計算機アーキテクチャ　144−2
（２００１．７．２５）

プログラマの意図により複数のキャッシュコヒーレンスプロトコルの利用を可能とするソフトウェア分散共有メモリ

城田祐介[†]　吉瀬謙二[†]
本多弘樹[†]　弓場敏嗣[†]

home-based ソフトウェア分散共有メモリにおいて，同一変数を複数ノードで更新するアクセスパタン (migratory access) を実行すると，ホームノードを更新するオーバヘッドが大きいことが指摘されている．本稿では，プログラマがキャッシュコヒーレンスプロトコルをクリティカルセクションごとに選択可能とする機構を提供することで， home-based ソフトウェア分散共有メモリ上で migratory access によるオーバヘッドを削減する方式を提案する． 提案する機構において用いる，migratory access を効率良く扱うためのキャッシュコヒーレンスプロトコルを 3 つ提案する．

Cache Coherence Protocols for Migratory Access in Home-based Software Distributed Shared Memory

Yusuke Shirota,[†] Kenji Kise,[†] Hiroki Honda[†]
and Toshitsugu Yuba[†]

In home-based Software Distributed Shared Memory(DSM) systems, the eager propagations of updates to the home node can have a great impact on its performance, especially for applications with migratory accessed shared data. In order to adapt to memory access patterns that DSM applications exhibit, we propose a mechanism which enables programmers to choose a protocol for each critical sections. For efficient execution of applications with migratory data, we propose three multiple-writer protocols designed for our system.

1. はじめに

計算機クラスタ上の分散メモリを透過的に扱うソフトウェア分散共有メモリシステム [5),7),12)] において，近年の研究 [4),5)] で，home-based ソフトウェア分散共有メモリの高い性能が明らかになってきた．

しかし，home-based ソフトウェア分散共有メモリでは，複数ノードによる同一変数への更新 (migratory access) が，多ノードで実行されるとホームノードの更新が過大なオーバヘッドとなってしまう．

この問題に対して，アクセスパタンに応じてシステムが Single Writer プロトコルと Multiple Writer プロトコルを選択するシステムが提案されている [1)]． こちらは、ページ全体を更新するような migratory access のみを対象としており，その適用範囲は狭い．

また，アプリケーションプログラムが示す様々なアクセスパタンに適合するように，ホームノードを動的に再配置するシステムも提案されている [2)]． この方式では， ページフォルトサービス時に再配置を行うので， migratory access ではホームノードが頻繁に再配置されてしまうため， 再配置に要するコストが問題となる．

本稿では， プログラマがキャッシュコヒーレンスプロトコルをクリティカルセクションごとに選択可能とする機構を提供することで，home-based ソフトウェア分散共有メモリ上で migratory access によるオーバヘッドを削減する方式を提案する． そして，本システムに特化したキャッシュコヒーレンスプロトコルを３つ提案する．

提案方式を評価するために，home-based ソフトウェア分散共有メモリ JIAJIA version 2.1[5)] に提案する１つのプロトコルを実装したのでこの評価結果の概要を報告する．残る２つのプロトコルは，実装には至っていないが，その可能性を検討する．また，ユーザの意図により３つのプロトコルを使い分けることによる利点を検討する．

以後，2 章では，本システムで利用するソフトウェア分散共有メモリ JIAJIA において本提案システムに関連する部分を説明する．3 章ではクリティカルセクションごとにプログラマの意図を反映できる枠組を提

† 電気通信大学 大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications

案し，4 章において migratory access を効率良く扱うためのプロトコルを 3 つ提案する．5 章において，提案するシステムを評価し，6 章で今後の課題について述べる．

2. 本システムで利用する home-based ソフトウェア分散共有メモリ：JIAJIA

JIAJIA[5] は，ユーザレベルライブラリで実現されたページベースソフトウェア分散共有メモリである．一貫性モデルには，Multiple Writer プロトコル [10] を用いて，ScC(Scope Consistency) モデル [8] を実装している．キャッシュコヒーレンスプロトコルには無効化プロトコルを用いている．

