S-DSMシステムにおけるページ要求時の受信通知を削減する方式

吉瀬謙二[†] 田邊浩志[†] 多忠行[†,☆]
片桐孝洋[†] 本多弘樹[†] 弓場敏嗣[†]

PCクラスタ上での，ソフトウェア分散共有メモリ（S-DSM）システムの高速化手法について議論する．JIAJIAなどのいくつかのS-DSMシステムは，計算ノード間の通信に，信頼性に関する機能を提供していないUDP（ユーザデータグラムプロトコル）を利用する．JIAJIAの実装では，ミドルウェアのレベルで，通信のエラーを検出して回復するために，1回のメッセージ送信に対して1回の受信通知（Ack）を利用する．本稿では，1回のメッセージ送信に対して，必ずしも1回の受信通知が必要ではないことを利用して，受信通知を削減する方式を提案する．評価結果から，MM（Matrix Multiply）のようにページ転送の実行頻度が高いベンチマークにおいて劇的な速度向上を達成し，16ノードの場合には，受信通知を省略しない場合と比較して，提案手法を用いることで92%の速度向上を達成できることを確認した．

A Method to Reduce the Acknowledgment Message for a Page Request of S-DSM Systems

Kenji Kise,[†] Hiroshi Tanabe,[†] Tadayuki Ohno,[†,☆]
Takahiro Katagiri,[†] Hiroki Honda[†] and Toshitsugu Yuba[†]

We discuss the inter-process communication in software distributed shared memory (S-DSM) systems. Some S-DSM systems, such as JIAJIA, adopt the user datagram protocol (UDP) which does not provide the reliable communication between the computation nodes. In the implementation of JIAJIA, an acknowledgment is used in order to detect a communication error. In this paper, first, we show that an acknowledgment is not necessarily required for each message transmission in the middleware layer. Second, a method to reduce the acknowledgment overhead for a page request is proposed. From the evaluation results, we show that Matrix Multiply (MM) of high page transfer frequency achieves a drastic speedup compared with the conventional communication method of not omitting the acknowledgment.

1. はじめに

汎用のPCを用いたクラスタシステム（PCクラスタ）が並列計算のための環境として普及してきている．このようなPCクラスタやワークステーションクラスタを利用して，仮想的な共有メモリを実現するソフトウェア分散共有メモリ（S-DSM）が提案され[5]，いくつかのシステム[7),9),11)~13),16)]が実装されている．

TreadMarks[4]やJIAJIA[2]などのいくつかのS-DSMは，PCクラスタを構成する計算ノード間の通信に，信頼性に関する保証機能を提供していないが通信オーバヘッドの少ないUDP（ユーザデータグラムプロトコル）を利用する．このため，通信のエラーを検出して回復するために，ミドルウェアのレベルで1つのメッセージ送信に対して1つの受信通知（Ack）を利用する．

図1 (a)に，node1からnode2へ，3つのメッセージMsg-1，Msg-2，Msg-3を送信する様子を示す．これは，S-DSMにおける一般的な通信の様子を示しており，受信側のノードnode2では，メッセージを受信するとただちに，受信通知のメッセージAck-1，Ack-2，Ack-3を送信する．

図1 (b)は，受信通知を省略した場合の通信の様子を示している．すべての送信メッセージが正しい順序で受信されると仮定すると，(b)に示すように，受信

† 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
☆ 現在，日本電気株式会社
Presently with NEC corporation

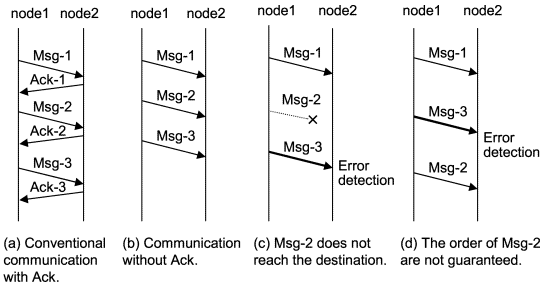


図 1 一般的な通信の様子 (a) と受信通知を省略した通信 (b)．通信が正しく行われない場合を (c) と (d) に示す
Fig. 1 S-DSM communication with and without the acknowledgment.

通知に関するコストを削減できる．受信通知を省略することは全体の送受信の回数を半分に削減できることを意味し，特に，通信が頻繁に発生している場合に大幅な高速化を期待することができる．ただし，我々が対象としている UDP 通信の場合には，(c) に示すように，一部のメッセージ（Msg-2）が正しく届かない，(d) に示すように，メッセージ Msg-2 と Msg-3 の順序が保証されないといった通信エラーが発生する．このような場合においても，正しく動作するように S-DSM システムを構築する必要がある．

本稿では，PC クラスタ上での S-DSM の高速化手法を議論する．図 1 (b) に示すように，UDP を利用する S-DSM において，ミドルウェアのレベルで 1 回のメッセージ送信に対して，必ずしも 1 回の受信通知が必要ではないことを利用して，受信通知を削減する方式を提案する．我々が開発を進めている S-DSM システム Mocha に提案方式を実装し，その有効性を検証する．

本稿の構成を示す．2 章で関連研究についてまとめ，3 章で，受信通知を省略する方式を提案する．4 章で方式の実装を述べ，5 章で評価を行う．6 章で議論し，7 章で本稿をまとめる．

2. 関 連 研 究

2.1 ソフトウェア分散共有メモリ

本節では，PC やワークステーションを汎用のイーサネットスイッチで接続するクラスタシステムをターゲットとする代表的な S-DSM システムと，それらが利用しているネットワークプロトコルを概観する．

1980 年代の後半に Li によって構築された IVY[5] が最初の S-DSM システムとされている．IVY では，すべての要求を逐次化する Sequential Consistency モデルを採用している．Sequential Consistency モデルには，プログラマが挙動を把握しやすいという利点があるが，一方で，ページ更新のオーバヘッドのために効率の良いシステムの構築が困難になるという問題点がある．

1990 年代の前半には，緩いメモリ一貫性モデルとして Lazy Release Consistency を採用する TreadMarks[4] が発表され，代表的な S-DSM システムの 1 つとなっている．TreadMarks は非営利団体に低価格で販売されている数少ない S-DSM システムであり，高い安定性と性能を提供する．TreadMarks はイーサネットと ATM の上で動作するように設計されている．イーサネットの上で動作させる場合には，UDP を利用する．

