Vol.2013-HPC-140 No.19
2013/8/1

データ配置を考慮したタスクスケジューリング

李 燮鳴[1,2,a)]　建部 修見[1,2]

概要：本稿では, データ配置を考慮したタスクスケジューリング手法を提案し, 評価を行う． これまでスーパコンピュータにおいて広く使われているファイルシステムは，計算ノードとは別に，ファイルシステムノードを集中して設置され, 特にデータ配置はタスクスケジューリングには影響しなかった．しかしながら, エクサスケールのスーパコンピュータにおいては, これまでの集中配置されたファイルシステムでは必要なバンド幅を達成することが難しいため, 計算ノード側に I/O ノードを分散配置させたスケールアウト型のファイルシスムアーキテクチャが有望となっている．このとき, データ配置はアクセス性能に著しい影響を与えるため, タスクを計算ノードに割り当てる際, データ配置を配慮すべきである．本研究では, 複製作成手法とデータ配置を考慮したタスク割り当てから成るタスクスケジューリング手法を提案し,Torque で実装を行った．その結果，Gfarm ファイルシステム上にあるファイルをアクセスするタスクのを用いた評価では従来の 3 倍の読み込み性能の向上を得ることができた．

キーワード：データ配置, タスクスケジューラ, 並列ファイルシステム, Torque, Gfarm

1. はじめに

近年, 自然現象の高解像度シュミレーション, 大規模画像解析, 気候モデリングなどの科学技術計算分野における解析データの大規模化が進んでいる．今後解析データサイズは exabyte から zettabyte スケールのデータを扱う事が予想される．

これまでの大規模計算環境で広く使われているファイルシステムとしては,GPFS[1], Lustre[2], PVFS[3] などが挙げられる．これらのファイルシステムは計算ノードと別にストレージを想定して設計している．計算ノードとストレージノードの間ではストレージエリアネットワーク (SAN) で連結されている．各計算ノードからファイルシステム上にあるファイルにアクセスする際はいずれも計算ノード, ネットワーク, ファイルノードのアクセスパターンとなり, 各計算ノードからファイルシステムノードへのアクセス性能は均等である．

均一アクセス型ファイルシステムは, 全体のノード数に対するファイルシステムノード数の割合やファイルシステムと計算ノードを繋げるストレージエリアネットワークの性能の問題によりスケールアウトするアーキテクチャとなっていない．次世代のエクサバイトやゼタバイト規模のデータを扱うにあたって十分な性能を得られない可能性がある．一方, 計算ノードとストレージノードを束ねるファイルシステムが提案されていて, 代表的なのは GFS[4], HDFS[5], Gfarm．[6] などがある．Gfarm は各計算ノードにあるローカルストレージを束ねて一つのネームスペースで管理することを可能にした．このようなアーキテクチャでは, 各計算ノードが自分のローカルにあるストレージにアクセスするため, より優れたスケーラビリティを持つと考えられている．しかし, このようなシステムでは, 計算ノードがローカルストレージとリモート (他の計算ノード) ストレージにあるファイルへアクセスする時の性能は異なるため, 非均一なアクセスとなる．

また, 大規模計算環境では, 単一ジョブがすべての計算資源を占有することは稀であり, ほんとんどの場合は, 複数のジョブが並列に実行される．その場合, ジョブを統一管理して，ジョブの競合による性能低下を防ぐため, タスクスケジューリングが必要となる．均一アクセス型のファイルシステムでは, 各ファイルへのアクセス性能はどのノードからも均等であるため, タスクスケジューリングを行う際はデータ配置を配慮する必要がなかった．しかし, 非均一アクセスのファイルシステムではファイルへのアクセス効率はファイルの位置がローカルあるいはリモートのストレージにあることで大きく異なる．従ってタスクスケジューリングをする際はデータ配置を考える必要がある．本稿では

[1] 筑波大学大学院　システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
[2] 独立行政法人科学技術振興機構 CREST
[a)] risyomei@hpcs.cs.tsukuba.ac.jp

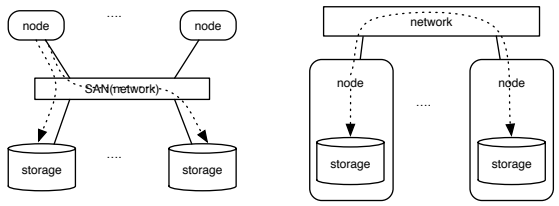

図 1: 均一と非均一アクセス型ファイルシステム．左図の均一アクセス型ファイルシステムではどのファイルノードへのアクセスもアクセス遅延は同等である．右図の非均一アクセス型ファイルシステムはローカルアクセスとリモートアクセスで性能が大きく異なる