JIAJIA は，home-based プロトコル [4] のソフトウェア分散共有メモリである． このため，随時最新の値を保持しているノード (以後ホームノード) がページごとに存在する．ホームノード以外のノードがページを共有する場合，ホームノードから最新のページのコピーをフェッチする．ストアがなされた場合には，diff(差分) をホームノードに eager に転送することでホームノードの更新を行う．

JIAJIA では，同期機構としてロックとバリア同期を提供している．ロックは分散ロックキューにより実装されている．ロックが作るクリティカルセクションの中で，あるページに対してストア命令が行われると，そのページ番号は当該ロックの構造体に記録される．それによって，ロックを獲得した際に，ダーティなキャッシュページを検出できる．

3. ユーザの意図により３つのプロトコルを対象とするアクセスパタンで使い分ける利点

3.1 対象とするアクセスパタン

本稿では，migratory access[3] と呼ばれるアクセスパタンを議論の対象とする． migratory access とは，ある共有変数に対して複数ノードが排他的に実行するメモリ参照 (以後更新操作とする) である．本稿では，migratory access を次の 2 つに分類する．

(1) 一般的な migratory access
(2) 特殊な migratory access

(2) の特殊な migratory access とは，集計処理操作 (4 章参照) のことであり，それ以外の migratory access を (1) として分類する．

3.2 home-based ソフトウェア分散共有メモリ上で migratory access を実行する問題点

home-based ソフトウェア分散共有メモリでは，ホームノードを最新のものとするために， 同期操作の度に diff をホームノードに転送する (**図 1**)☆． このため，応用アプリケーションのアクセスパタンに適合するようにホームノードを指定しないと，diff の生成及び diff によるホームノードの更新がオーバヘッドとなる．

多くの home-based ソフトウェア分散共有メモリシステムでは，ホームノードの割り当てをプログラマがアプリケーション実行前に指定できる． また一部のシステム [5][7] では，バリア同期時にホームノードを動的に再配置することにより diff の転送量を減らすよう工夫している．

しかしながら，migratory access パタンを示す応用において，これらの方法ではどのようにホームを配置しても， diff によるホームノードの更新オーバヘッドを削減することはできない．

JUMP[2] の Migrating-Home プロトコルでは，ページフォルトサービス時に再配置を行うことで，ホームノードの再配置を migratory access パタンにまで適合させることに成功している．しかし， ホームノードの移動には，それを通知するブロードキャストを伴うため， ホームノードの移動が多発する migratory access ではそのコストが問題となる．

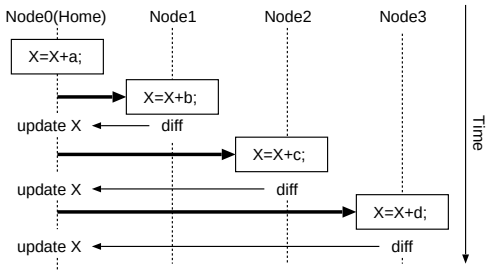

図 1 home-based プロトコルにおけるホームノードの更新
Fig. 1 Fixed-home Protocol

3.3 クリティカクセクションごとにキャッシュコヒーレンスプロトコルを選択する手法の提案

migratory access は他のアクセスパタンと切り離して扱う必要がある．migratory access の高速化を実現するための一つの実現方法として，システムが複数のロック変数の種類を用意する手法を提案し，プログラマは，ロック変数の種類を選択することで，クリティカルセクションごとにキャッシュコヒーレンスプロトコルを選択可能とする．

図 2 に具体例を示す． 例では，全ノードが共有領域の配列 shared に migratory access を行う． また，クリティカルセクションの中で，途中経過をローカルな配列 local にメモリコピーする．

4 章の 3 つのプロトコルのためのロック変数を，それぞれ LM_LAZY，LM_EAGER，LM_COUNTUP と定義する． また，各プロトコルで複数のロック変数が必要なので，LM_protocol_#No. として利用する．特に指定しない場合は，JIAJIA のデフォルトのプロトコルが選択される．