JIAJIA は，1990 年代の後半に実装された比較的新しい S-DSM システムの 1 つで，Scope Consistency モデルを採用する．フリーウェアとして公開されている点，比較的ソースコードの可読性が高いという点などから性能比較の対象の 1 つとして利用されることが多い．JIAJIA は，UDP を用いて実装されている．

JUMP（JIAJIA Using Migrating Home Protocol）[1] は，ホーム移動を行わない JIAJIA を改良し，ホーム移動の手法を追加した S-DSM システムである．JIAJIA と同様に，UDP を用いて実装されている．

国内では，1990 年代の後半から，SMS[16] と呼ばれる S-DSM システム☆が構築されている．SMS では，システム作成時に UDP あるいは TCP のどちらかを選択できるように実装されている．しかしながら，評価結果から，TCP を用いた実装と比較して，UDP を用いた実装の性能が高いことが明らかになっている．

岡本ら[10] は，アクセス履歴に基づくホーム移動の手法を提案し，独自に構築した S-DSM システムの上に提案方式を実装し評価を行っている．この S-DSM においても，UDP が利用されている．

このように，PC やワークステーションを汎用のイーサネットスイッチで接続する代表的ないくつかの S-DSM システムの実装において，UDP が広く利用されていることが分かる．

2.2 受信通知メッセージの省略

本稿では，UDP を利用する S-DSM において，ミドルウェアのレベルで 1 回のメッセージ送信に対して，必ずしも 1 回の受信通知が必要ではないことを利用して，受信通知を削減する方式を検討する．

検討する受信通知の省略というアイデアは古くから議論されているものである．たとえば，文献 6) では，クライアント・サーバモデルにおける通信の問題と

☆ 残念ながら，SMS のソースコードは提供されていない．

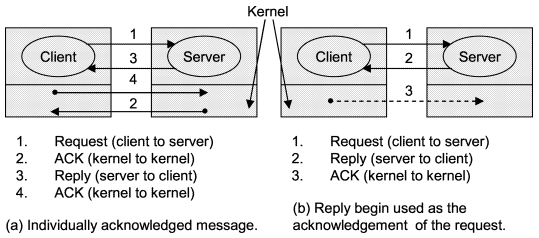


図 2 クライアント・サーバモデルにおける通信（文献 6) からの引用）
Fig. 2 Communication in the client-server model [6].

して，受信通知（Ack）を省略する手法が議論されている．この議論で利用されている動作を**図 2** に示す．ここでは，Reply を受信通知として利用することで，受信通知を省略できる点，Reply の性質によっては，Reply に対する受信通知が必要ないという点が議論されている．このため，図 2 (b) の Ack が破線になっている．

本稿は，図 2 に示されるクライアント・サーバモデルにおける受信通知の省略の考え方を，初めて，S-DSM の領域において利用するものである．また，その具体的な実装方法を議論し，システムの評価を通じて提案方式による高い性能向上を確認する．

3. 受信通知を省略する方式の提案

3.1 ソフトウェア分散共有メモリ Mocha

本章では，前提とする S-DSM システム Mocha の位置付けを明確にし，その後，受信通知を省略する方式を提案する．

Mocha は，S-DSM システム JIAJIA を参考にしながら，次の 2 つを設計方針として独自に構築を進めている S-DSM である．

- 並列処理の環境として手軽に利用できる．
- ノード数が大きい部分で高い性能を達成する．

Mocha のアプリケーションプログラムは JIAJIA と同様の API を用いて記述される．このため，変更を加えることなく JIAJIA のアプリケーションを Mocha の上で動作させることができる．

本稿の評価に利用する版 Mocha Version 0.2 は，JIAJIA と同様に，8 KB のページ単位で共有メモリを管理する．共有メモリを構成するページはユーザの指定に従って，1 つのホームノードに対応付けられる．現在の版では，ホームのマイグレーションは実装されていない．また，一貫性モデルとして Scope Consistency [3] を採用している．このため，Mocha は，JIAJIA に近い挙動を示す．

Mocha の実装に際しては，JIAJIA のコードの持つオーバヘッドや扱いにくい点を慎重に検討し，これらを考慮してコードを書き直している．本稿で提案する方式の実装に加えて，特に，送受信で利用するメッセージの形式を改良することで通信量を削減するチューニングが施されている．

```
 1  #define OP_NULL       100 /* null: not allocated    */
 2  #define OP_EXIT       101 /* server: exit           */
 3  #define OP_GETP       110 /* server: get page       */
 4  #define OP_GETPGRANT  111 /* server: get page grant */
 5  #define OP_DIFF       112 /* server: diff           */
 6  #define OP_DIFFGRANT  113 /* server: diff grant     */
 7  #define OP_BARR       114 /* server: barrier        */
 8  #define OP_BARRGRANT  115 /* server: barrier grant  */
 9  #define OP_ACQ        116 /* server: aquire         */
10  #define OP_ACQGRANT   117 /* server: aquire grant   */
11  #define OP_WAIT       118 /* server: wait           */
12  #define OP_WAITGRANT  119 /* server: wait grant     */
13  #define OP_INV        120 /* server: invalidate     */
14  #define OP_WTNT       121 /* server: write-notice   */
15  #define OP_REL        122 /* server: release        */
16  #define OP_BCAST      123 /* server: broadcast      */
```

図 3 S-DSM システム Mocha Version 2.0 が利用するメッセージの種類．ページ要求に関する通信 OP_GETP, OP_GETPGRANT の高速化手法を評価する
Fig. 3 The list of message types in Mocha Version 0.2.

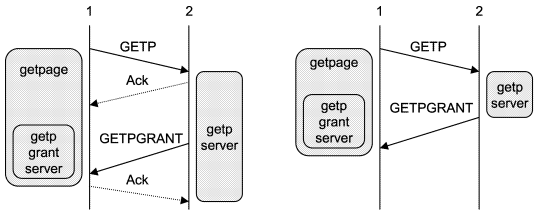


図 4 ページ要求に関するオリジナルの通信 (a) と受信通知を省いた通信 (b)
Fig. 4 (a) The behavior of a page request. (b) Page request without acknowledgment.