このような非均一型アクセスのファイルシステムのためのデータ配置を配慮したタスクスケジューリング手法を提案する．

本稿の構成は以下のようになっている．まず 2 章は技術背景として Torque と Gfarm を紹介する．次の 3 章では提案手法のスケジューリング手法を紹介し,4 章で性能評価の結果をしめす．そして，5 章は関連研究を述べ，6 章でまとめと今後の課題を述べる．

2. Gfarm ファイルシステムと Torque タスクスケジューラ

本研究では,torque[7] をベースにしてタスクスケジューラを実装した．また, スケジュールする際に Gfarm の特性を利用するため, この章でではまず，torque と Gfarm とについて, 簡単に紹介する．

2.1 Torque タスクスケジューラ

Torque は PBS に基づいて作られたオープンソースのタスクスケジューラである．Torque は**図 2** で示されたように主に pbs_server,pbs_sched,pbs_mom の三つのコンポネントから成り立つ．pbs_server はシステム全体とユーザーのインターフェイスであり, タスクのエンキュや修正, スケジューラ指令の発信,pbs_mom にタスク実行の指令の発信などを行う．pbs_sched は Torque のスケジューラであり,pbs_server からタスクの情報を受け取り,pbs_mom と通信して得た計算ノードの状態を利用してスケジューリングを行う．

タスクはまずクライアントによって pbs_server に送られ, そして,pbs_server が pbs_sched タスク待ち行列にエンキュー (図 2 の 1*) する．そして,pbs_sched はスケジューリングポリシでタスクをソートして, 負荷の低いノードにある pbs_mom に割り当てられる (図 2 の 2*).

今回の研究ではデータの配置を配慮するため，予めデータの情報を得る必要がある．Torque では, タスクが使用するファイルを指定する機能がなかったため, は二次開発を行い, タスクを提出するコマンドのパラメータで-g $f1 \cdots fn$, またはタスクのスクリプトの中に#PBS -g $f1 \cdots fn$ のディレクティブを加えることファイルを指定するを可能した．

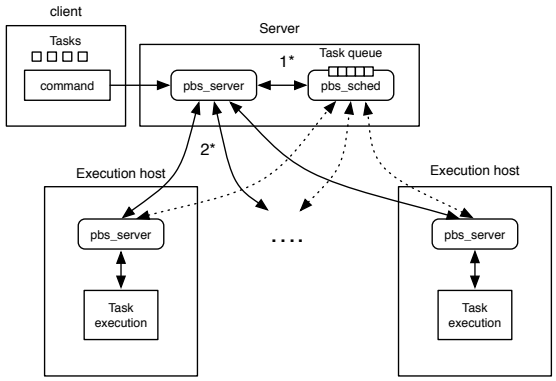

図 2: Torque タスクスケジューラ

2.2 Gfarm ファイルシステム

Gfarm ファイルシステムはオープンソースで開発を進めているファイルシステムである．Gfarm は多数の計算ノードのローカルストレージを束ねて, 一つのネームスペースで管理することを可能にしている．Gfarm は一つのメタデータサーバー (gfmd), と複数のファイルシステムサーバー (gfsd) から成り立っている．gfmd はファイルの属性, ディレクトリ構造やファイルの複製を管理している．gfsd は計算ノードに配置され, 計算ノードのローカルストレージと他の計算ノード gfsd を介してリモートストレージにアクセスを行う．

Gfarm では, ファイルの分割を行っておらず,1 つのファイルが複数のファイルシステムノー ドにまたがって存在することはないが, ファイルの複製を多数のノードに作成し, スケーラブルな性能を得ることができる．

3. データ配置を配慮したタスクスケジューリング

Gfarm 上の計算ノードはネットワークを通じてリモートストレージのファイルをアクセスすることが可能である．しかし, これはローカルストレージへのアクセスに比べると非効率なので, 可能な限りローカルストレージを使った方が効率が良い．従来のタスクスケジューラはデータ配置を考慮しないため,Gfarm のようなデータ配置でアクセス性能が大きく異なるファイルシステムでは, データと実行プログラムの局所性を利用することがでない．この章で提案される二つの手法はいずれも可能な限り Gfarm のローカルストレージを利用する．