図 2 の例では， 4 章で述べる eager な Temporary

☆ 図中の太線はページの転送，細線は diff の通信である．

Ownership プロトコルをクリティカルセクションに適用することが効果的である．システムが提供するロック変数 LM_EAGER_1 をプログラマが選択することでこれを実現している．

```
Lock(LM_EAGER_1);
  for (i = 0; i < MAXKEY; i++){
     shared[i] += local[i];
     local[i] = shared[i];
  }
Unlock(LM_EAGER_1);
```

図 2 提案する Temporary Ownership プロトコルをクリティカルセクションに適用する例
Fig. 2 Code adapting Temporary Ownership protocol

3.4 ユーザの意図により 3 つのプロトコルを使い分けることによる利点

4 章において，(1) の一般的な migratory access を高速化するプロトコルを 2 つ提案する．提案する 1 つのプロトコルは，ホームノードを更新する複数ノードでの diff を最後に一括して転送することで，migratory access のオーバヘッドを削減する Temporary Ownership プロトコルである．もう 1 つのプロトコルは，もう少し積極的な Temporary Ownership プロトコルである．一般的に，migratory access においては同じ複数ページを利用する可能性が高いと考えられる．仮にこれを予測できれば，migratory access の次点のノードにページを eager に転送することで更にオーバヘッドを削減できる可能性がある．このプロトコルを eager な Temporary Ownership プロトコルとし，これに対して前者を lazy な Temporary Ownership プロトコルとする．

4 章では，lazy な Temporary Ownership プロトコルを用いて，Temporary Ownership プロトコルの基本動作を説明し，その後で eager な実装を議論する．最後に残りの，特別な migratory access である集計処理操作に特化した Countup プロトコルの説明をする．

4. migratory access を対象としたキャッシュコヒーレンスプロトコルの高速化手法の提案

4.1 Temporary Ownership プロトコルの提案

ホームノードを更新する複数ノードでの diff を最後に一括して転送☆することで，同オーバヘッドの削減を目指すプロトコルとして，Temporary Ownership プロトコルを提案する．

図 3 を用いて説明する．Temporary Ownership プロトコルは，ホームノード 0 に対してページをリクエストしたノード 1 に最新のページを送ると同時に，

☆ 実際にはページを転送する．

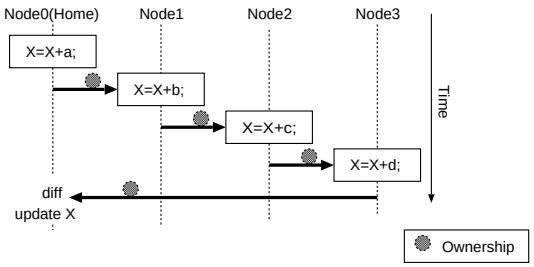

図 3 Temporary Ownership home-based プロトコル
Fig. 3 Temporary Ownership Protocol

ロック変数に対応した，そのページのホームノードの権限を一時的に委譲する．ホームノードの権限を委譲されたノード 1 は，diff を生成して本来のホームノードを更新する必要はない．また，ホームノードの権限を委譲されたノード 1 が，該当ページのホームノードの権限が一時的に委譲されたことをロックに記録してロックを解放することで，次にノード 2 がロックを獲得した際に，そのページのホームの権限の委譲を検出できる．これにより，更に権限を次々に委譲することが可能となり，migratory access パタンにおいてホームノードを介することなく共有変数を更新できる．

ホームノードは diff を利用して更新する．従来 diff は，ホームノード以外で生成してホームノードを更新していた．Temporary Ownership プロトコルでは，ホーム権限を委譲している間に行われる変更点のみ (diff) を，ホーム権限が返却されるときにホームノードに反映させる必要がある．このために，ホーム権限が委譲される直前のコピー (twin) をホームノードにつくっておく．ホームノードは，ホーム権限が戻ると，diff を作成し，更新する．