Mocha Version 0.2 が利用するメッセージの種類を**図 3** にまとめる．本章では，この中で，ページ要求（getpage）に関する通信（OP_GETP, OP_GETPGRANT）の受信通知を省略する方式を提案する．ページ要求とは，キャッシュされていない共有メモリ空間の参照によりページフォールトが発生したときに，必要とするページを持っているノードからページを取得する処理である．

3.2 ページ要求に関する受信通知の省略

ページ要求の挙動を**図 4** (a) に示す．今，node1 の計算機があるページを必要としており，そのページは node2 が持っているとする．node1 は，参照しているメモリアドレスを引数として，getpage という関数を呼び出す．getpage では，ページ要求に対応する GETP という操作のメッセージを作成して，node2 に送信する．このメッセージを受信した node2 は応答の受信通

知（Ack）メッセージを返信する．その後，node2 では，ページ要求に応えるために，getpserver という関数を呼び出す．この関数は，必要とするページの情報を GETPGRANT という操作のメッセージに格納し，node1 に送信する．node1 は，GETPGRANT という操作のメッセージを受信し，これに対応する getpgrantserver という関数を呼び出す．この関数が，受信したページの情報を適切なメモリ領域に格納し，最初に起動した getpage の処理を終わらせるための変数をリセットする．getpage はグローバル変数を用いてビジーウェイトでページが到着するまで待っているが，GETPGRANT の到着により，ビジーウェイトを抜けて，アプリケーションの処理を継続する．

このようなページ要求の処理の流れにおいて，GETP のメッセージを送信して，GETPGRANT の受信まで getpage という関数がビジーウェイトで待つことに着目し，受信通知を省略する方式を提案する．

getpage という関数がビジーウェイトを抜けることにより，GETP，GETPGRANT という 2 つのメッセージにおいて通信エラーが発生していないことを保証できる．一方，ビジーウェイトにおいて，一定時間の待合せを行い，このタイムアウト時間内に GETPGRANT を受信しなかった場合には，通信エラーが発生した可能性があると推定できる．この場合には，再度，GETP を送信して，GETPGRANT の到着を待つ．ただし，GETP，GETPGRANT に関して，同じメッセージが複数回到着することでシステムが意図していない状態に推移しないように設計する必要がある☆．これにより，図 4 (b) に示すように，GETP，GETPGRANT に関する受信通知メッセージを省略できる☆☆．

図 5 に，受信通知の省略を行わないオリジナルの getpage の擬似コードを示す．2 行目でグローバル変数 getpwait をセットする．3 行目で，ページ要求のメッセージを生成し，4 行目で作成したメッセージを送信する．要求に対応するメッセージが到着すると，受け取ったページを適切に処理し，グローバル変数 getpwait がリセットされる．これにより，5 行目の while ループが終了し，ページ要求の関数 getpage が終了する．

```
1  void getpage(address_t addr){
2    getpwait=1;
3    generate_message(OP_GETP, addr);
4    send_message();
5    while(getpwait); /** busy wait **/
6  }
```

図 5 受信通知の省略を行わないオリジナルの getpage の疑似コード
Fig. 5 The pseudo-code of the original getpage for which the acknowledgment is not omitted.

```
1  void getpage(address_t addr){
2    getpwait=1;
3    for(i=0; i<GETPAGE_MAX_RETRY; i++){
4      generate_message(OP_GETP, addr);
5      send_message();
6      while(not_timeout() && getpwait); /* busy wait */
7      if(getpwait==0) break;
8    }
9  }
```

図 6 提案手法を施した getpage の疑似コード．タイムアウトの場合には，ページ要求を繰り返す
Fig. 6 The pseudo-code of the function getpage to be implemented in the proposed method. The while loop in Line 6 finishes when the global variable getpwait has been reset or a timeout has occurred.

図 6 に提案する方式を実装する getpage の疑似コードを示す．グローバル変数 getpwait がリセットされた，あるいは，タイムアウトになった時点で 6 行目の while ループを終了する．7 行目の if 文で，getpwait がリセットされていると判定された場合には，要求したページを受け取ったことになるので，ページ要求の処理を終了する．そうでなければ，タイムアウトにより while ループが終了し，通信エラーが発生した可能性があるので，3 行目の for ループにより，再度，ページ要求のメッセージを送信する．

4. 実　　装

本章では，3.2 節で議論した受信通知の省略方式の実装を議論する．

S-DSM システム Mocha では，通信の種類を識別するために 8 ビットの char 型の変数を用いている．しかし，図 3 にまとめたように，操作の種類はたかだか 20 種類程度しかないので，8 ビットすべてを必要とすることはない．そこで，下位の 7 ビットを操作の種類として利用して，上位の 1 ビットを受信通知を必要とする（信頼性を要求する）通信なのかを示すフラグとして利用する．このビットのことを reliable_msg_flag と呼ぶことにする．reliable_msg_flag がセットされている場合には受信通知を必要し，セットされていない場合には受信通知を必要としない通信と定義する．

☆ たとえば，6.1 節で議論する待合せを行う処理の場合には，サーバがメッセージを受信した回数をカウントし，この回数がノード数と等しくなった時点で待合せを終了する．このような設計では，同一のクライアントから複数のメッセージを受け取ることが間違った動作を引き起こす．この場合には，カウンタを用いた実装から，各ノードに対応するフラグを用いた実装に変更するといった対策が必要となる．

☆☆ 図 4 の提案手法において，getpage を Client，getpserver を Server に置き換えると，図 2 と同じ構成となる．

```
 1  #define MSG_MASK 0x80  /* 10000000 */
 2  void send_one_message(){
 3
 4    if(sendqh->op==OP_GETP || sendqh->op==OP_GETPGRANT)
 5      sendqh->op = sendqh->op | MSG_MASK;
 6
 7    for(i=0; i<MAX_RETRY; i++){
 8      ret = sendto(message);
 9      if(sendqh->op & MSG_MASK) return;
10
11      while(not_timeout())
12        if(FD_ISSET(fds[serverproc], &fds)!=0){
13          recvfrom(message);
14        }
15    }
16  }
```

図 7 メッセージを送信する関数 send_one_message の疑似コード
Fig. 7 The pseudo-code of the message transmission function send_one_message.