3.1 データ配置を配慮したタスク割り当て

タスクに計算ノードのローカルストレージを使用させるためにはタスクがアクセスするファイルのあるノードにタスクを割り当てる必要がある．この手法では, 図 2 の手順

2*のタスクを計算ノードに割り当てる選択を行うとき, タスクがアクセスするファイルの中で, 計算ノード上に存在するファイルと存在しないファイルをそれぞれ統計し, この計算ノードのファイル局所スコア ($fileLocality$) を計算する．そして, このスコアを CPU の負荷レベルと統合し, 総合的なスコア ($Score$) を得る．最後, スコアが一番低いノードを選び，タスクを割り当れる．

この過程を正式に述べるため, タスクを $t_1 \cdots t_m$, 計算ノードを $h_1 \cdots h_n$ と定義し, そしてタスク t_x がアクセスするファイルを $f_{x1} \cdots f_{xq}$ と定義し, その内のファイル f_{xn} の複製が計算ノード h_y に存在することを $on(f_{xn}, h_y)$ と定義する．また, タスク t_i を h_j に割り当てることを $alloc(t_i, h_j)$ と表す.

従って, タスク t_i をデータ配置を配慮したタスク割り当てをする事を式で表すと

$$\arg\min_x Score(alloc(t_i, h_x)) \tag{1}$$

となり, そのうち

$$\begin{aligned} Score(alloc(t_i, h_x)) =& fileLocality(t_i, h_x) \times \beta \\ &+ load_{h_x} \times (1-\beta) Node (0 \le \beta \le 1) \\ fileLocality(t_i, h_x) =& [\sum_{y=1}^{n} locality(f_{iy}, h_x) / \sum_{y=1}^{n} f_{iy} + 1]/2 \\ locality(x, y) =& \begin{cases} sizeof(x) & \text{if } on(x, y) \\ -sizeof(x) & \text{other} \end{cases} \end{aligned} \tag{2}$$

式 2 の内,$locality(x, y)$ の中の $sizeof(x)$ は, ファイル x のサイズ (バイト) を示し,$load_{h_x}$ は計算ノード h_x の CPU 負荷を示している．Torque の負荷 $load_{torque}$ の範囲は 0 から CPU 数までとなるので, ここで使う $load$ は $load_{torque}/CPUs$ となる． $fileLocaliy$ の計算でプラス 1 割 2 をしたのは,$fileLocality$ を $[0, 1]$ の範囲に押さえるためである.

また, この式の中の β はデータ配置がスケジューラの判断に影響する程度を表すものであり,β を 1 に設定すると, スケジューラはノードの負荷である $load$ を完全に無視する．一方,β を 0 に設定すると, スケジューラはファイル配置を無視し, 一般なスケジューラとなる.

3.2 複製作成

分散型ファイルシステムにおいて, 頻繁にアクセスされるファイルがあるノードに集中するとホットスポットが生成し, ネットワークやディスクアクセスのリクエストが増加し, 性能が著しく劣化する．また,3.1 で提案した手法を使うと, 頻繁に使用されるファイルが集中的配置されているノードにタスクが集中的割り当てられ, この現象を起こす傾向がある．複製作成手法の目的は頻繁に使われるファイルのレプリカを作成することにより特定計算ノードへのアクセス負担を複数のノードに分散し, 並列なアクセスを可能にすることである.

ここで重要なのは「頻繁」と判断する基準である．この研究では頻繁を以下のように定義する：

$$\begin{aligned} fileUsedCount(f) > replicaCount(f) \times \alpha \\ (\alpha = 0, 1, \ldots, N) \end{aligned} \tag{3}$$

この式の中 $fileUsedCount(f)$ はファイル f がすべてのタスクによってアクセスされる回数であり,$replicaCount(f)$ はファイル f が Gfarm ファイルシステムで複製されている数である．α はレプリカの作成強度をコントロールする変数であり, 厳密ではありませんが, 一つのレプリカに同時にアクセスできる数を示している.

ここで提案した手法では, タスクが pbs_sched のタスク待ち行列に入った時, 待ち行列の中にあるすべてのタスクがアクセスするファイルをスキャンし, アクセスされる全部のファイルの集合 F を生成する．そして,F 中の各ファイル $f \in F$ に使用される回数 $fileUsedCount(f)$ を統計する, また,Gfarm へ複製の数 $replicaCount(f)$ を問い合わせる．もし, あるファイル f に対して, 式 3 で示された関係を満足する際, それを成立させないように f の複製を $fileused(f)/\alpha - replicaCount(f)$ 個作成する.

