社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

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†,†††

† 東京工業大学　〒 152–8552 東京都目黒区大岡山 2–12–1
†† 産業技術総合研究所　〒 305–8568 茨城県つくば市梅園 1–1–1
††† 国立情報学研究所　〒 101–8430 東京都千代田区一ツ橋 2–1–2
E-mail: †{machida,takizawa}@matsulab.is.titech.ac.jp, ††hide-nakada@aist.go.jp, †††matsu@is.titech.ac.jp

あらまし　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。
キーワード　グリッド、スケジューリング、レプリカ管理システム、データインテンシブアプリケーション

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

Yuya MACHIDA†, Shin'ichiro TAKIZAWA†, Hidemoto NAKADA††, and Satoshi MATSUOKA†,†††

† Tokyo Institute of Technology　Ookayama 2–12–1, Meguro-ku, Tokyo, 152–8552 Japan
†† National Institute of Advanced Industrial Science and Technology　Umezono 1–1–1, Tsukuba, Ibaraki, 305–8568 Japan
††† National Institute of Informatics　Hitotsubashi 2–1–1, Chiyoda-ku, Tokyo, 101–8430 Japan
E-mail: †{machida,takizawa}@matsulab.is.titech.ac.jp, ††hide-nakada@aist.go.jp, †††matsu@is.titech.ac.jp

Abstract　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.
Key words　Grid, Scheduling, Replica Management System, Data-Intensive Application

1. はじめに

高エネルギー物理学や天文学、ライフサイエンスなどの分野において大規模なデータ処理能力を必要とするデータインテンシブアプリケーション実行のためグリッド環境の利用が高まっている。例えばライフサイエン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、計算時間が数時間にも及ぶことがある。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実行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ルさ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をデータステージング機構として利用し、レプリカロケーションを意識した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] はデータ配置ジョブ用のスケジューラで、アプリケーションから障害を隠蔽できる。この障害隠蔽は “kill and start” 機構により実現されており、指定した時間を超過してもデータ転送が終了しなかった場合に自動的に転送が再実行される。Stork ユーザはデータ転送要求時に図 1 のような ClassAd [3] を用意する必要がある。

Stork は DAGMan(Directed Acyclic Graph Manager) [4] と連携して利用される。DAGMan は Condor [5]〜[7] および Stork のメタスケジューラで、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の ClassAd 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。

```
[
  dap_type = ‘‘transfer’’;
  src_url  = ‘‘any://src.host.name/tmp/data’’;
  dest_url = ‘‘any://dest.host.name/tmp/data’’;
  max_retry = 10;
  restart_in = ‘‘2 hours’’;
]
```

図 1　データ転送ジョブを記述した ClassAd の例
Fig. 1　Example of a ClassAd describing a data transfer job

2.2 BAD-FS

Batch-Aware Distributed File System(BAD-FS) [8] 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 Replica Location Service

Globus Toolkit [9] は Replica Location Service(RLS) [10]、GridFTP [11]、RFT(Rapid File Transfer) によって構成されたデータ管理コンポーネントを提供している。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 などを用いてレプリカ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行う論理ファイルと LRC のマッピングを行う Local Replica Catalog(LRC) と Replica Location Index(RLI) で構成される。

RLS はデータのレプリケーションによりアクセスレイテンシの低減、データのローカリティの向上や耐故障性を達成している。しかしデータ転送に 1 対 1 通信が用いられているため同一ファイルに対するアクセス集中を回避することができない。

3. 設　　計

データアクセスの集中によるパフォーマンスの低下を避けるため、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レプリカのロケーション

情報をもとに効率的にレプリカデータ再利用することで高速かつ低コストなローカル I/O を利用できるようになる。以下では本システムの設計方針について述べる。同一データセットを利用するようなジョブもコストの高いステージングをすることなく効率的に実行できるように設計されている。

- データの再利用性を意識したスケジューリング

本システムが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ので、各タスクを効率的にスケジュールするためには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。スケジューリングシステムとレプリカ管理システムを統合することによりデータのロケーションを考慮したスケジューリングが可能となり、データ保持ノードへの効率的なスケジューリングが実現される。

- レプリケーションコストを意識したスケジューリング

データインテンシブアプリケーションはデータ転送が完了するまで実行を開始することができないので、実行マシンはジョブがスケジュールされたとしてもデータ転送中はアイドル状態である。この待機 (アイドル) 時間は、転送すべきデータサイズは大きいため無視できるものではない。そこでバンド幅やレイテンシなどのネットワーク情報からデータを転送するのに必要なコストを見積もり、そのコストの低い実行マシンへのスケジューリングができることが必要である。