図 7 に，メッセージを送信する関数の疑似コードを示す．4 行目と 5 行目で，送信するメッセージが GETP，GETPGRANT であるかを判定し，そうであれば，reliable_msg_flag をセットする．8 行目の sendto でメッセージを送信する．reliable_msg_flag がセットされている場合には，受信通知を待つ必要がないので，9 行目の return によりメッセージ送信の関数を終了し，速やかに次の処理を開始できる．そうでない場合には，受信通知メッセージの受信を待つ必要がある．11 行目から 14 行目の while 文で受信通知メッセージを受信するまで待ち，その後，メッセージ送信の関数を終了する．

メッセージを受信する部分では，受信したメッセージの reliable_msg_flag を参照して，このフラグがセットされている場合には，従来システムと同様に，受信通知のメッセージを送信する．そうでない場合には，受信通知のメッセージの送信を省略する．

ページ要求のメッセージ（GETP）を送信する関数 getpage は，図 6 に示したコードを利用する．タイムアウトの間隔には次のトレードオフがある．この間隔が短すぎる場合，サーバの負荷が高いときには GETPGRANT を送信するまでの間隔が長くなるので，誤って通信エラーと判断される頻度が高くなる．これにより，無駄なページ要求の再送が発生し，アプリケーションの性能が低下する要因となる．一方，この間隔が長い場合，この間隔が通信エラーを検出するためのオーバヘッドとなるため，通信エラーの頻度に応じてアプリケーションの処理性能が低下する．すなわち，タイムアウトの間隔は，間違って通信エラーと判定されない程度に十分に長く，通信エラーが発生した場合にアプリケーション性能が大幅に低下しない程度に短く設定する必要がある☆．

これら，メッセージを送信する関数，メッセージを受信する関数，ページ要求のメッセージ（GETP）を送信する関数の変更により，提案手法を実現できる．

5. 提案方式の評価

本章では，4 章の実装を施した S-DSM システム Mocha を用いて，提案手法の性能を評価する．提案手法を利用しない版の Mocha のことを Mocha base と呼ぶことにする．

5.1 評価環境

評価には，16 ノードの PC（マザーボード上に 1000BASE-T のポートを装備）をギガビットスイッチ（NETGEAR GS524T）で接続した PC クラスタを利用する．各ノードは，Intel Pentium4 Xeon 2.8 GHz を 2 個搭載する SMP 型の計算機で，1 GB のメモリを搭載する．クラスタのシステムソフトウェアとして RedHat Linux 7.3 をベースとした SCore 5.6.1 を用いている．

本稿に示すデータは 1 台のノードに 1 つのプロセスのみを割り当てて（ノードあたり 1 個の CPU のみを用いて）測定したものである．それぞれのベンチマークプログラムの実行時間は，3 回の測定の算術平均により計算する．

5.2 ベンチマークプログラム

ベンチマークプログラムには，N-queens の世界記録を樹立したベンチマーク[8] を Mocha のために書き直したもの，JIAJIA Version 2.2 に添付されている LU（parallel dense blocked LU factorization，no pivoting），Water（N-body molecular simulation），SOR（Red-Black Successive Over-Relaxation），MM（Matrix Multiply）を利用する．これらには，実行時間の計測などのために多少の変更を加えている．

N-queens のパラメータとして，N = 17，タスクの割当て単位 8 を利用する．主要計算部の逐次実行時間は 55 秒である．

LU のパラメータとして，行列サイズ 1,024×1,024，ブロックサイズ 8，要素 double を利用する．主要計

☆ GETPGRANT を送信するまでの間隔や，通信エラーの頻度は PC クラスタの構成やネットワーク性能，アプリケーションに依存する点に注意する必要がある．ただし，5.4 節の通信エラーの頻度に示すように，16 ノードの PC をギガビットスイッチで接続した PC クラスタでは，通信エラーの頻度は非常に小さい．このため，本評価では，間違って通信エラーと判定されない程度の長い間隔として，タイムアウトまでの間隔を 50 msec に設定している．

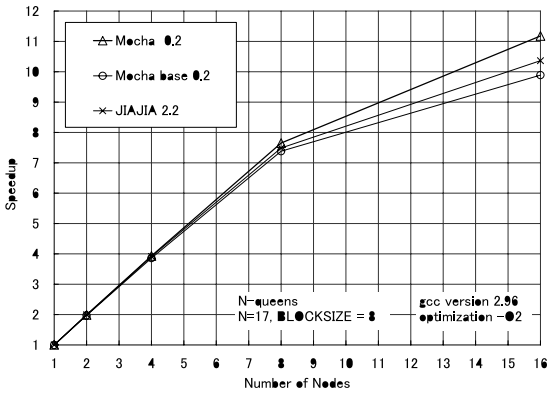


図 8 ベンチマーク N-queens．受信通知を省略する S-DSM の性能．JIAJIA の 1 ノードの性能を 1 として正規化

Fig. 8 The performance comparison of the S-DSM systems. Benchmark is N-queens. The speedup is normalized by the elapsed time of JIAJIA single node.

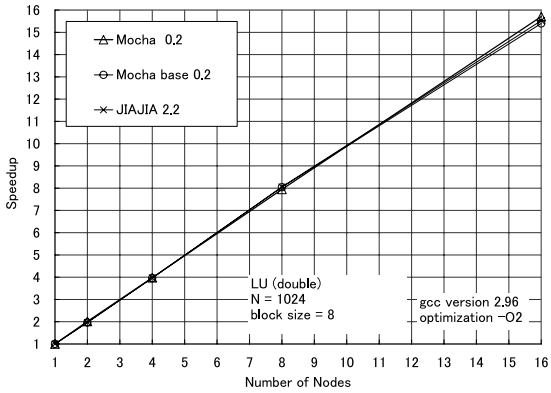


図 9 ベンチマーク LU．受信通知を省略する S-DSM の性能．JIAJIA の 1 ノードの性能を 1 として正規化

Fig. 9 The performance comparison of the S-DSM systems. Benchmark is LU.

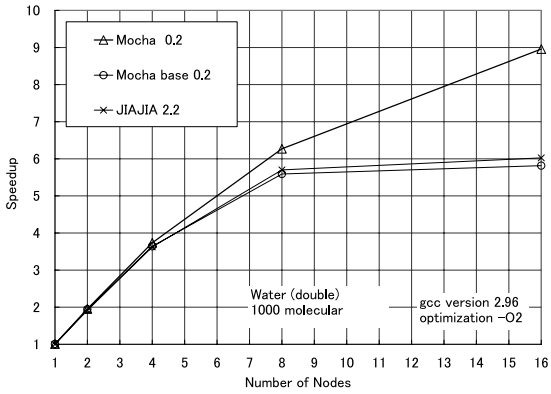


図 10 ベンチマーク Water．受信通知を省略する S-DSM の性能．JIAJIA の 1 ノードの性能を 1 として正規化．

Fig. 10 The performance comparison of the S-DSM systems. Benchmark is Water.

