社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－HPC－107 (10)
2006／7／31

グローバルスケジューリングのためのローカル計算資源管理機構

中田秀基[†] 竹房あつ子[†] 岸本誠[†,††]
大久保克彦[†,†††] 工藤知宏[†]
田中良夫[†] 関口智嗣[†]

グリッド上で資源の同時確保を実現する方法のひとつとして事前予約を行う方法がある．複数資源の同時予約操作は本質的に分散トランザクションであり，操作失敗時の挙動を保障するためには，スケジューラとローカル資源管理機構の間で適切なプロトコルを用いる必要がある．そのためには，ローカル資源管理機構のインターフェイスとローカル資源管理機構内部の予約管理機構が，そのプロトコルに対応しなければならない．われわれは，プロトコルとして分散トランザクションで一般に用いられる2相コミットプロトコルを採用し，これを可能にするべく，計算資源上のローカル資源管理機構の予約インターフェイスを設計・実装した．予約インターフェイスはWSRF(Web Services Resource Framework)を基盤プロトコルとし，Globus Toolkit 4を用いて実装されている．さらにローカル資源管理機構内部で使用する予約管理機構として，TORQUEおよびGridEngineと協調動作することのできるPluS予約管理機構を改良し，2相コミットの機構を組み込んだ．

A Local Computation Resource Manager to enable Global Scheduling

Hidemoto Nakada ,[†] Atsuko Takefusa ,[†]
Makoto Kishimoto,[†,††] Katsuhiko Ookubo,[†,†††]
Tomohiro Kudoh ,[†] Yoshio Tanaka [†] and Satoshi Sekiguchi [†]

Advance reservation is one viable way to enable co-allocation of several resources on the Grid. Co-allocation with advance reservations is essentially a distributed transaction, where it is crucial to have proper commit protocol to guarantee stable behavior. We designed and implemented reservation interface with 2-phased commit protocol, which is often used in the distributed transaction area. The interface is based on the WSRF (Web Services Resource Framework), one of the standard technologies in Grid, and implemented with Globus Toolkit 4. We also implemented backend local resource management module for computational resources, called 'PluS'. PluS is capable of handling 2-phased commit protocol and provides advance reservation capability for TORQUE and GridEngine.

1. はじめに

グリッドの目的の一つとして，ネットワーク上に散在する資源を同時に確保，使用する大規模な計算がある[1]．資源の同時確保を実現するにはいくつかの方法が考えられるが，もっとも直接的な方法は事前予約を行う方法である．各資源上のローカル資源管理機構が事前予約機構を持つことを前提にし，全体を統合するスーパスケジューラから，同時刻に事前予約を行うことで，その時刻にすべての資源を同時に確保できることを保証する[2),3)]．

この際に留意するべき点の一つとして，スーパスケジューラとローカル資源管理機構間のプロトコルがある．複数資源に対する同時予約操作は，本質的に分散トランザクションであり，操作失敗時の挙動を保障するためには，スーパスケジューラとローカル資源管理機構の間で適切なプロトコルを用いる必要がある．このためには，ローカル資源管理機構のインターフェイスと，ローカル資源管理機構内部の予約管理機構がそのプロトコルに対応する必要がある．われわれは，プロトコルとして分散トランザクションで一般に用いられる2相コミットプロトコル[4)]を採用し，これに対応した計算資源上のローカル資源管理機構の予約インターフェイスを設計・実装した．この予約インターフェイスはWSRF(Web Services Resource Framework)[5)]を基盤プロトコルとし，Globus Toolkit 4[6)]を用いて実装されている．

このプロトコルを使用するためには，ローカル資源管理機構内部で用いられる予約管理機構においても，2

† 産業技術総合研究所 National Institute of Advanced Industrial Science and Technology (AIST)
†† エス・エフ・シー S.F.C Co., Ltd.
††† 数理技研 SURIGIKEN Co., Ltd.

