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。そこで本研究ではレプリカ管理とジョブスケジューリングをタイトに結合し、データを効率的に再利用する。プロトタイプシステムとして複数ノードへ 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利用し、効率的なファイル転送を提供するとともにデータ転送と計算を同時実行するような効率的なスケジューリングを可能にした。評価実験によりプロトタイプシステム上で従来手法よりも効率的なジョブ実行、スループット向上が達成されたことを確認した。

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. So we propose to tightly couple replica management and computation scheduling in order to reuse already replicated data effectively. We implemented a prototype system which uses a replica management system that embodies a scalable multi-replication framework, where multiple copies could be made in O(1) transfer time, and enables scheduling computation and data trasfer to single node simultaneously. The evaluation result shows our proposed technique performs superior to the traditional techniques and improves the throughput.

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。ユーザは通例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の実行に適したマシンを決定する。実行時に利用されるデータ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実行マシン上で利用される。しかしデータインテンシブ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、大幅な性能低下につながってしまう。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とスケジュー

† 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

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することにより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。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るとともにデータ転送中の遊休サイクルを有効利用を可能にした。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、従来手法よりも高い転送効率とスケーラビリティ、そしてスループットの向上が確認できた。

2. 関連研究

2.1 Stork

Stork[1] 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。Stork は DAGMan(Directed Acyclic Graph Manager)[2] を介して Condor[3),4)] 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。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。しかし Stork ではデータ転送ジョブのサブミットファイルにユーザが転送元・先ノードを指定する必要があり、ユーザとリソース双方にとって最適な転送ノードが選択されず、アクセス集中などが発生してしまう。サブミット時に最適なノード選択がされたとしてもリソース状況は動的に変化するため転送開始時には最適ではなくなる可能性が高い。このような状況は計算ジョブとデータ転送ジョブをそれぞれ独立に管理しているために生じると考えられる。

2.2 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、耐故障性などの向上をもたらす。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。Globus Toolkit[5] のデータ管理コンポーネントは Replica Location Service(RLS)[6]、GridFTP[7]、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。しかし Globus などで提供される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

データインテンシブなジョブを効率的に実行するためにはジョブ実行時の高速なデータアクセスを提供することが重要となるため、本システムでは各実行マシンのローカルディスクへのステージングによるローカル I/O の積極的な利用を行う。しかし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。

- ユーザがロケーションなどを考慮して意識的にレプリカ作成・アクセス分散などの対処をとる必要がある。
- 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。
- 同一データを使用するタスクが飽和している場合も実行後ステージングされたデータは消去されるため同一データを反復的にステージングする可能性がある。
- 効率的なデータ転送が行われたとしても対象とするデータサイズが大規模であるため転送中の実行マシンの遊休サイクルが発生してしまう。
- 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ジョブスケジューリングとデータ管理の統合を提案する。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されローカル I/O が利用できるようになると同時に転送中の実行マシンの計算サイクルも有効に利用される。以下で本システムの設計方針について述べる。

- データロケーションの仮想化によるユーザ負荷の軽減
  データロケーションの管理を行うことによりユーザにデータロケーションを意識させないようにするとともに自動的に低コストのステージングが行われるようにし、ユーザの利用負荷を軽減する。
- ジョブスケジューリングとデータ管理の連動
  従来は独立に行われてきたジョブスケジューリングとデータ管理を連携させることで、ジョブスケ

ジューリングにデータのロケーション情報を加味し、コストの低い転送を行う。そこでロケーション情報をもとに各マシンは定期的にデータをステージングするのに必要なコストを見積もっておき、スケジューリング時にその情報を利用してより転送コストの低いノードへのスケジューリングを行う。

- データのキャッシング
  同一データを使用するジョブが多数サブミットされた場合、同一マシン上で同一データを使用するジョブが連続して実行される可能性がある。従来のシステムではジョブ実行後ステージングされたデータはすぐに消去されてからマシンが解放される。そこでデータをジョブ実行後すぐには消去せずローカルにキャッシングした状態でマシンを解放し、キャッシングされたデータのロケーションを登録することで同一データの非効率な反復転送が避けられ、リソースの効率的な使用につながる。
- データ転送と計算の同時実行
  対象アプリケーションが扱うデータサイズは大規模であるため、いかに効率的なステージングを行ったとしても転送コストは無視できず、遊休サイクルが発生してしまう。そこでジョブがどのような状態であるか判断し、計算リソースを待機中のジョブにその遊休サイクルを割り当てる。転送が終了すると計算ジョブをサスペンドし、データインテンシブジョブの計算を開始する。また同様に計算実行中にはデータ転送を行い、遊休サイクル時間を低減させるとともにシステム全体のスループット向上を実現する。
- 同一ファイルに対する転送リクエストの集約
  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、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 1 に示す。データ管理システムはデータのロケーション情報を管理する。各マシンはデータ管理システムからロケーション情報を問い合わせ、ステージングコストを見積もり、マシン情報とともにスケジューラに送信する。ファイル F を使用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。すでにファイル F がローカルに格納されている A が遊休状態の