算部の逐次実行時間は 267 秒である．

Water のパラメータとして，粒子数 1,000 を利用する．主要計算部の逐次実行時間は 7.7 秒である．

SOR のパラメータとして，行列サイズ M = 4,096, N = 4,096，iterations = 400 を利用する．主要計算部の逐次実行時間は 98.1 秒である．

MM のパラメータとして，行列サイズ 2,048×2,048, 要素 double を利用する．主要計算部の逐次実行時間は 39.7 秒である．

5.3 受信通知の省略方式による性能向上

動作速度を測定した結果を図 8 から図 12 にまとめる．横軸はノード数，縦軸は速度向上率で，JIAJIA Version 2.2 の 1 ノード（1 プロセッサ）の性能を 1 として正規化している．1 ノードで動作させた JIAJIA と，並列化されていない逐次プログラムの実行時間はほぼ同様であるため，逐次プログラムに対する性能向上も同様の挙動となる．

JIAJIA 2.2，受信通知メッセージを省略する提案手法を実装した Mocha，提案手法を利用しない Mocha base，一部のベンチマークでは TreadMarks Version 1.0.3.3[4] の結果を示す．

図 8 の N-queens の結果を議論する．N-queens の場合には，16 ノードの構成で JIAJIA の場合に 10.3 という台数効果を達成する．一方，提案方式を実装する Mocha の台数効果は 11.1 であり，JIAJIA と比較して 8%の高速化を達成する．また，Mocha base と Mocha との比較より，16 ノードのときに，提案手法による 13%の高速化を確認できる．特に，ノード数が多いところで，提案手法を実装する Mocha が有利となる．

図 9 の LU の結果を議論する．LU では計算量と比較して通信量が少ないので，すべての S-DSM において，ノード数を増やすに従って理想的な速度向上を達成する．このように，提案手法によるオーバヘッドはほとんどみられない．

図 10 の Water の結果を議論する．Water の場合には，16 ノードの構成で JIAJIA の場合に 6.0 という台数効果（1 ノードに対する速度向上）を達成する．一方，提案方式を実装する Mocha の台数効果は 8.9 であり，JIAJIA と比較して 49%の高速化を達成する．また，Mocha base と Mocha との比較より，16 ノードのときに，提案手法による 54%の高速化を確認できる．

図 11 の SOR の結果を議論する．SOR の場合には，16 ノードの構成で JIAJIA の場合に 10.3 という台数効果を達成する．一方，提案方式を実装する Mocha

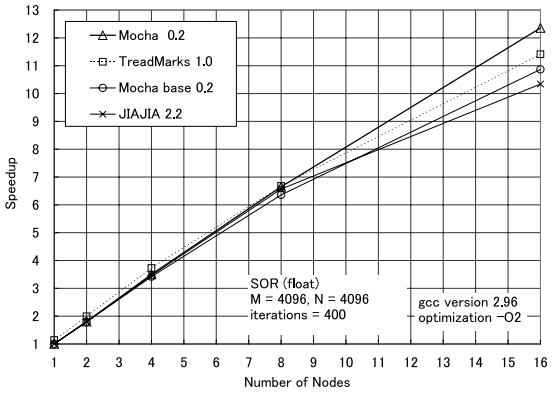


図 11 ベンチマーク SOR．受信通知を省略する S-DSM の性能．JIAJIA の 1 ノードの性能を 1 として正規化
Fig. 11 The performance comparison of the S-DSM systems. Benchmark is SOR.

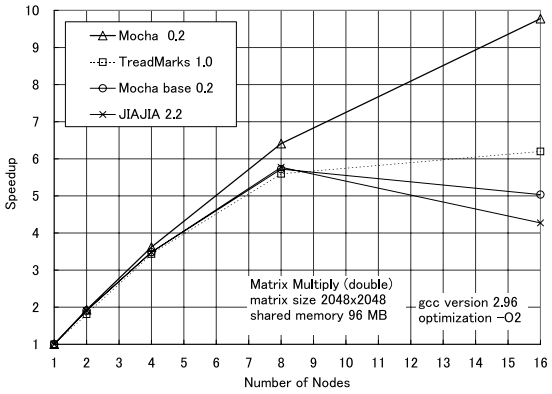


図 12 ベンチマーク MM．受信通知を省略する S-DSM の性能．JIAJIA の 1 ノードの性能を 1 として正規化
Fig. 12 The performance comparison of the S-DSM systems. Benchmark is MM.

の台数効果は 12.3 であり，JIAJIA と比較して 19%の高速化を達成する．また，Mocha base と Mocha との比較より，16 ノードのときに，提案手法による 13%の高速化を確認できる．特に，ノード数が多いところで，提案手法を実装する Mocha が有利となる．

図 12 の MM の結果を議論する．MM の実行では，GETP と GETPGRANT という 2 種類の通信が全通信に対する大部分の処理時間を占める．このため，ページ要求に関するオーバヘッドを低減する提案手法の効果が顕著になっている．従来システムの JIAJIA 2.2，Mocha base では，8 ノードを超えるところで性能向上が得られていない．TreadMarks では性能向上が緩やかになる．一方，提案手法を実装した Mocha では，16 ノードを用いたときにも高い性能向上が維持され，このとき，9.7 倍という高い台数効果を達成する．16 ノードの Mocha base と比較すると，提案手法より，16 ノードの Mocha は 92%という高い高速化を達成する．

表 1 PC クラスタ全体の 1 秒あたりの通信エラー回数
Table 1 The number of communication errors per second of a PC cluster.

benchmark	2 node	4 node	8 node	16 node
N-queens	0.0	0.0	0.0	0.0
LU	0.0	0.0	0.0	0.0
Water	0.0	0.0	0.0	0.0
SOR	0.0	111	692	1,360
MM	0.0	0.0	0.0	0.0

本節の評価結果から，次の結論を得る．提案手法を実装する Mocha は，すでに理想的な速度向上を示す LU を除くベンチマークで高速化を達成する．特に，MM のようにページ転送の実行頻度が高いベンチマークにおいて，劇的な高速化を達成する．MM の 16 ノードの場合には，受信通知を省略しない場合と比較して，提案手法を用いることで 92%という高い高速化を達成する．