- 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用いる複数のタスクからなるような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、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本システムでは同一ファイルに対する転送要求を自動的に検出し、それらの要求を集約し、並列にレプリケーションを行う。

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レプリケーションを行う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。ジョブをスケジュールされたマシンはファイル F を転送してく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、転送要求を送信する。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る。ロケーション情報登録後、ジョブ実行が開始される。複数のジョブが同時にファイル F の転送要求が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。

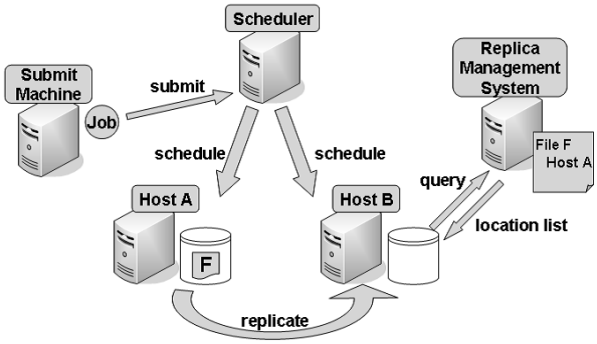

図 2　システムの概要
Fig. 2　System Overview

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 [12] 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+ [13] 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 [14] 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

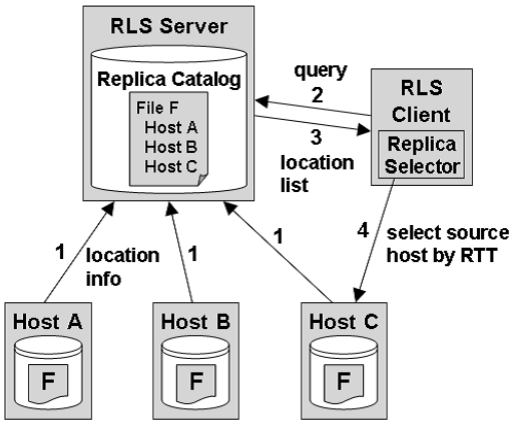

図 3　転送元ホスト選択機構
Fig. 3　A Source Host Selection Mechanism

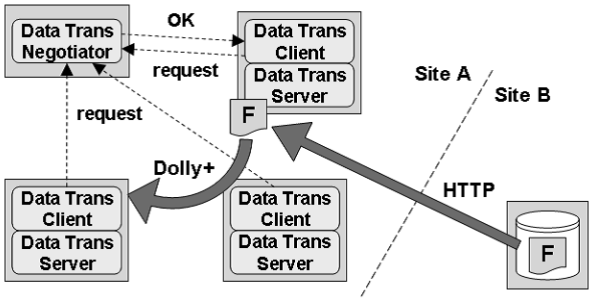

図 4　データ転送機構
Fig. 4　A Data Transfer Mechanism

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 [15] 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そのホストにレプリケートするのに必要なコスト情報 (現在の実装では RTT) 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。