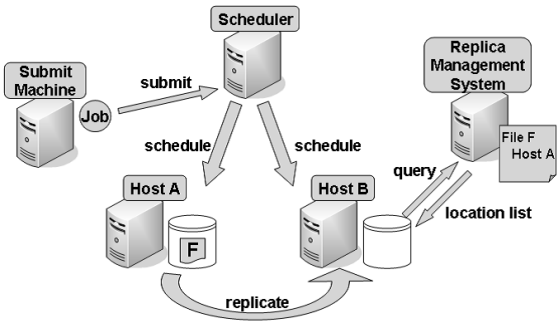

図 1 システムの概要

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転送コストの低い B にスケジュールされ、転送終了後ジョブが実行される。このファイル転送中にリソース待ちの計算ジョブがあれば転送中のマシンで計算を実行する。転送終了すると計算ジョブはサスペンドされる。

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[8] を用いた。Jay は Condor のコアコンポーネントを 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にはレプリカのロケーション管理を行い、高速ファイル転送ツール Dolly+[9] が組み込まれたマルチレプリケーションフレームワーク[10] を利用した。

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。ファイルの複製要求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、データ転送サービスにより効率的に転送される。

4.1.1 転送元ホスト選択機構

転送元ホストの選択は RLS と Replica Selector の 2 つのコンポーネントで構成される。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や問合せなどを出す RLS クライアントで構成される。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存在するレプリカの各位置情

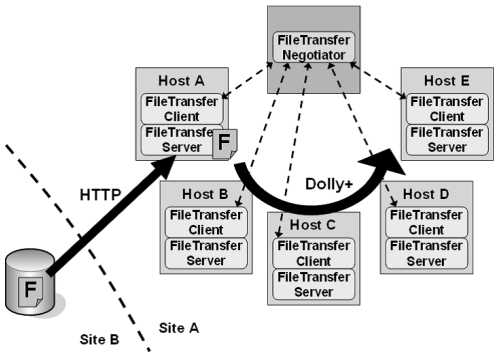

図 2 データ転送機構

報が登録される。Replica Selector は登録された位置情報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 値を採用している。

4.1.2 データ転送サービス

選択された転送元からのデータ転送は図 2 のように転送ネゴシエータ、転送サーバ、転送クライアントの 3 つのコンポーネントで構成されたデータ転送サービスにより行われる。サイト内の複数ホストの転送クライアンがサイト外にある同一レプリカを要求する際にはサイト内で転送ネゴシエータが要求を集約し、代表ノードの転送クライアントのみがそのレプリカを取得し、その後サイト内で転送サーバが Dolly+による高速な並列転送を行う。

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[11] を行い、ClassAd[12] 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。

4.3 ジョブ実行の流れ

ここで本システムがどのような流れでジョブ実行するか説明する (図 3)。Startd はマシン情報やレプリカ情報を記述した ClassAd を定期的にセントラルマネージャに発行する。レプリカ情報としては RLS サーバに登録されている各ファイルを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 は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。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。マッチしたマシンとジョブの ClassAd の発行マシンにスケジューリングされたことを通知する。通知が来ると Schedd は Startd にジョブ実行の可否を問い合わせる。可能であれば Schedd は Shadow プロセスを、Startd は Starter を起動する。Shadow と Starter 間で必要なデータが送信されジョブ実行が開始される。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データ転送サービスにより効率的な転送実行後、計算を開始する。