いつ本来のホームに権限を戻すかについては 4.2 節で議論する．また，Multiple Writer への実装については 4.3 節で議論する．

4.2 プロトコルの起動及び終了条件

Temporary Ownership プロトコルが効果を発揮するのは，短い間に同一ページへのアクセスが集中し，キューにロックリクエストが複数たまっているような migratory access の場合である．なぜなら，キューにたまっているロックリクエスト数に等しい数のノードは，ロックサービスを待っているだけで，有益な仕事をしていないからである．

そこで，キューにロックリクエストが一定数以上たまった時に，Temporary Ownership プロトコルを起動することで，ロックサービスのスループットをあげる仕組みが必要となる．

JIAJIA 上に，Temporary Ownership プロトコルを実装する**図 4** の例を使って説明する．ここでは，ノード 0(ページ 0，1 のホームノードとする) 以外のノードが，Temporary Ownership プロトコルを適用するためのロック変数 LM_LAZY_1 を利用して，ページ

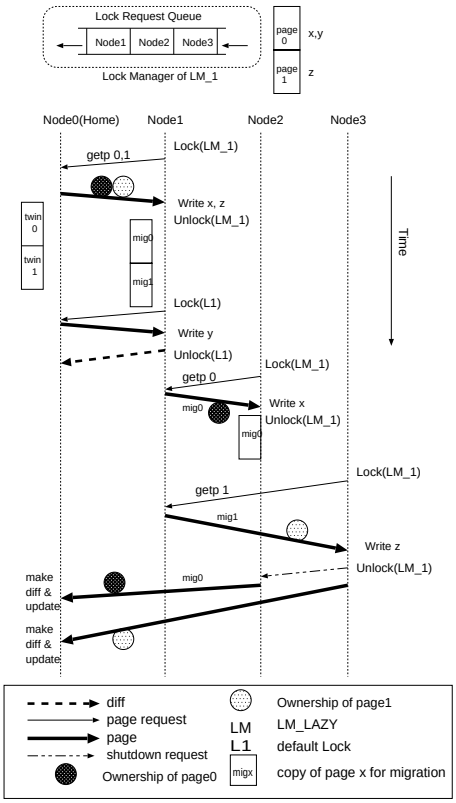

図 4 Multiple Writer プロトコルを利用した Temporary Ownership プロトコル
Fig. 4 Temporary Ownership Protocol with Multiple Writers

0(共有変数： x,y) とページ 1(共有変数： z) に対して migratory access を実行すると仮定する． ロックのリクエストはロックマネージャの FIFO キューに，(ノード 1 →ノード 2 →ノード 3) の順番でたまっている．この順番を，旅の順番と定義する☆．

例において，Temporary Ownership プロトコルの起動条件を，リクエストが 3 つ以上たまることとする． キューにたまっているリクエストが 3 未満のとき，ロックマネージャは JIAJIA のデフォルトのプロトコルに従いロックサービスをする．

キューに 3 つ以上のリクエストがたまると，ロックマネージャはプロトコルを起動し，以下の手順が実行される．

ロックマネージャは，ノード 1 にロック LM_LAZY_1 をサービスするときに，旅の順番も通知する． ノード 1 は，ロック LM_LAZY_1 を解放するとき，ロックマネージャにはロック LM_LAZY_1 を転送せずに，旅の順番において次点のノード 2 に，ロック LM_LAZY_1 と旅の順番を通知する☆☆．同様に，ノード 3 がロックと旅の順番を受け取る．

旅の順番の最後であるノード 3 は，プロトコルを終了するルーチンを起動する．ホームノードを更新するように一時的にホーム権限を委譲されている全てのノード (ノード 2 と自分自身) にプロトコルの終了リクエスト (shutdown) を発行する．これにより，ホームノードに権限が戻り，更新も正しく行われる．

この時，ロック LM_LAZY_1 は，ロックマネージャに返却される．ロックマネージャが，Temporary Ownership プロトコルを起動してから，ロックマネージャにロックが返却されるまでを旅と定義する．