5.4 通信エラーの頻度

提案手法を利用しない Mocha base を利用し，S-DSM で発生するすべての通信を対象としてエラーの回数を計測した結果を**表 1** にまとめる．5.1 節にまとめた Pentium4 Xeon とギガビットスイッチの構成の上で動作させた結果である．このデータは，すべてのノードで発生した通信エラーの回数の和を求め，この値をベンチマークの処理時間で割ることで，PC クラスタ全体としての 1 秒あたりのエラーの回数（3 回の測定の平均）を計算したものである．

表 1 の結果から，SOR 以外のベンチマークでは通信エラーがまったく発生していないことが分かる．SOR では，ノード数を増やすに従って，通信エラーの頻度が増加し，16 ノードの場合には，1 秒あたり 1,360 回のエラーが発生する．

一方，提案手法を実装した Mocha で同様の測定を行ったところ，SOR を含むすべてのベンチマークで通信エラーの回数がゼロという結果を得た．提案手法を用いることで受信通知メッセージの通信が削減される．このため，ネットワークのトラフィックが緩和され，通信エラーが削減されたと考えられる．

本節では，信頼性に関する保証機能を提供していない UDP を利用する S-DSM システムの通信エラーの発生頻度を検討した．この結果，通信エラーの頻度が非常に小さいこと，また，提案方式を利用することで通信エラーを解消できることを明らかにした．

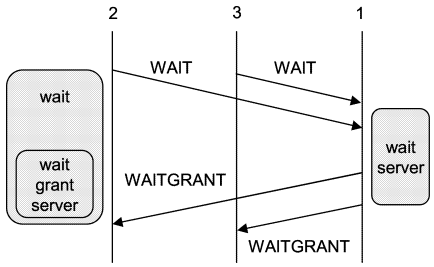


図 13 受信通知を省略する待合せの様子．node1 を管理サーバに設定している
Fig. 13 Omission of acknowledgment for wait. Node 1 is the server that manages the wait.

6. 議　　論

6.1 待合せ処理に関する受信通知の省略

本節では，S-DSM におけるその他の通信処理の 1 つとして，すべてのプロセスの待合せを行う処理（wait）における受信通知の省略方式を検討する．バリア処理（barrier）と異なり，待合せ（wait）はメモリの一貫性の維持を行わない処理である．

node1，node2，node3 という 3 つのノードで待合せを行う処理の様子を**図 13** に示す．図が煩雑になることを避けるために，node3 で起動される関数 wait，waitgrantserver を省略している．この図では，node1 が待合せを管理するサーバとする．node2 と node3 は，サーバに WAIT を送信する．サーバは，WAIT が到着した回数をカウントして，すべてのノードから WAIT を受信した時点で待合せが終了したとして，WAITGRANT をブロードキャストして処理を継続する．サーバ以外のノードでは，WAIT を送信した後に，ビジーウェイトにより WAITGRANT を待つ．WAITGRANT を受信することで waitgrantserver という関数が呼び出され，ビジーウェイトを抜けて，アプリケーションの処理を継続する．

ページ要求処理 getpage の場合は，1 つのクライアントと 1 つのサーバの間の通信により処理が進行していく．一方，待合せ処理（wait）に関しては，複数のクライアントが 1 つのサーバにアクセスするという点でページ要求と異なる特徴を持つ．しかしながら，待合せ処理の場合にも，node2，node3 をクライアント，node1 をサーバとして，GETP が WAIT に，GETPGRANT が WAITGRANT に置き換わったととらえると，基本的な処理はページ要求と同様に実現できる．ただし，待合せ処理の場合には，ノード間の処理のばらつきなどにより，WAIT を送信してから長い時間が経過した後に WAITGRANT を受信するという場合が発生する．このとき，WAITGRANT を待つことによるタイムアウトと，通信エラーにより生じるタイムアウトとを区別することは難しい．通信エラーではないときに複数の WAIT を送信する場合には，これが新しい通信のオーバヘッドとなる．このため，待合せなどの通信処理における受信通知の省略手法に関しては，新たに発生する通信オーバヘッドとのトレードオフを考慮した詳細な検討が必要となる．

6.2 その他の受信通知の省略方式

本節では，受信通知を省略するいくつかの方式を示し，提案した方式との利害得失を議論する．

受信通知を省略する一般的な方式として，受信側で，メッセージに付加されているシーケンシャル番号が期待する値と異なることで通信エラーを検出し，エラーの場合にのみ回復を行う方式を考えることができる．この方式には，提案手法と異なり，ページ要求を含むすべてのメッセージの受信通知を省略できるという利点がある．しかしながら，シーケンシャル番号などを用いた単純なエラー検出方式では，エラーが発生した時刻と検出される時刻との開きが大きくなり回復が複雑になる可能性があるという欠点を持つ．たとえば，ある通信がエラーとなり，1 秒後の次のメッセージ送信によりエラーが検出されるケースを考える．このとき，1 秒間という長い間の処理☆を巻き戻すためには，この間隔の処理の履歴を保存し，これを用いて回復するという複雑な機構が必要となる．一方，提案手法では，あるタイムアウト間隔（本評価では 50 msec）の範囲で，通信エラーを検出することができる．また，受信通知を省略する範囲をページ要求に限定することで，通信エラーからの回復が容易となる．

別の方式として，n 回のメッセージ送信に対して，1 回の受信通知メッセージを送信する方式を考えることができる．この方式は，上に述べた受信通知を省略する一般的な方式に近い（n を無限大とすると，上の方式と同様となる）．この方式にも，ページ要求を含むすべてのメッセージの受信通知を省略できるという利点がある．しかしながら，この方式を用いたとしても，エラーが発生した時刻と検出される時刻との開きが大きくなり，複雑な回復処理が必要となるという欠点を克服することはできない．

エラーが発生した時刻と検出される時刻との開きが大きくなるという問題を解決するために，一定の間隔で，エラーが発生しているかどうかを確認する通信を

☆ あるノード間の通信が 1 秒間発生していないとしても，それ以外のノード間の通信が頻繁に発生する場合があることに注意する必要がある．

導入する手法を考えることができる．この通信の間隔を短くすることで，エラーが発生した時刻と検出される時刻との開きを小さくすることができる．しかしながら，新しい通信を導入することに関しては，これが新たなオーバヘッドとなる可能性があるため，慎重に議論する必要がある．また，S-DSM として新しいメッセージの導入とそれを処理する関数の実装が必要となる．これらは，容易な作業ではない．