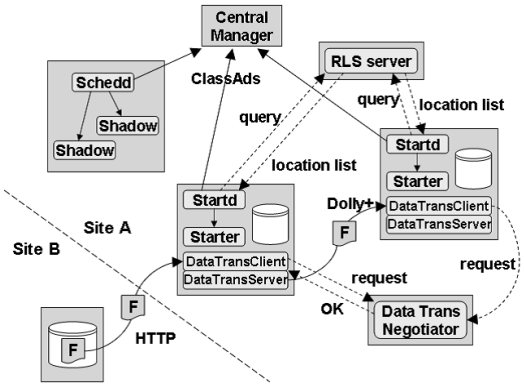

図 3 データインテンシブアプリケーション実行の流れ

4.4 データ転送と計算の同時実行

実行マシンは実行中のジョブが計算実行中であるか、データ転送中であるかをジョブ実行を監視し、ClassAd に記述して定期的にセントラルマネージャに送信する。セントラルマネージャは実行マシンがデータ転送中であればコンピュートインテンシブジョブを割り当て計算実行を、計算実行中であれば次のデータインテンシブジョブ実行のためのデータステージングを行う。ジョブがデータインテンシブであるかどうかは現在の実装ではジョブ実行に必要なデータサイズがあるサイズを超えているかどうかで判断している。この閾値のサイズはマシンの性能などにより異なるため各マシンごとに設定可能であり、マッチメイキングにより柔軟なスケジューリングが可能である。このようなデータ転送と計算実行のオーバーラップが行われているマシンでデータ転送が終了し、計算実行が開始されようとすると 2 つのジョブの計算が同時に実行されてしまう。そこで優先度の低い方のジョブのプロセスグループにシグナルを送信し、サスペンドを行う。

5. 評価実験

5.1 従来手法との比較

5.1.1 評価実験の概要

本システムの有効性を示すためデータインテンシブジョブを実行し、従来手法との比較を行った。サンプルアプリケーションとして相同性検索ツール BLAST[13] を利用し、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。実験には本研究室の PrestoIII クラスタ (CPU: Opteron 242, Memory: 2GB, OS: Linux 2.4.30, Network: 1000Base-T) を利用し、セントラルマネージャ、サブミットマシン、RLS サーバをそれぞれ 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 個の核酸配列をクエリとして約 3GB のデータベースに対して検索を行うというものである。実験ではデータベースを NFS で共有する手法、scp で毎回ステージングする方法、各実行マシンにすでに格納されている理想状態、提案手法の 4 つを比較した。

5.1.2 評価実験結果

実行マシン 16 ノードでファイル共有手法、ステージング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 4 に示した。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図 5 に示した (比較のため図 4 には共有手法のグラフ、図 5 には共有手法およびステージング手法のグラフが重ねて表示されている)。図 4, 5 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 6 に示した。図 6 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。

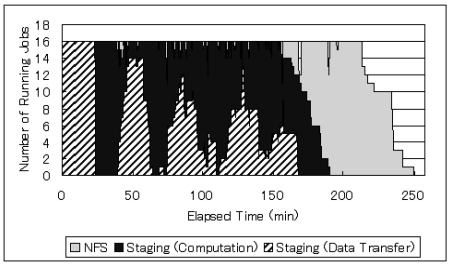

図 4 ステージング手法におけるジョブ数の推移

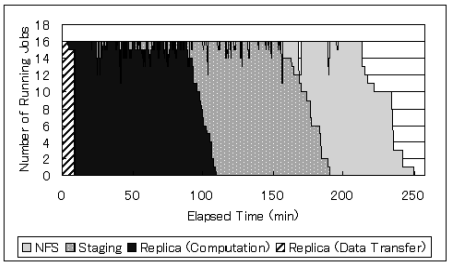

図 5 提案手法におけるジョブ数の推移

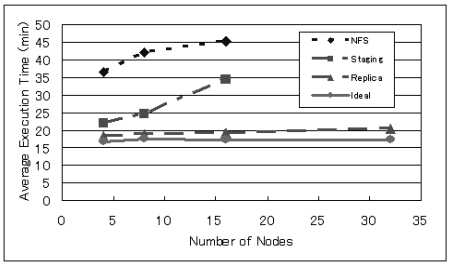

図 6 ジョブの平均実行時間

5.2 スループット向上評価

データインテンシブジョブとコンピュートジョブをサブミットし、データ転送と計算の同時実行によるスループット向上を確認する。実行マシンは 8 ノード用意し、約 3GB のデータベースファイルを毎回ステージングして BLAST 実行するジョブを 40 個サブミットした後にモンテカルロ法により円周率を計算するコンピュートジョブを 8 個サブミットした。同時実行しなかった場合とした場合の計算実行中のジョブ数の推移をそれぞれ図 7、8 に示す。前者ではデータ転送中の計算サイクルが利用されず全ジョブ終了に約 111 分かかっているのに対して後者ではデータ転送中の遊休サイクルを効率的に利用して約 88 分で全ジョブが終了し、スループットの向上が確認された。