相コミットがサポートされていなければならない．われわれは，計算資源の予約管理機構として，PluS 予約管理機構[7]を改良し，2相コミットの機構を組み込んだ．PluS 予約管理機構は，キューイングスケジューリングシステムの TORQUE および GridEngine と協調動作し，2相コミット機構つきの事前予約機能を提供する．

本稿の以下の構成を以下に示す．2節で同時予約とコミットプロトコルの関係を議論する．3節で予約インターフェイスの設計と実装について述べる．4節では，PluS 資源予約機構について述べる．5節でコミットプロトコルの選択に関する議論を行う．6節はまとめである．

2. 同時予約と2相コミットプロトコル

2.1 同時予約の問題点

複数資源の予約においてコミットプロトコルが必要であることを示すために，複数の資源 (A,B) で予約した時間帯を変更することを考えてみよう．単純な実装では，資源 A および B に対して順番に時間帯変更のリクエストを出すことになるだろう．この場合，資源 A で時間帯の変更に成功したのち，資源 B で変更に失敗した場合に，問題が生じる可能性がある．期待される動作は，資源 A での変更をキャンセルし，予約を一旦もとの時間帯に戻した上で，変更の失敗を上位レイヤに返すことである．

しかし，資源 A ではすでに予約時間帯が変更されているため，変更をキャンセルする際には，再度時間帯を変更しなければならない．しかし，変更時に一旦開放した資源をキャンセル時に再度取得できる保証はない．このため，最悪の場合，時間帯の変更に失敗しただけでなく，もともと保持していた時間帯も維持できないことになってしまう．

2.2 2相コミット

複数資源の同時予約は，本質的に分散トランザクションである．分散トランザクションに関しては長い研究の歴史があり[4]，いくつものプロトコルが提案されている．そのもっとも基本的なプロトコルが2相コミットである．ここでは，文献[4]にならって，2相コミットを紹介する．

コミットプロトコルには Coordinator と Cohort の2種類の参加者がある．Coordinator がプロトコルを始動する参加者で1つだけ存在する．Cohort は一般に複数となる．プロトコルの目的は，1つの Coordinator と複数の Cohort がすべて，アボートかコミットかのいずれかの状態に至ることである．

資源予約の観点でいえば，Coordinator が上位のスーパスケジューラ，Cohort が下位のローカルスケジューラである．目的は，ある予約操作が成功したか失敗したかを，すべてのスケジューラが共有すること

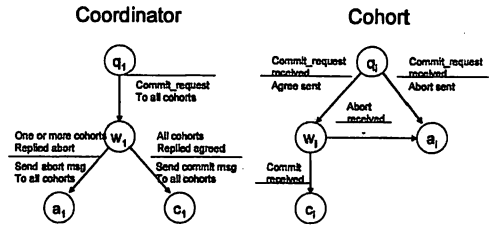

図1 2相コミットプロトコル

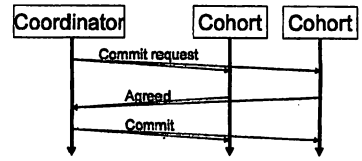

図2 2相コミットのメッセージダイアグラム

である．

2相コミットでは Coordinator と Cohort はそれぞれ4つの状態 (q,w,a,c) をとる．双方とも q からスタートし，a(abort) もしくは c(commit) 状態に至る．coordinator はまず，すべての cohort に commit request を送信する．commit request を受信した cohort は，内部の状態に従ってリクエストに対する返信を決定し，agree もしくは abort のメッセージを送信する．coordinator はすべての cohort から agree を受信した場合に限って commit を行い，その他の場合は abort を行う．図1に Coordinator，Cohort の状態遷移図を，図2に commit 成立時のメッセージダイアグラムを示す．

2.3 2相コミットによる同時予約．

2相コミットを用いることで，上述の問題が解決することを見てみよう．複数の資源 (A,B) で予約した時間帯を変更することを考える．資源 A および B に対して順番に時間帯変更のリクエストを出すのは素朴な方法と同じであるが，この際に資源の側では直ちに時間帯の変更を確定せず，w 状態に入り，agree メッセージをスーパスケジューラに返す．この場合，変更前に確保していた部分資源をリリースしないことに注意してほしい．

資源 A，B の双方から agree メッセージを受け取ると，スーパスケジューラは資源 A, B に commit メッセージを発行する．資源はこのメッセージをうけて初めて，予約の変更を確定し，変更前に確保していた部分資源を開放する．

資源 B で予約の変更に失敗した場合，資源 B はスーパスケジューラに abort メッセージを送る．スーパスケジューラはこれを受けて，資源 A に abort メッセージを送る．資源 A は，これを受けて予約の変更をキャンセルし，変更前の状態に戻る．