旅の間に発行されるロックのリクエストは新規にロックマネージャのキューにたまる．

4.3 Multiple Writer プロトコルの実装

False Sharing の問題を解決するためには，diff を利用して実装する Multiple Writer プロトコルは不可欠である． よって，Temporary Ownership プロトコルにおいても，Multiple Writer プロトコルを利用する．

図 4 の例を使って説明する．ノード 1 に注目する．ノード 1 は，ロック変数 LM_LAZY_1 を獲得し，x に対してストアする． ロック変数 LM_LAZY_1 を解放後，続けて JIAJIA のデフォルトのプロトコルを利用するロック変数 L1 を獲得し， y に対してストアするために，ページ 0 をフェッチする☆☆☆．しかし，この後でノード 2 にページ 0 をサービスするときに，先に行った x へのストアが反映されていなければならない．このため，上のケースにおいて，ノード 1 はページ 0 のコピー (mig0) をとっておき，ノード 2 からページ 0 がリクエストされたら，mig0 をサービスする．

4.4 積極的な Temporary Ownership プロトコル

一般的に，migratory access において，同じ複数ページを利用する可能性は高いと考えられる． 前述の Temporary Ownership プロトコルは，ページがリクエストされた時点でページサービスを行う lazy な実装であったが，旅においては同じページを利用する可能性が高いと予測できれば， ページサービスを eager に行うことで，更なるオーバヘッドの削減が可能となる．具体的には， 旅の順序における次点のノードにロックを転送するとき，migratory access を行ったページ群も合わせて転送する．lazy な実装ではなくて eager な実装にする利点は，ページごとのページサービスがプリフェッチによりなくなることである． 欠点は，予め転送しておいたページを利用しなかったときに，転送量が増えることである．

☆ この順番は，最適化可能である．

☆☆ これにより，ロックをロックマネージャに返却するメッセージと，キューにおいて次点のノードにロックを転送するメッセージが一つのメッセージに削減される．

☆☆☆ このとき，フェッチしてくるページが最新とみなせることは ScC モデルが保証している．

4.5 特別な migratory access である集計処理操作

集計処理操作は，特殊な migratory access である．本稿では，集計処理操作を次のように定義する．

集計処理操作: 1) 更新操作に可換/結合法則が成立し，2) 明示的な同期操作まで，演算の中間結果を別途使用しない migratory access.

集計処理操作に対して，Log Based Consistency(LBC) プロトコル [11] が提案されている． しかし，クリティカルセクションごとに選択するキャッシュコヒーレンスプロトコルとしては，LBC プロトコルを利用することは難しい．

上記の 2 つの特性を利用し，集計処理操作に特化したキャッシュコヒーレンスプロトコルを提案する．

図 5 の例を使って説明する． アドレス X に, アドレス X を含むページのホームノード 0, ノード 1, ノード 2，ノード 3 がそれぞれ, a，b，c，d の加算操作を n 個繰り返し実行した結果, それぞれ, $A = na$，$B = nb$, $C = nc$，$D = nd$ 加算したとする．これまで述べてきたように，migratory access は排他制御されるので, X への加算操作は逐次化される．提案するプロトコルは, 集計処理操作の特性を利用し，並列に同一アドレスへ加算することを許す．集計処理操作を終了する明示的な一貫性維持操作が実行されるまでの間, アドレス X への加算は完全にローカルな処理になり, 並列実行される．この目的で，新たに *sumup_barrier* という操作を導入する．この一貫性維持操作 *sumup_barrier* によって各ノードがローカルに加算した値 (intdiff) がホームノードに転送され，その和をホームノードが計算することで, 一貫性を保証する．

