成蹊大学理工学研究報告
J. Fac. Sci. Tech., Seikei Univ.
Vol.42 No.1 (2005) pp.19-26

ソフトウェア DSM のトランザクションシステムへの適用

長尾　知幸[*1]，緑川　博子[*2]，飯塚　肇[*2]

An Application of a Software DSM to Transaction System

Tomoyuki NAGAO[*1], Hiroko MIDORIKAWA[*2] and Hajime IIZUKA[*2]
{nagao,midori,jim}@sirius.is.seikei.ac.jp

ABSTRACT： This paper proposes a new model to implement a transaction processing system on a cluster with software distributed systems (SDSM). The model was used to experimentally implement an auction system and the usefulness of SDSM's to transaction processing systems was evaluated. The result showed that SDSM's have possibility to be effectively used for these applications. However, it became clear that current SDSM's should be augmented with several features to be effectively used for the implementations of transaction systems.

Keywords：Cluster, Software DSM, Transaction system

(Received March 25, 2005)

１．はじめに

高性能マイクロプロセッサの低価格化と高バンド幅ネットワークの普及に伴い，安価に構築できる PC クラスタが一般化している。こうしたクラスタを用いた並列処理の API としては，メッセージパッシングモデルによる MPI が普及しているが，共有メモリモデルプログラムをクラスタ上で利用したいという要望も強い。

その理由は，共有メモリモデルはメッセージパッシングモデルで必要な煩雑なメッセージ送受信記述を用いずにプロセス間通信を記述できるため，可読性の高いプログラムを書きやすいからである。しかし，クラスタでは，他ノードのメモリに直接アクセスすることができないので，そのまま共有メモリモデルによってプログラムすることはできない。そこで，クラスタ上で共有メモリモデルを利用できるようにするために，“仮想的に共有メモリを構築する”ソフトウェア分散共有メモリ(SDSM)と呼ばれる機構について研究が行われている。

現在のところ，SDSM は，科学技術計算問題の並列処理への適用が主体で，商用分野に属する問題への適用は報告例が殆どない。今後，いっそうクラスタが普及していくであろうことを考えると，上記のような利点を持つソフトウェア分散共有メモリを商用分野でも利用することは，価値があると考えられる。

本稿では，特に負荷分散に焦点を当て，代表的商用分野の応用であるトランザクション処理を対象に，SDSM を有効に利用するためのシステムモデルを提案し，そのモデルに基づいたオークションシステムの実装について報告する。現状の，限られた条件での SDSM をそのまま用いて，トランザクション処理を行うための初期実験ではあるが，有効なデータが得られた。

２．クラスタと SDSM

2．1　並列コンピュータ

複数の計算資源を 1 つの処理に用いる場合，ハードウェア構成としては，対称型マルチプロセッサ(SMP)や NUMA 型分散共有メモリマシンのような共有メモリ型と，クラスタを代表とする分散メモリ型がある。

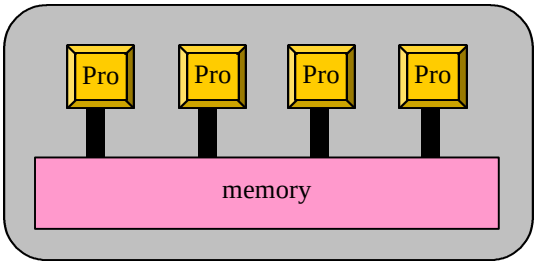
共有メモリ型は，図 1(a)のように，複数のプロセッサが 1 つのメモリを共有しているアーキテクチャである。

[*1] 工学研究科情報処理専攻学生[現:三菱電機(株)]

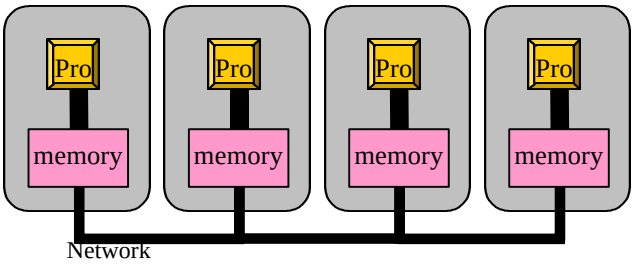
[*2] 工学研究科情報処理専攻

どのプロセッサからもメモリに高速にアクセスできるが，ハードウェアが高価になってしまう上，メモリバスがボトルネックとなりプロセッサ数を多くできないという弱点がある。

一方，分散メモリ環境は，図 1(b)のようにプロセッサとメモリで構成される計算ノードをネットワーク接続して構築できるため，ノードに通常の PC，接続に通常の LAN を用いれば安価に構築することができ（この場合は，通常クラスタと呼ばれる），拡張性もよい。しかし，各プロセッサは自ノードのメモリにしか直接アクセスできないので，そのままではメッセージパッシングモデルしか使用できない。



(a) 共有メモリマシン



(b) 分散メモリマシン

図１　並列マシンのメモリ構成

2．2　並列プログラミングモデル

並列処理用のプログラミングモデルには，メッセージパッシングモデルと共有メモリモデルがある。

メッセージパッシングモデルでは，プログラマがメッセージの送受信を明示的に記述することで通信を行うためプログラマの負担が大きい。

一方，共有メモリモデルは，すべてのプロセスがメモリ空間を共有しているので，メッセージではなく，共有変数への Read/Write を用いてプロセス間の通信を行う。そのため，通常，コードは記述がより容易で可読性も高い。