このように，2相コミットプロトコルを用いること

で同時予約を安全に行うことができる．

3. 予約インターフェイスの設計

2相コミットをサポートしたローカル計算資源管理機構を実現するために，WSRF をベースプロトコルとして使用したインターフェイスを設計した．WSRF はグリッドで標準的に用いられる規格の一つで，Web サービスをベースに，リソースとよばれるサービスに関連付けられた「状態」を実現する．一般的なオブジェクト指向言語のアナロジーで述べれば，サービスがオブジェクトインスタンスのメソッド定義に，リソースがデータ部に相当する．このサービスとリソースのペアを EPR (End point Reference) と呼ばれる構造で参照する．リソースに収められた情報は，サービスに対して getResourceProperty メッセージを送ることで取り出すことができる．

3.1 設計の指針

インターフェイスの設計においては以下の指針をおいた．

- **通知の不使用**
  一般に Web サービスではクライアントとサーバが区別され，両者の通信はかならずクライアントが主導するが，WSRF では Notification と呼ばれる通知の枠組みがあり，双方から同じように通信を行うことが可能である．
  われわれはあえてこれを用いないことを選択した．これは，Notification を用いると，クライアント側もネットワークの到達性を持たなければならず，運用の自由度が低下するため，および Notification は到達が保障されないのでポーリングなどの手法と併用しなければならず実装が煩雑になるため，である．
  本インターフェイスでは，クライアント側からのポーリングですべての情報通知を行う．
- **コミット状態の別サービス化**
  2相コミットの過程はひとつのトランザクションである．これを明示的に扱うために，個々のコミットの過程を予約とは独立した別のサービスとした．これによって，複数の主体が同時に単一の予約に対して予約操作を行っても混乱が生じることがない．

3.2 予約インターフェイスの概要

予約インターフェイスは次の3種のサービスから構成される．

- **ReservationFactoryService**
  ReservationService に対応する ReservationResource を生成する．対応するリソースはない．
- **ReservationService**
  対応するリソース ReservationResource とともに予約とその操作を実現する．予約の状態として，created, reserved, activated, released, failed の4つの状態をとる．
- **CommandStatus**
  対応するリソース CommandStatusResource とともに，2相コミットを実現するための，コミット状態 (コマンドの処理状態) を表現する．2.2 で示した，(q,w,a,c) に対応する initial, prepared, aborted, committed の4つの状態をとる．

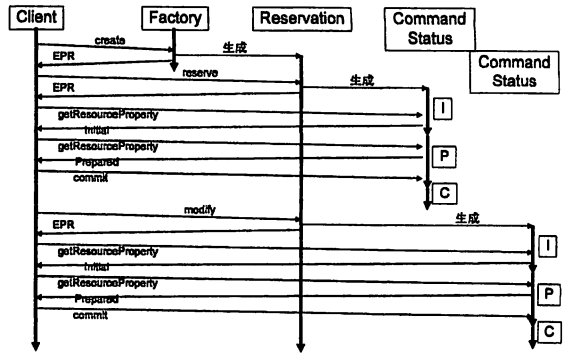

図3 予約インターフェイスのメッセージダイアグラム

表1に，各サービスのオペレーションを示す．ReservationService に対する各オペレーションは CommandStatus の EPR を返す．クライアントは処理を確定するために，CommandStatus に対して commit を行う必要がある．

図3に予約インターフェイスのメッセージダイアグラムを示す．同時予約を行う場合には，スーパスケジューラが左端の Client となる．Client はまず Factory に create メッセージを送り，ReservationServiceProperty を作成し，その EPR を受けとる．

次に Client は，その EPR に対して reserve メッセージを送る．すると，CommandStatus が生成され，その EPR が返却される．Client は，この EPR を用いて CommandStatus の状態をポーリングし，waiting になったことを確認してから，commit を行う．

図3では，予約に引き続き modify 操作を行っている．基本的に同じことを行っているが，reserve の際と modify の際で，別の CommandStatus のインスタンスを用いていることに注意されたい．CommandStatus は ReservationService に対して発行されるメソッド呼び出しに対して，毎回作成される一時的な存在なのである．

4. PluS 予約管理機構

資源管理機構として2相コミットをサポートするためには，バックエンドとなるローカルなスケジューラが2相コミットをサポートする必要がある．われわれは，これを実現するために PluS 予約管理機構 [7] を改

表 1　予約関連サービスオペレーション

オペレーション名	機能	入力	出力
ReservationFactoryService			
create	ReservationResource を生成する	なし	ReservationService EPR
ReservationService			
reserve	予約を行う	予約期間要求，資源要求	CommandStatus の EPR
modify	予約を変更する	予約期間要求，資源要求	CommandStatus の EPR
release	予約した資源を解放する	なし	CommandStatus の EPR
CommandStatus			
commit	コミットする	なし	なし
abort	アボートする	なし	なし