表 1: 性能評価環境

Hardware	
CPU	Intel Xeon E5-2665 (2．4GHz 20M 8Core) x2
HDD	146GB (6Gbps SAS 15,000rpm) x4
メモリ	64GB (8GB 1600MHz) x8
Software	
Gfarm	2.5.8
Torque	2.5.12

表 2: ノードのソフトウェア設定

ノード	Gfarm	Torque
Chris21−24	gfsd	pbs_mom
Chris25	gfmd	pbs_sched,pbs_server

4. 性能評価

本章ではデータ配置を配慮したタスクスケジューリングの評価結果について述べる．評価は Gfarm 上にある 512MB のファイルを読み込むタスクを用いる.

性能評価は筑波大学のクラスタである Chris20-24 の五ノードを用いて行った．Chris20 から 25 の仕様と Gfarm と Torque の設定を環境をそれぞれ**表 1** と**表 2** に示した,

4.1 データ配置を配慮したタスク割り当ての評価

ここで評価に使ったタスクは Gfarm 上にある異なる 20

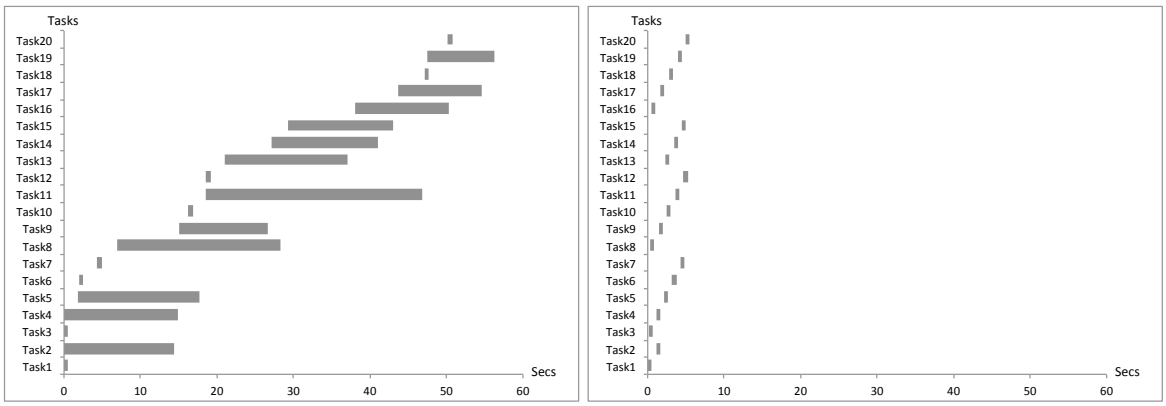

図 3: データ配置を配慮したタスク割り当ての評価結果．左図はオリジナルの Torque，右図は提案手法でタスクを割り当た場合の結果である．x 軸は経過時間であり，それぞれのタスクに開始時間から終了時間までを表してる