我々は，本節で議論した方式を候補として検討した結果として，比較的実装が容易な方式として，ページ要求時の受信通知を削減する方式を提案し，実装・評価を行った．

6.3 今後の課題

本稿では，ページ要求時の受信通知を削減する方式を提案評価した．一方，受信通知の削減ではないが，通信オーバヘッドの削減に関して，メッセージのとりまとめや相乗りによる手法も有効と考えられる．

メッセージのとりまとめに関しては，将来利用するであろうページ番号をストライド値予測やコンテキストベースの値予測の手法を用いて予測[17] して，複数のページをまとめて送信することで通信回数を削減する研究[15] を進めている．予備評価の結果から，メッセージのとりまとめの有効性を確認している．今後，メッセージをまとめて送受信する手法を S-DSM システム Mocha に実装し，その有効性を評価していく予定である．

メッセージのとりまとめは，複数のページ要求や複数のページ転送といった同一のメッセージをまとめて送信する方式である．同様に，異なる種類のメッセージをパッキングして転送することで，送受信の回数を削減するメッセージの相乗りの手法を考えることもできる．このような，メッセージの相乗りによる高速化の検討は今後の課題である．

本稿では，ミドルウェアのレベルで受信通知を削減する方式と実装の詳細を議論した．ミドルウェアの層ではなく，アプリケーションやオペレーティングシステムのレベルのチェックポイントを利用することができれば，ミドルウェアの送受信のレベルで受信確認の必要性を判断する必要はない．これらいくつかのレベルの実現可能性と性能に関する検討は今後の課題である．

S-DSM システムの普及のためには，様々な支援ツールを提供しなければならない．特に，S-DSM では，通信や共有メモリの利用に関する詳細をユーザが記述しない．このことは，MPI などと比較して，S-DSM を用いたアプリケーションの記述が容易になるという利点をもたらす．一方で，共有メモリがどのような状態にあり，どのように推移しているかを把握することが困難となり，デバッグやチューニングが困難になるという欠点となる．この欠点を補うために，共有メモリや通信の状態をユーザに見せるビューアやデバッグ支援ツールを提供することが望ましい．現在，S-DSM システム Mocha の設計・実装と並行して通信などの履歴情報を表示する開発支援ツール S-CAT（**図 14**）の開発を進めている[14]．画面の左上のウィンドウには，それぞれのノード間の通信の履歴が矢印で表示されている．下のウィンドウにはアプリケーションプログラムの全実行における大域的な情報が表示されている．右側には，個別の通信の情報などの詳細データや統計データが表示されている．

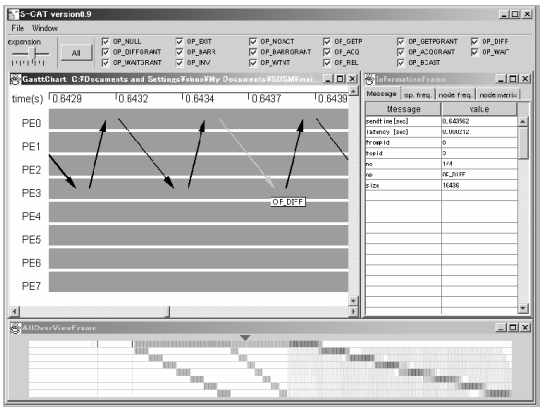


図 14 S-DSM システム Mocha と並行して開発を進めている支援ツール S-CAT の画面．8 ノードで MM を実行した様子．下のウィンドウには大域的な通信の様子が表示されている

Fig. 14 The screenshot of the programming support tool S-CAT.

これらの開発支援ツールを含むソフトウェア分散共有メモリ環境として Mocha の開発を進め，そのソースコードを公開していく予定である．

7. おわりに

本稿では，PC クラスタ上での，ソフトウェア分散共有メモリ（S-DSM）システムの高速化手法について議論した．

JIAJIA などのいくつかの S-DSM システムは，計算ノード間の通信に，信頼性に関する機能を提供していない UDP を利用する．このため，通信のエラーを検出して回復するために，ミドルウェアのレベルで，1 つのメッセージ送信に対して 1 つの受信通知（Ack）を利用するが，この受信通知は必ずしも必要というわけではない．このことを利用して，ページ要求に関する受信通知を削減する方式を提案するとともに，その

実装を議論した．

我々が開発を進めている S-DSM システム Mocha に，ページ要求の受信通知を省略する方式を実装し，いくつかのベンチマークを利用して，その性能を測定した．測定結果から，MM のようにページ転送の実行頻度が高いベンチマークにおいて劇的な速度向上を達成し，16 ノードの場合には，受信通知を省略しない場合と比較して，提案手法を用いることで 92%という高い速度向上を達成できることを確認した．また，UDP を利用する S-DSM システムの通信エラーの発生頻度を測定し，通信エラーの頻度が非常に小さいこと，提案方式のページ要求の受信通知を省略する方式を利用することで通信エラーを解消できることを明らかにした．

謝辞 本研究の一部は，文部科学省科学研究費補助金（課題番号 16300004「スーパクラスタを指向した性能拡張性を持つソフトウェア分散共有記憶方式の研究」）の援助による．