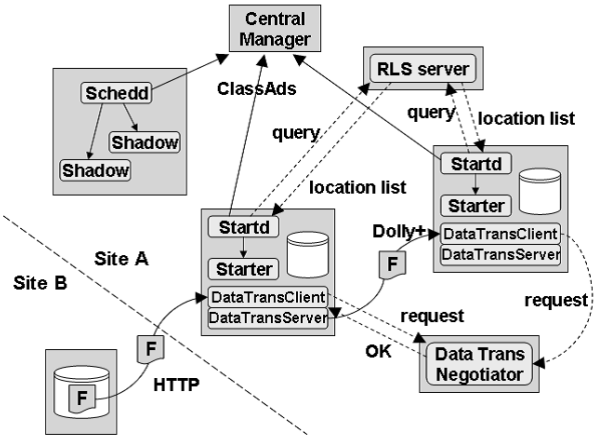

図 5　データインテンシブアプリケーション実行の流れ
Fig. 5　A Flow of the Execution of Data-Intensive Applications

```
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
Fig. 6　A example of a Submission File

表 1　PresotIII クラスタのスペック
Table 1　Specification of PrestoIII Cluster

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

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。実験では本システムだけなく、従来の手法を利用したシステム上でもアプリケーションを実行し、それぞれの実行時間を比較した。サンプルアプリケーションとしてホモロジー検索ツール 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。

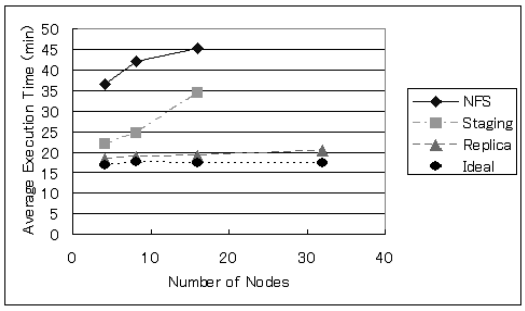

図 7　Starter プロセスの平均実行時間
Fig. 7　Average Execution Time of Starter Process

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

Starter プロセスの平均実行時間を図 7 に示した。図 7 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。

実行マシン 16 ノードでファイル共有手法、ステージング手法、レプリカステージング手法を利用した状況における実行中の Starter プロセス数の推移をそれぞれ図 8, 9, 10 に示した (灰色部分はデータ転送を行っていることを示す)。この結果よりファイル共有手法では NFS サーバにアクセスが集中し、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確認できる。一方、本システムではレプリカ管理システムによりデータ転送が効率的に行われており、さらにレプリカが効率的に再利用されていることが確認できる。

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スケジューリングシステムとレプリカ管理システムの連携を提案した。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と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を統合し、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

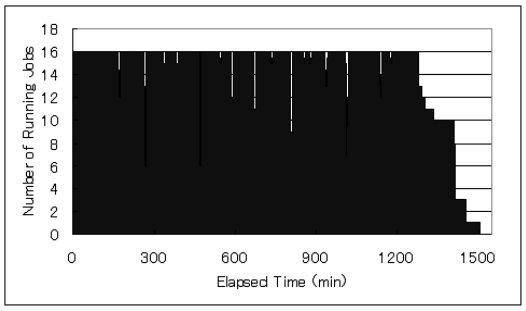

図 8　ファイル共有手法におけるジョブ数の推移
Fig. 8　Number of Running Jobs with NFS

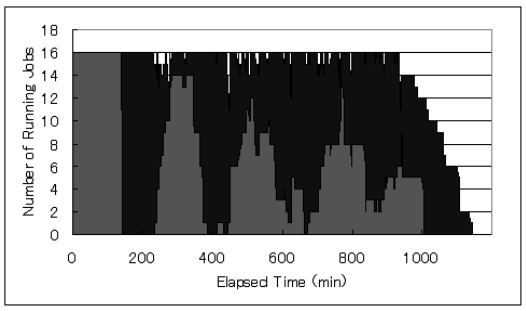

図 9　ステージング手法におけるジョブ数の推移
Fig. 9　Number of Running Jobs with a Simple Staging Technique

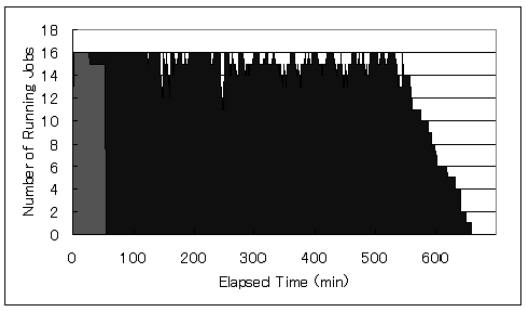

図 10　提案手法におけるジョブ数の推移
Fig. 10　Number of Running Jobs with a Propose Technique

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。

- ジョブの特性を考慮したスケジューリング

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ためデータ転送中は計算資源が利用されて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。そこでデータ転送を 1 つのジョブとして分離し、それぞれのジョブの特性に応じてスケジューリングできるようにすることで資源の利用効率がさらに上がると考えられる。具体的にはデータ転送ジョブの実行中は計算資源が遊休状態になってしまうのでそのマシンへ計算ジョブを割り当てることで資源全体の利用効率が上昇する。

- レプリカの動的管理

レプリカ管理システムがレプリカの作成・削除要求の頻度を調べ、それに応じて動的にレプリカ数を調整するような機構を組み込むことにより資源の利用効率が上昇し、ユーザがディスクスペースを管理する必要もなくなる。

- 大規模環境における評価

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、複数のレプリカファイルを利用するような大規模かつ実際的なアプリケーションを利用して現実的なシナリオにおけるシステムの評価を行う必要がある。

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 Computer Systems*, Vol. 12, 1996.

[8] 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.

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

[10] 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.

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

[12] "町田悠哉, 中田秀基, and 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.

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

[14] 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.

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