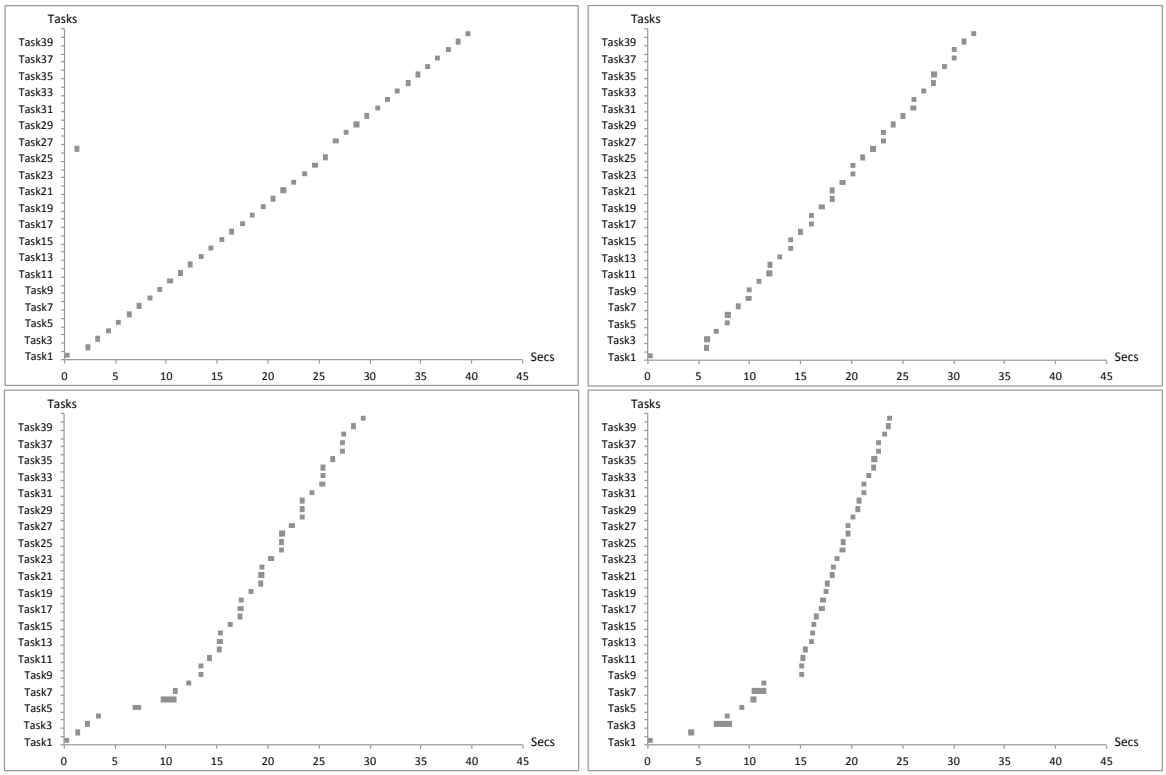

図 4: 複製作成の評価結果: 左上図は $\alpha = 20$，右上図は $\alpha = 15$，左下図は $\alpha = 10$，右下図は $\alpha = 5$ 時の結果を示している．x 軸は経過時間で，それぞれのタスクの開始時間から終了時間までを表してる

個の 512MB のファイルを読み込むタスクで，ファイルは各ノード 10 個ずつ均等に分布している．

評価の結果は**図 3** に示す．図の x 軸は経過した時間であり，バーが短いほど実行時間が短く，性能が高い．左の図は Torque のストックのスケジューラでは，比較的短いバーと長いバーがあるが，短いバーは偶然タスクが適切なノードに割り当てられた場合である．それに比べて，右の提案手法ではすべてのタスクがアクセスするファイルのあるノードに割り当てられているため，バーが短い．また，すべてタスクが完成する時間も，われわれが提案手法の法が圧倒的に短い．

4.2 複製作成の評価

われわれが提案する手法の複製作成の目的は，頻繁にアクセスされるファイルが一つまたは少数のノードに集中している際，複製を作成することにより，並列アクセスを可能にするからである．私たちは 40 個同じファイルをアクセスをするタスクを用いて評価を行い，評価はこのファイルの複製が一つしか存在しな状態で始まる．また，この評価では 3.2 で紹介した α を異なる値に設定し，性能の違いを得た．

図 4 に示されたのは，それぞれ α 値を 20, 15, 10 ,5 と設定したケースである．また，評価では，3.1 の手法も使

用している．3.2 ではアクセスするファイルがスキャンされるタスクは，待ち行列に入れられるタスクであるため，α が 20 の時必要な複製数は $(40-4) \div 20 = 1.8 < 2$ であり，つまり，複製を作らないことにる．また，α が 15,10,5 の時に必要な複製の数はそれぞれ 2,3,4 となる．図 4 の中で，α が 20 と 15 の時はタスクが割り当てられる速度ほぼ均一であるが，15 の時は 20 の時より効率が高い．そして，α が 10 と 5 の時は複製作成はタスクの実行を影響し，評価の初期では効率が低いが，一旦複製が作成されると効率が著しくあがるのが分かる．

5. 関連研究

5.1 複製作成が計算プロセスに対する影響

[8] では，プロセススケジューリングが I/O 処理が計算に与える影響，余剰コアの影響について評価を行った．評価の結果により，ファイルアクセスを大量に行っている計算ノードに対して直接ファイルアクセスとファイル複製作成はアクセス性能が最大で 50 %ほど低下するのが明らかになった．また，直接アクセスと複製作成では，性能に与える影響の違いは５%以下であった．