参 考 文 献

1) Cheung, B.W.-L., Wang, C.-L. and Hwang, K.: Migrating-Home Protocol for Implementing Scope Consistency Model on a Cluster of Workstations, *International Conference on Parallel and Distributed Processing Techniques and Applications* (*PDPTA '99*) (1999).
2) Eskicioglu, M.R., Marsland, T.A., Hu, W. and Shi, W.: Evaluation of the JIAJIA Software DSM System on High Performance Computer Architectures, *Proc. 32nd Annual Hawaii International Conference on System Sciences* (*HICSS-32*) (1999).
3) Iftode, L., Singh, J.P. and Li, K.: Scope Consistency: A Bridge between Release Consistency and Entry Consistency, *Proc. 8th Annual ACM Symposium on Parallel algorithms and architectures*, pp.277–287 (1996).
4) Keleher, P., Cox, A.L., Dwarkadas, S. and Zwaenepoel, W.: TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems, *Proc. Winter 94 Usenix Conference*, pp.115–131 (1994).
5) Li, K.: IVY: A Shared Virtual Memory System for Parallel Computing, *Proc.International Conference on Parallel Processing* (*ICPP '88*), Vol.2, pp.94–101 (1988).
6) Tanenbaum, A.S.: *Modern Operating Systems*, Prentice-Hall International Editions (1992).
7) 早田恭彦，中田秀基，松岡 聡，小川宏隆：Java 向けソフトウェア分散共有メモリの実現，情報処理学会論文誌：プログラミング，Vol.42, No.SIG 3(PRO 10), pp.14–26 (2001).
8) 吉瀬謙二，片桐孝洋，本多弘樹，弓場敏嗣：PC クラスタを用いた N-queens 問題の求解，電子情報通信学会論文誌，Vol.J87-D-I, No.12, pp.1145–1148 (2004).
9) 小島好紀，佐藤三久，朴 泰佑，高橋大介：MPI 上のソフトウェア分散共有メモリシステム，情報処理学会研究報告 2003-HPC-98 (2004).
10) 岡本秀輔，阿部拓弥：ソフトウェア分散共有メモリにおけるアクセス履歴に基づくホーム移動，情報処理学会論文誌：コンピューティングシステム，Vol.45, No.SIG 6(ACS 6), pp.66–73 (2004).
11) 立川 純，小出 洋：マルチスレッドを用いてページ転送遅延を隠蔽するソフトウェア DSM システム，情報処理学会研究報告 2004-HPC-99, pp.151–156 (Aug. 2004).
12) 丹羽純平：コンパイラが支援するソフトウェア DSM におけるプリフェッチ機構，情報処理学会研究報告計算機アーキテクチャ研究会，2004-ARC-156, Vol.2004, No.156, pp.7–12 (Feb. 2004).
13) 城田祐介，吉瀬謙二，本多弘樹，弓場敏嗣：ホームベースソフトウェア分散共有メモリ上で Migratory Access を効率良く処理する権限委譲プロトコル，情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム，Vol.44, No.SIG 1(HPS 6), pp.103–113 (2003).
14) 多 忠行，吉瀬謙二，片桐孝洋，弓場敏嗣：複数の S-DSM を対象とする開発支援ツール S-CAT の設計と実装，情報処理学会研究報告 2005-ARC-161 (Jan. 2005).
15) 田邊浩志，吉瀬謙二，本多弘樹，弓場敏嗣：通信粒度を動的に変更するソフトウェア分散共有メモリ，2002 年電子情報通信学会総合大会，No.D-6-5, (Mar. 2002).
16) 緑川博子，飯塚 肇：ユーザレベル・ソフトウェア分散共有メモリ SMS の設計と実装，情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム，Vol.42, No.SIG 9(HPS 3), pp.170–190 (2001).
17) 坂口朋也，鈴木 祥，吉瀬謙二，弓場敏嗣：通信粒度予測機構を実装したソフトウェア分散共有メモリ，情報処理学会第 67 回全国大会，Vol.1, No.4ZB-2, pp.173–174 (Mar. 2005).

(平成 17 年 1 月 20 日受付)
(平成 17 年 4 月 15 日採録)

吉瀬　謙二（正会員）

1995 年名古屋大学工学部電子工学科卒業．2000 年東京大学大学院情報工学専攻博士課程修了．工学博士．同年電気通信大学大学院情報システム学研究科助手．計算機アーキテクチャ，並列処理に関する研究に従事．電子情報通信学会，IEEE-CS，ACM 等各会員．

田邊　浩志（学生会員）

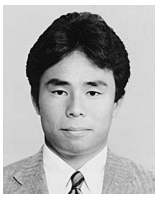
2002 年電気通信大学情報工学科卒業．2004 年同大学大学院情報システム学研究科修士課程修了．現在，同大学院博士後期課程在学中．分散メモリシステムにおける並列処理実行方式の研究に従事．

多　　忠行

1980 年生．2003 年電気通信大学情報システム学研究科卒業．2005 年同大学大学院情報システム学研究科修士課程修了．ソフトウェア分散メモリシステムの研究に従事．現在，日本電気株式会社にて社会人として鋭意勉強中．

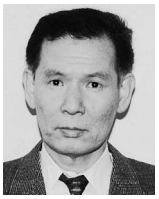
片桐　孝洋（正会員）

電気通信大学情報システム学研究科助手．1994 年豊田工業高等専門学校情報工学科卒業．1996 年京都大学工学部情報工学科卒業．2001 年東京大学大学院理学系研究科情報科学専攻博士課程修了．博士（理学）．2001 年 4 月日本学術振興会特別研究員 PD，2001 年 12 月科学技術振興事業団研究者を経て，2002 年 6 月より現職．並列計算機を用いた効率の良い行列計算アルゴリズムの研究，およびソフトウェア自動チューニングの研究に従事．2002 年山下記念研究賞受賞．著書『ソフトウエア自動チューニング：数値計算ソフトウエアへの適用とその可能性』，（慧文社，2004）．日本ソフトウェア科学会，日本応用数理学会，ACM，IEEE-CS，SIAM 等各会員．



本多　弘樹（正会員）

1984 年早稲田大学理工学部電気工学科卒業．1991 年同大学大学院理工学研究科博士課程修了．1987 年より同大学情報科学研究教育センター助手．1991 年より山梨大学工学部電子情報工学科専任講師．1992 年より同助教授．1997 年より電気通信大学大学院情報システム学研究科助教授．並列処理方式，並列化コンパイラ，並列計算機アーキテクチャ，グリッド等の研究に従事．工学博士．電子情報通信学会，IEEE-CS，ACM 各会員．平成 15 年度山下記念研究賞受賞．



弓場　敏嗣（フェロー）

1966 年神戸大学大学院工学研究科修士課程修了．（株）野村総合研究所を経て，1967 年通商産業省工業技術院電子技術総合研究所（現，独立行政法人産業技術総合研究所）に入所．以来，計算機のオペレーティングシステム，見出し探索アルゴリズム，データベースマシン，データ駆動型並列計算機等の研究開発に従事．その間，計算機方式研究室長，知能システム部長，情報アーキテクチャ部長等を歴任．1993 年より，電気通信大学大学院情報システム学研究科教授．並列処理・分散処理の科学技術一般に興味を持つ．工学博士．情報処理学会，電子情報通信学会各フェロー．日本ソフトウェア科学会，日本ロボット学会，ACM，IEEE 各会員．