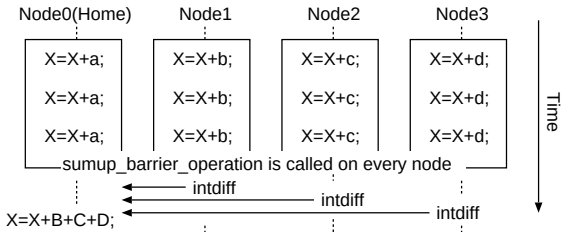

図 5 集計処理に特化したプロトコルの一貫性維持操作

Fig. 5 Barrier synchronization designed for our count-up operation-specific protocol

5. 評　　価

5.1 評 価 方 法

ソフトウェア分散共有メモリ JIAJIA version 2.1[5] に集計処理操作のためのプロトコルを実装した．この評価結果を報告する．残る 2 つのプロトコルは，実装には至っていないが，その可能性を検討する．また，ユーザの意図により 3 つのプロトコルを使い分けることによる利点を検討する．

5.2 集計処理操作の評価

評価には, データマイニング応用における Apriori アルゴリズム [9] のアソシエーションルール検索を用いた．Apriori では, 膨大なトランザクションデータベースを走査し集計処理するので，加算操作を実行するクリティカルセクションに集計処理に特化したキャッシュコヒーレンスプロトコル選択することが有効である．

Myrinet で相互接続された PentiumIII 600MHz + Red Hat Linux の 4 プロセッサで構成されるパーソナルコンピュータクラスタ上に, 提案するプロトコルを追加した JIAJIA を実装し，その上にアソシエーションルール検索プログラムを移植し，実行した．トランザクションデータは文献 [9] に従い, 人工的に作成した: 総アイテム数 =600 件, レコード中のアイテム数 = 平均 10 件のポアソン分布, 最小サポート =0.5%, トランザクション数 =400,000 件.

表 1 は，アソシエーションルール検索プログラムの実行時間の大半を占めるパス 2 におけるアソシエーションルール検索時間を示している． 表 1 より，十分な台数効果が得られていることが確認できた．

ノード数	パス 2 の実行時間 [sec]	台数効果
1	618	1.00
2	360	1.71
4	187	3.30

表 1 集計処理に特化したプロトコルを適用したアソシエーションルール検索時間 [s]

Table 1 Performance of association rule mining with our protocol

5.3 Temporary Ownership プロトコルの検討

eager な Temporary Ownership プロトコルと lazy な Temporary Ownership プロトコルの可能性を検討する． 検討には，NAS Parallel Benchmarks から IS(Integer Sort) を用いる．IS では，各ステップ終了時に，図 2 に示す migratory access が行われる．評価に用いる問題サイズは， 鍵の数 $= 2^{27}$，鍵の最大値 $= 2^{12}$ とする．N ノードで実行される IS の通信量として，ホームノードの更新に要するメッセージ数と，最新のページのコピーをホームノードにリクエストする回数 (理論上の最適値) を表 2 に示す． IS では，図 2 より明らかであるように，ページ全体にストアしているので，diff のデータ量とページサイズが等しい． これにより，ホームノードの更新に要する通信量は，提案する 2 つのプロトコルではノード数 N に対しておおよそ $1/(N-1)$ になることが期待される．IS では，図 2 から明らかであるように，eager な Temporary Ownership プロトコルの予測がすべて当たる． これにより，eager にページを転送しておくことが効果を発揮するので， IS では大幅な性能向上を

見込むことができる．

message type	JIAJIA	lazy	eager
page request	$2(N-1)$	$2(N-1)$	$N-1$
update message	$N-1$	1	1

表 2 IS の通信量
Table 2 Communication Costs for Integer Sort

本方式は，固定のホームと複数の仮ホームとが混在することを許す．従って， Multiple Writer の状況で，migratory access が実行される場合などには，最も効果を発揮するはずである (**図 6**)．