良した．

PluS 予約管理機構の実体は Java で記述したモジュールであり，これが既存のローカルキューイングシステムと連動することで，予約機能を実現する．

4.1　2 相コミットへの対応

2 相コミットに対応するために，PluS を拡張した．表 2 に，コマンドラインインターフェイスを示す．基本的な予約コマンドの構成は以前と同じであるが，予約操作コマンドには，2 相コミットを示す -T オプションを設けた．このオプションをつけた場合，予約操作は waiting 状態となるので，新たに設けた`plus_commit` もしくは`plus_abort` を用いてコミット，アボートを行わなければ操作が完了しない．

4.2　PluS の GridEngine 対応

PluS は，従来 OpenPBS の亜種のひとつである TORQUE[8] を対象としていたが，使用対象を広げるべく Sun の GridEngine[9] への対応をおこなった．両者に対する実装方式は大きく異なる．

TORQUE に対しては，TORQUE が提供するオリジナルのスケジューラモジュールを，PluS のモジュールで完全に置き換えている．これに対して，GridEngine に対する実装では，スケジューラとしては既存のものをそのまま維持し，PluS モジュールが外部からキューを制御するだけで予約機能を実現している．

これが可能なのは，GridEngine のキューでは，そのキューに投入されたジョブを実行するノードを限定することが可能なためである．TORQUE にもキューが存在するが，ノードに対する割り当てができないため，この実装法をとることはできない．

スケジューラを置き換えずに，外部モジュールとして予約機構を実装する方法には，1) スケジューラとして既存のものを使用しているため実装のコストが低い，2) 予約を必要としないジョブの挙動が元のスケジューラと完全に同一 3) 予約がキューとして実現されるので，既存のシステムとの連携が容易といったメリットがある．GridEngine 対応時の PluS スケジューラの構造を図 4 に示す．

GridEngine における予約機構の動作を示す．

(1)　予約のリクエストを受けると，サスペンド状態のキューを作成し，そのキュー名を予約 ID として返却する．

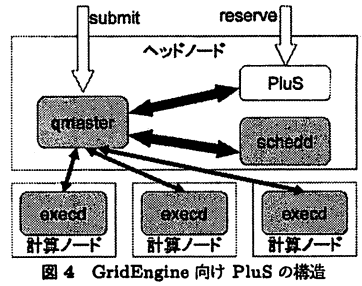

図 4　GridEngine 向け PluS の構造

(2)　ユーザはキューに対してジョブをサブミットする．予約時間が到来するまでは，キューがサスペンド状態であるため，ジョブは実行されない．

(3)　予約開始時間が来たら，予約に対応したキューを再活性化する．この際に他のキューが予約対象ノードを使用することがないよう，他のキューのノード情報も操作する．投入されていたジョブが走り出す．
この際に，当該ノードで実行中のジョブがあった場合には，PluS は，そのジョブを一旦サスペンドし，qresub コマンドを用いて当該ジョブのコピーを再投入し，もとのジョブを削除する．これによって，ジョブは他のノードで，最初から再実行されることになる．

(4)　予約終了時間が来たら 予約に対応したキューを削除する．同時に，2 で操作した他のキューに対して，当該ノードを再び使用するように再設定する．

5.　コミットプロトコルに関する議論

コミットプロトコルとしては，2 相コミットのほかにもいくつものプロトコルが提案されている．

5.1　3 相コミット

2 相コミットの問題点は，ブロッキングプロトコルである点である．つまり，いずれかの参加者が停止してしまった場合，停止しなかった参加者もブロックしてしまう場合がある．

表 2 予約関連コマンド

コマンド名	機能	引数	出力
plus_reserve	予約をリクエストする	[-R ホスト名] [-T] -s 開始時刻 -e 終了時刻 -n ノード数	予約 ID
plus_cancel	予約をキャンセルする	[-R ホスト名] [-T] -r 予約 ID	
plus_modify	予約の修正を行う	[-R ホスト名] [-T] -r 予約 ID [-s 開始時刻] [-e 終了時刻] [-n ノード数]	
plus_status	予約の状態を表示	[-R ホスト名] [-r 予約 ID]	予約状態
plus_commit	予約操作をコミットする	[-R ホスト名] -r 予約 ID	
plus_abort	予約操作をアボートする	[-R ホスト名] -r 予約 ID	