6. おわりに

本研究ではレプリカ管理とジョブスケジューリング

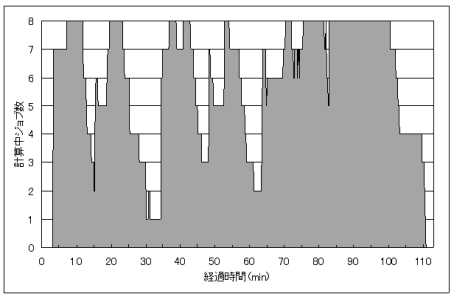

図 7 計算実行中のジョブ数の推移 (同時実行なし)

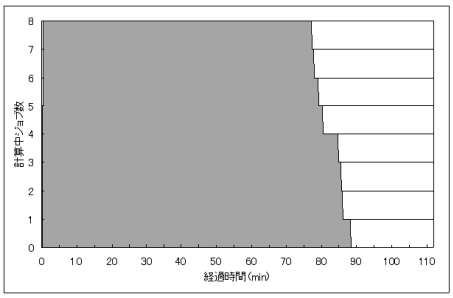

図 8 計算実行中のジョブ数の推移 (同時実行あり)

のタイトな結合による効率的なデータインテンシブジョブ実行環境を提案した。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を拡張し、複数ノードへの並列ファイル転送が可能なレプリカ管理システムと連動させた。またデータ転送中の計算リソースの有効利用を可能にした。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。またデータ転送と計算の同時実行によるスループットの向上も確認された。

今後の課題としては以下が挙げられる。現在は入力データが大規模な場合だけが想定されており、出力データについては考えられていない。これを考慮することでさらに大規模データを利用するワークフローの効率的な実行につながると考えられる。また今後さらに大規模な環境でさまざまなアプリケーションを用いた評価を行う必要がある。

参 考 文 献

1) Kosar, T. and Livny, M.: Stork: Making Data Placement a First Class Citizen in the Grid, *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS2004)*, Tokyo, Japan (2004).
2) : Directed Acyclic Graph Manager. http://www.cs.wisc.edu/condor/dagman.
3) : Condor Project Homepage. http://www.cs.wisc.edu/condor/.
4) Litzkow, M., Livny, M. and Mutka, M.: Condor - A Hunter of Idle Workstations, *Proceedings of the 8th International Conference of Distributed Computing Systems*, pp. 104–111 (1988).
5) Foster, I. and Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit, *Internatinal Journal of Supercomputer Applications*, Vol.11, No. 2, pp. 115–128 (1997).
6) Chervenak, A. L., Palavalli, N., Bharathi, S., Kesselman, C. and Schwartzkopf, R.: Performance and Scalability of a Replica Location Service, *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC-13)*, Honolulu, HI, pp. 182–191 (2004).
7) Project, T. G.: GridFTP: Universal Data Transfer for the Grid, White Paper (2000).
8) 町田悠哉, 滝澤真一朗, 中田秀基, 松岡聡: 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, ハイパフォーマンスコンピューティングと計算科学シンポジウム HPCS 2006 論文集, pp. 9–16 (2006).
9) Manabe, A.: Disk cloning program 'Dolly+' for system management of PC Linux cluster, *Proceedings of the Computing in High Energy and Nuclear Physics 2001 (CHEP'01)* (2001).
10) Takizawa, S., Takamiya, Y., Nakada, H. and Matsuoka, S.: A Scalable Multi-Replication Framework for Data Grid, *Proceedings of the 2005 International Symposium on Applications and the Internet (SAINT 2005 Workshops)*, Trento, Italy, pp. 310–315 (2005).
11) Raman, R., Livny, M. and Solomon, M.: Matchmaking: Distributed Resource Management for High Throughput, *Proceedings of the 7th IEEE International Synposium on High Performance Distributed Computing (HPDC-7)*, Chicago, IL (1998).
12) Livny, M., Basney, J., Raman, R. and Tannenbaum, T.: Mechanisms for High Throughput Computing, *SPEEDUP Journal*, Vol. 11 (1997).
13) : NCBI BLAST. http://www.ncbi.nlm.nih.gov/BLAST/.