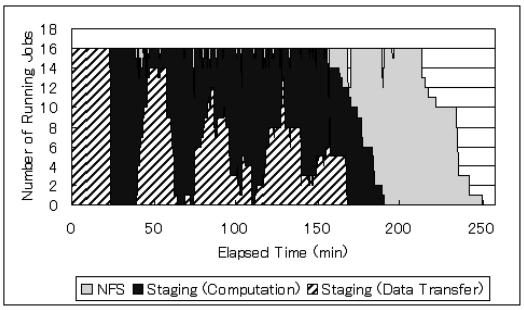

図 8 ステージング手法におけるジョブ数の推移

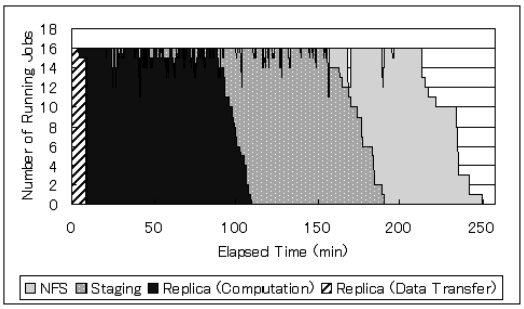

図 9 提案手法におけるジョブ数の推移

ジューリングシステムを拡張し、複数ノードへの並列ファイル転送ができるマルチレプリケーションフレームワークと連動させ、レプリカロケーションを意識したスケジューリングを実現した。本システムの有用性を確認するためサンプルアプリケーションを実行した結果、提案手法が従来のファイル共有手法や単純なステージング手法と比較して高い効率とスケーラビリティを備えていることが示された。

今後の課題としては以下が挙げられる。

- 計算とデータ転送の同時実行
  現在の実装では実行マシンにジョブがスケジュールされたとしてもデータ転送が完了するまでジョブの実行は開始されない。そのためデータ転送中は計算資源が利用されていないにも関わらずジョブがすでに割り当てられているため他のジョブをスケジュールすることができない。そこで実行マシンがジョブ実行中の場合、セントラルマネージャーに送信するマシン情報にどこに負荷をかけているかを表すような情報も含めることで、データ転送中の実行マシンに計算ジョブを割り当てたり、計算中の実行マシンにデータ転送をさせるということができるようになり、さらに資源の利用効率を高めることが可能となる。
- より現実的なシナリオでの評価実験
  本研究で用いたデータインテンシブアプリケーションは非常に単純で、実験環境も小規模であった。そこで今後、複数のファイルを利用するようなアプリケーションについても評価を行う必要がある。またグリッド環境での振る舞いを評価するため複数サイトを利用したシステムの評価も行う必要がある。

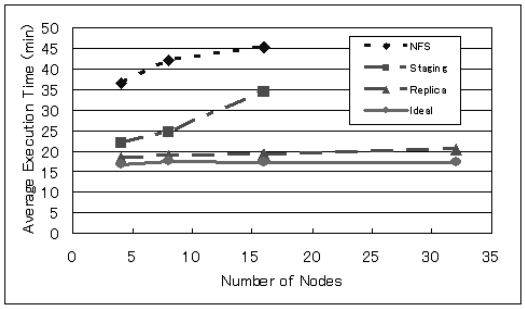

図 10 ジョブの平均実行時間

参 考 文 献

1) NCBI BLAST. http://www.ncbi.nlm.nih.gov/BLAST/.
2) T. Kosar and M. Livny. Stork: Making Data Placement a First Class Citizen in the Grid. *Proceedings of 24th IEEE International Conference on Distributed Computing Systems (ICDCS2004)*, 2004. Tokyo, Japan.
3) M. Livny, R. Raman, and T. Tannenbaum. Mechanisms for High Throughput Computing. *SPEEDUP Journal*, Vol. 11(1), 1997.
4) Directed Acyclic Graph Manager. http://www.cs.wisc.edu/condor/dagman.
5) Condor Project Homepage. http://www.cs.wisc.edu/condor/.
6) M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. *Proceedings of 8th International Conference of Distributed Computing Systems*, pages 104–111, 1988.
7) D. Epema, M. Livny, R. Dantzig, X. Evers, and J. Pruyne. A Worldwide Flock of Condors: Load Sharing among Workstation Clusters. *Journal on Future Generations of Com-*

puter Systems, Vol. 12, 1996.

8) G. Kola, T. Kosar, and M. Livny. A Fully Automated Fault-tolerant System for Distributed Video Processing and Off-site Replication. *In the 14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV2004)*.

9) P. Carns, W. Ligon III, R. Ross, and R. Thakur. PVFS: A parallel file system for linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, 2000. USENIX Association.

10) lustre. http://www.lustre.org/.

11) O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, and S. Sekiguchi. Grid datafarm architecture for petascale data intensive computing. *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 102–110, 2002.

12) J. Bent, D. Thain, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Explicit Control in a Batch-Aware Distributed File System. In *Proceedings of the First USENIX Symposium on Networked Systems Design and Implementation (NSDI '04)*, San Francisco, CA, March 2004.

13) I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Internatinal Journal of Supercomputer Applications*, Vol. 11(2):115–128, 1997.

14) EU DataGrid project. http://www.eu-datagrid.org/.

15) A. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf. Performance and scalability of a replica location service. *The Thirteenth IEEE International Symposium on High-Performance Distributed Computing*, 2004.

16) W. Allcock, J. Bresnaham, I. Foster, L. Liming, J. Link, and P. Plaszczac. Gridftp update january 2002. *Globus Project Technical Report*, 2002.

17) D. et al. Bosio. Next-Generation EU DataGrid Data Management Services. In *Computing in High Energy Physics (CHEP 2003)*, La Jolla, CA, March 2003.

18) K. Ranganathan and I. Foster. Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. In *Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, Edingurgh, Scotland, July 2002.

19) K. Ranganathan and I. Foster. Identifying Dynamic Replication Strategies for High Performance Data Grids. In *Proceedings of International Workshop on Grid Computing*, Denver, CO, November 2002.

20) A. Takefusa, O. Tatebe, S. Matsuoka, and Y. Morita. Performance Analysis of Scheduling and Replication Algorithms on Grid Datafarm Architecture for High-Energy Physics Applications. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*, pages 34–43, June 2003.

21) 町田悠哉, 中田秀基, 松岡聡. ポータビリティの高いジョブスケジューリングシステムの設計と実装. In **情報処理学会研究報告** *2004-HPC-99 (SWOPP 2004, pp217-222, July 30-August 1, 2004)*, 2004.

22) A. Manabe. Disk cloning program 'dolly+' for system management of pc linux cluster. *Computing in High Energy Physics and Nuclear Physics*, 2001.

23) S. Takizawa, Y. Takamiya, H. Nakada, and S. Matsuoka. A Scalable Multi-Replication Framework for Data Grid. *Proceedings of the 2005 International Symposium on Applications and the Internet (SAINT 2005 Workshops)*, January 2005.

24) R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed Resource Management for High Throughput. *Proceedings of 7th IEEE International Synposium on High Performance Distributed Computing*, July 28-31 1998.