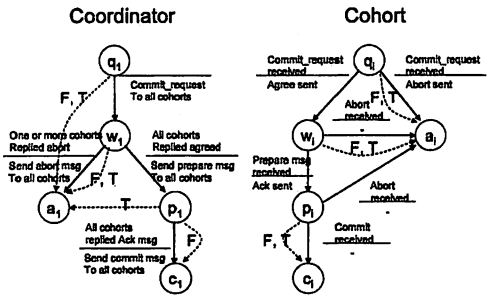

図 5 3 相コミットプロトコル

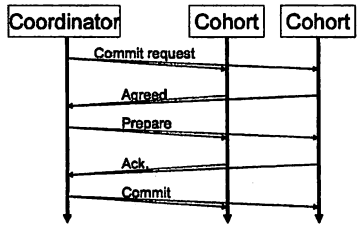

図 6 3 相コミットのメッセージダイアグラム

例としてある Cohort が commit request に対して agree を返信した後に，Coordinator が停止してしまった場合を考えてみよう．他の Cohort の状態がわからないため，この Cohort は commit してよいのか abort してよいのかわからず，Coordinator の復活を待ち続けなければならない．この問題点を解決するべく考案されたのが 3 相コミットである [10),11)]．

ここでは，[11)] にしたがって 3 相コミットプロトコルを概説する．Coordinator と Cohort はそれぞれ 5 つの状態 (q,w,a,p,c) をとる．それぞれの状態遷移図を図 5 に示す．図中の T は，タイムアウトが生じた際の状態遷移を，F はそのコンポーネントがクラッシュから回復した際にとるべき状態遷移を示している．成功時のメッセージダイアグラムを図 6 に示す．

2 相コミットとの相違点は，p という状態 (prepared) が，w と c との間に挿入されていることと，タイムアウトとクラッシュからのリカバリの挙動が明確に定義されていることである．Coordinator はすべての Cohort から agree を受け取ると，すべての Cohort に対して prepare を送信する．すべての Cohort からの prepare に対する ack を待ってから，commit を行う．

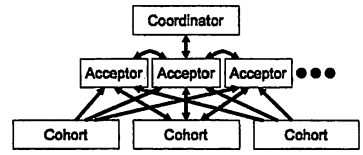

図 7 Paxos コミットの参加者

p 状態が追加されたことによって，Cohort が状態 p にいたった場合に，システム全体がコミットするべきかアボートするべきかがわかっているため，タイムアウトやクラッシュリカバリの際の挙動が明確になるのである．

5.2 Paxos コミットプロトコル

より複雑なコミットプロトコルとして，Paxos コミット [12)] がある．Paxos コミットは，ネットワーク資源と計算資源の同時確保を行うスーパスケジューラ機構である HARC[13)] で使用されている．

Paxos コミットは，資源を確保する側のフォールトに対する寛容性を得ることを主眼としている．2 相コミットでは，特定のタイミングで Coordinator がダウンした場合，資源がコミットされないままブロックしてしまうという自体が生じうる．これに対して,Paxos コミットでは，資源管理モジュールの手前に acceptor と呼ばれるモジュールを多数用意することで対処する．

図 7 に，Paxos コミットプロトコルの参加者を示す．

これらは Paxos コンセンサスアルゴリズムで協調するため，複数の acceptor が停止しても正常に動作することができ，結果として資源をブロックしてしまうことはない．具体的には，操作開始時点で $2N+1$ の acceptor がある場合，そのうち $N+1$ 個の acceptor が動作してさえいれば，総体として正常に機能することが保障されている．

Paxos コミットの問題点としては，機構が複雑になることが挙げられる．ノードの障害に対応するためには，acceptor を独立したノードに用意しなければならず，システム全体の管理のコストが増加する．また，Paxos コミットアルゴリズムを実装するためには，Cordinator からのリクエストに対するレスポンスだけでなく，資源側から acceptor に対して通知を行う必要があり，システム構築が複雑になる．