5.2 LSF plug-in による gfarm でのデータ配置を配慮したタスクスケジューリング

[9] では複製を作成する手法に提案した．彼らの研究では複製を作成判断基準はタスクの優先度 $p(t_i)\times$ タスクの予測実行時間 $T(t_i)$ である．これは本研究で用いる $fileUsedCount(f) \leq replicaCount(f) \times \alpha$ とは大きく異なる．

また，[10] はサブミッションされる batch job をタスク数が少ない few-batch-mode-job (以下 FBMJ) とタスク数が多い mass-batch-mode-job (以下 MBMJ) に分類し，FBMJ では複製を作成せずに直接タスクスケジューリングを行い，MBMJ では負荷が低いノードに複製を作成した後スケジューリングする．

5.3 Stork

Stork[11][12] では，広域グリッドファームでのデータ移動作業を計算タスクのように扱い，データ移動作業を待ち行列にエンキュすることや，スケジューリング，監視，チェックポイントを可能にした．しかし，Stork はデータ移動専門のスケジューラで，計算タスクを扱わない，この点で本研究と大きく異なる．

6. 結論と今後の課題

提案手法のデータ配置を配慮したタスク割り当てでは，Gfarm ファイルシステムに置いて，従来のタスクスケジューラに比べ，タスク全体の実行時間を約 1/10 まで短縮することができ，タスク平均ファイルアクセス速度は従来の三倍以上に上げた．また，複製作成の手法でも最大従来の約二倍の性能へと向上した．

しかし，今回の研究で，タスクがアクセスするファイルがノード上に存在するかだけでそのノードがふさわしいと判断するのは不十分であると感じた．従って，次は計算ノードのアクセス負荷も統計するように研究をすすめたい．また，この研究で評価に使われたのはシンプルタスクである．これからは，MPI-IO などの並列タスクのスケジューリングについても研究すべきである．

参考文献

[1] Frank B Schmuck and Roger L Haskin. Gpfs: A shared-disk file system for large computing clusters. In *FAST*, volume 2, page 19, 2002.
[2] Philip Schwan. Lustre: Building a file system for 1000-node clusters. In *Proceedings of the 2003 Linux Symposium*, volume 2003, 2003.
[3] Robert B Ross, Rajeev Thakur, et al. Pvfs: A parallel file system for linux clusters. In *in Proceedings of the 4th Annual Linux Showcase and Conference*, pages 391–430, 2000.
[4] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.
[5] Dhruba Borthakur. Hdfs architecture guide. *Hadoop Apache Project. http://hadoop. apache. org/common/docs/current/hdfs_design. pdf*, 2008.
[6] Osamu Tatebe, Noriyuki Soda, Youhei Morita, Satoshi Matsuoka, and Satoshi Sekiguchi. Gfarm v2: A grid file system that supports high-performance distributed and parallel data computing. In *Proceedings of the 2004 Computing in High Energy and Nuclear Physics*, 2004.
[7] Adaptive Computing. Adaptive computing.
[8] 木村浩希 and 建部修見. Mpi-io/gfarm におけるデータ配置を考慮したプロセススケジューリングの検討. **情報処理学会研究報告**.*[***ハイパフォーマンスコンピューティング***]*, 2011(33):1–7, 2011.
[9] Xiaohui Wei, Wilfred W Li, Osamu Tatebe, Gaochao Xu, Liang Hu, and Jiubin Ju. Implementing data aware scheduling in gfarm (r) using lsf (tm) scheduler plugin mechanism. *GCA*, 5:3–5, 2005.
[10] Yonghong Yan and Barbara Chapman. Comparative study of distributed resource management systems–sge, lsf, pbs pro, and loadleveler. Technical report, Technical Report-Citeseerx, 2008.
[11] Peter Couvares, Tevfik Kosar, Alain Roy, Jeff Weber, and Kent Wenger. Workflow management in condor. In *Workflows for e-Science*, pages 357–375. Springer, 2007.
[12] Tevfik Kosar and Miron Livny. Stork: Making data placement a first class citizen in the grid. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 342–349. IEEE, 2004.

正誤表を以下に示します。訂正してお詫び申し上げます

	(誤)	(正)
式(2)	$Score(alloc(t_i, h_x)) = fileLocality(t_i, h_x) \times \beta + load_{h_x} \times (1 - \beta) Node$	$Score(alloc(t_i, h_x)) = fileLocality(t_i, h_x) \times \beta + load_{h_x} \times (1 - \beta)$
図(2)