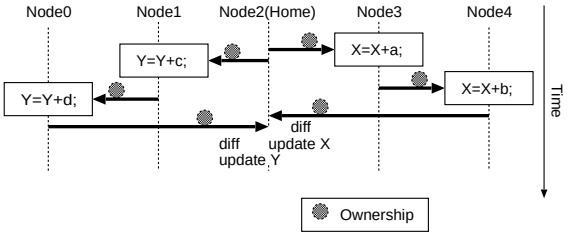

図 6 複数ライタによる migratory access
Fig. 6 migratory access by multiple writers

6. 今後の課題

JUMP の Migrating-Home プロトコルは，多くのベンチマークで大幅な性能向上を示しているので，Temporary Ownership プロトコルとの組み合わせ的なアプローチの検討をしたい． また，本稿の趣旨とは反するが，lazy な実装の Temporary Ownership プロトコルは，ユーザが明示的にロック変数で指定しなくても，キューにロックリクエストがたまるというプログラムの特性を利用することで，システムが自動的に migratory access パタンを検出し，プロトコルを切替えることが可能である．

今後，クリティカルセクションごとに複数のキャッシュコヒーレンスプロトコルを選択することが有効なアプリケーションを移植し，評価したい．また，無効化プロトコルと更新プロトコルを選択可能にすることも課題の一つであると考えている．

参 考 文 献

1) Cristiana Amza, Alan L. Cox, Sandhya Dwarkadas, Li-Jie Jin, Karthick Rajamani, Willy Zwaenepoel. Adaptive protocols for software distributed shared memory. Proc. of IEEE Special Issue on Distributed Shared Memory, pp.467-475, Mar. 1999.
2) B. Cheung, C.L. Wang, Kai Hwang. A Migrating-Home protocol for implementing Scope consistency model on a cluster of workstations. pp. 821-827, The 1999 International Conference on parallel and Distributed Processing Techniques and Applications(PDPTA'99), Las Vegas, Nevada, USA.
3) W. -D. Weber, A. Gupta. Analysis of cache invalidation patterns in multiprocessors. In Proceedings of the 3rd Symposium on Architectural Support for Programming Languages and Operating Systems, pp. 243-256, Apr. 1989.
4) Y. Zhou, L. Iftode, K. Li. Performance evaluation of two home-based lazy release consistency protocols for shared virtual memory systems. In Proceedings of the Second USENIX Symposium on Operating System Design and Implementation, pp. 75-88, Nov. 1996.
5) Weiwu Hu, Weisong Shi, Zhimin Tang. JIAJIA: An SVM system based on a new cache coherence protocol. Proc. of the High Performance Computing and Networking(HPCN '99), pp.1147-1150, April 1999.
6) Alan L. Cox, Eyal de Lara, Charlie Hu, Willy Zwaenepoel. A performance comparison of homelss and home-based lazy release consistency protocols in software shared memory. Proc. of the Fifth High Performance Computer Architecture Conference, pp. 279-283, Jan. 1999.
7) 手塚 宏史, 堀 敦史, 石川 裕, 原田 浩. ソフトウェア分散共有メモリ SCASH におけるページ管理ノードの動的再配置機構の実装と評価, 情報処理学会研究報告 99-HPC-77(SWoPP'99), pp.89-94, 1999.
8) L. Iftode, J. P. Singh, K. Li. Scope consistency: A bridge between release consistency a nd entry consistency. Proc. of the 8th ACM Annual Symp. on Parallel Algorithms and Architectures(SPAA' 96), pp.277-287, June 1996.
9) R. Agrawal, R. Srikant. Fast algorithms for mining association rules. Proc. of 20th Very Large Database Conference(VLDB'94), pp. 487-499, Sept. 1994.
10) P. Keleher, A. L. Cox, S. Dwarkadas, W. Zwaenepoel. An evaludation of software-based release consistent protocols. Journal of Parallel and Distributed Computing. Vol. 29, pp. 126-141, Oct. 1995.
11) Hideaki Hirayama, Hiroki Honda, Toshitsugu Yuba. Scalable data mining with log based consistency DSM for high performance distributed computing. Proc. 6th IEEE Int. Conf. on Engineering of Complex Computer Systems, pp.143-150, Sept. 2000.
12) P. Keleher. The coherent virtual machine. TR93-215, Department of Computer Science, University of Maryland, Sept. 1995.