5.3 議　　論

われわれは,3 相コミット，Paxos コミットを検討したうえで，2 相コミットを採用した．これは，確保対

象となる資源の性質を考慮すると，上記の問題点によるデメリットが，Paxosコミットによる耐故障性のメリットを上回ると考えたからである．

われわれのプロトコルで確保の対象となるのは，資源の特定の時間帯における使用権限である．この権限は高価なものではありうるが，基本的にその時刻がすぎればなくなってしまう揮発性のプロパティである．したがって，かりに Coordinator がダウンすることによって，ある予約が不当にブロックしてしまった状態になったとしても，対象の時間帯以降になれば，システム全体としては整合した状態に回復するため，致命的な問題にはなりえない．このため，われわれは実装の容易な2相コミットプロトコルを採用した．

6. おわりに

われわれは，複数資源の同時確保を安全に行うための機構として，分散トランザクションで一般に用いられる2相コミットプロトコルを用いる，資源予約インターフェイスを設計・実装した．予約インターフェイスは WSRF(Web Services Resource Framework) を基盤プロトコルとし，Globus Toolkit 4 を用いて実装されている．

さらにバックエンドとなる計算資源上の予約管理機構として，PluS 予約管理機構を改良し 2 相コミットの機構を組み込んだ．これらを組み合わせることで，2相コミットプロトコルで操作することのできる計算資源を上位レイヤに対して提供することができる．

今後の課題としては以下が挙げられる．

- 汎用化
  今回設計したインターフェイスは，計算資源を直接の対象としているものの，計算資源に直接依存する部分はなく，汎用的な事前予約のインターフェイスとして使用できる可能性がある．ネットワークやストレージなどの他種資源や，階層的な資源を管理する機構としてのスーパスケジューラのインターフェイスとしての利用を検討する．
- 大規模実験による有効性の確認
  われわれは，9月に日米のネットワーク，計算資源を事前予約による同時予約によって使用する実験を計画している．この実験によって本システムの有効性を確認する．
- PluS 資源予約機構の改良
  現在，PluS 資源予約機構は，計算資源が均質であることを仮定しており，指定できる項目はノード数のみである．今後，非均質なクラスタ環境でも適切な予約操作が行えるよう，アーキテクチャ，メモリ量，ディスク容量などに対応する予定である．

謝　辞

本研究の一部は，文部科学省科学技術振興調整費「グリッド技術による光パス網提供方式の開発」による．

参考文献

1) Foster, I., Carl Kesselman, C. L., Lindell, B., Nahrstedt, K. and Roy, A.: A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation, *Proc. Intl Workshop on Quality of Service* (1999).
2) 竹房あつ子, 中田秀基, 工藤知宏, 田中良夫, 関口智嗣: 計算資源とネットワーク資源を同時確保する予約ベースグリッドスケジューリングシステム, *SACSIS 2006*, pp. 93–100 (2006).
3) Takefusa, A., Hayashi, M., Nagatsu, N., Nakada, H., Kudoh, T., Miyamoto, T., Otani, T., Tanaka, H., Suzuki, M., Sameshima, Y., Imajuku, W., Jinno, M., Takigawa, Y., Okamoto, S., Tanaka, Y. and Sekiguchi, S.: G-lambda: Coordination of a Grid Scheduler and Lambda Path Service over GMPLS, *Future Generation Computing Systems (to appear)* (2006).
4) Tannenbaum, A. S.: *Distributed Operating Systems*, Prentice Hall (1994).
5) : WSRF. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.
6) Foster, I.: Globus Toolkit Version 4: Software for Service-Oriented Systems, *IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779*, pp. 2–13 (2005).
7) 中田秀基, 竹房あつ子, 大久保克彦, 岸本誠, 工藤知宏, 田中良夫, 関口智嗣: 事前予約機能を持つローカルスケジューリングシステムの設計と実装, 情報処理学会 HPC 研究会 2006-HPC-105, pp. 217–222 (2006).
8) : TORQUE Resource Manager. http://www.-clusterresources.com/pages/products/torque-resource-manager.php.
9) : Grid Engine. http://gridengine.sunsource.net.
10) Skeen, D.: Nonblocking Commit Protocols, *SIGMOD '81: Proceedings of the 1981 ACM SIGMOD international conference on Management of data*, ACM Press, pp. 133–142 (1981).
11) Gaddam, S. R.: Three-Phase Commit Protocol. http://ei.cs.vt.edu/ cs5204/ fall99/ distributedDBMS/sreenu/3pc.html.
12) Gray, J. and Lamport, L.: Consensus on Transaction Commit, Technical Report MSR-2003-96, Microsoft Research.
13) MacLaren, J. and Keown, M. M.: HARC: A Highly-Available Robust Co-scheduler, *http://www.realitygrid.org/publications/HARC.pdf* (2006).