当然ながら，クラスタは，そのハードウェア構成から，メッセージパッシングモデルを用いるのが自然で，共有メモリモデルは単純に適用することができない。

2．3　SDSM

クラスタのような分散メモリ環境において，共有メモリモデルを適用できるようにするために，ソフトウェアによって仮想的に共有メモリ環境を提供するためのシステムがソフトウェア分散共有メモリ（SDSM）である。SDSM は，図 2 のように，各ノードのメモリの一部をどのプロセスからも同じ内容に見える（常に全く同じ内容が記憶されているわけではないが）ようにして，実効的にプロセス間で共有するメモリ空間を提供する。代表的な SDSM としては，SMS[1]，TreadMarks[2]，JIAJIA[3]等がある。

この SDSM 環境では，プログラミングの際にデータがノード内にあるか，外にあるかによって，アクセスするための記述を使い分ける必要がなく，可読性が高いプログラムが書きやすくなる。しかし，SDSM は物理的には分散メモリ環境であるため，ノード外メモリにアクセスする場合はネットワークを経由する。そのためノードの内外によってメモリのアクセス速度が異なり，本来の共有メモリモデルをそのまま適用してしまうと，共有する必要のない変数でさえも共有されることが発生する。その通信によるオーバーヘッドが速度低下の要因となる可能性がある。

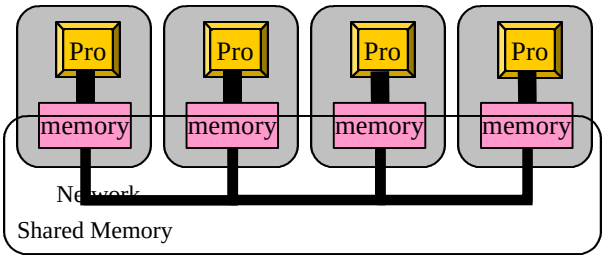


図２　SDSM の構成

3．クラスタを用いたトランザクション処理

ファイルアクセスを伴うデータベースを実現するサーバシステムのクラスタでの実装は，ディスクへのアクセス方法によって以下に示す 2 種類に分類される。

3．1　シェアド･ナッシング方式

シェアドナッシング(SN)方式は，クラスタ内のノードごとに処理するファイルを固定し，そのファイルへのアクセスが必要なときにはリクエストは担当ノードへ必ず送られる方式である。図 3 では，ノード 0 はファイル A にのみアクセスができ，ノード 1 はファイル B のみに，ノード 2 はファイル C のみに，ノード 3 はファイル D のみに，それぞれアクセスできる。この方式では，ファイルをアクセスするノードが限定されるのでノード内にそのデータをキャッシュに保持することができ，処理効率が良くなるという利点がある。しかし，適切なファイル分割を行わないと一部のノードに負荷が集中し，一方でその他のノードは遊休状態になっている可能性が生じる。また，適切なノードを決定するためにはその分割状態を把握していなければならない。更に，あるノードが故障した場合，システムが完全には動かなくなってしまうこともあり，新しいノードを加えたときには改めて分割を行わなければならない，という欠点もある。

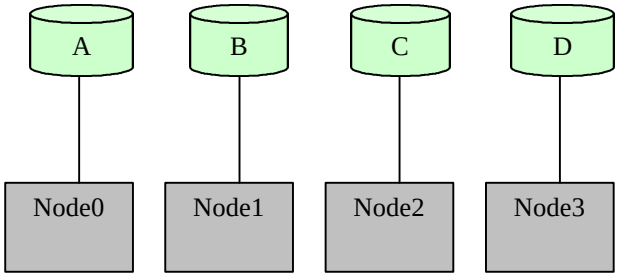


図３　シェアドナッシング方式

3．2　シェアド･オール方式

シェアドオール(SA)方式は，複数のノードに対して，物理的もしくは論理的に 1 つのディスクを共有する方式である。図 4 では，システムが保持しているファイル A,B,C,D を全ノードが共有している。

この方式ではすべてのノードがすべてのファイルにアクセスできるためにノードごとの負荷の偏りは起こりにくい。更に，新しいノードを加えた場合も特別な処理は必要なく，ファイルの分割状態を把握する必要はない。しかし，その一方，全てのノードが 1 つのディスクにアクセスするので，ディスクアクセスがボトルネックになる。また，ノード間でファイルの相互排他処理を行わなければならないため，SN 型に比べて処理効率やスケーラビリティに劣る可能性が高い。更に，ノード間で最新の情報を共有するためには，アクセスごとにディスクに書き戻さねばならないという欠点もある。

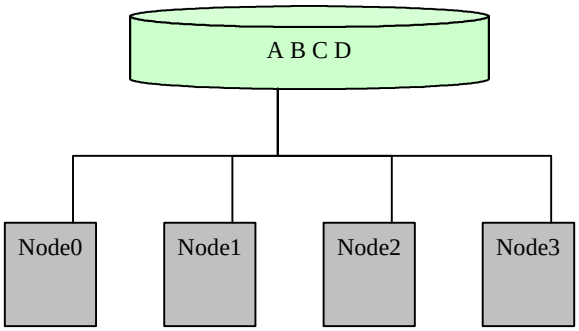


図４　シェアドオール方式

４．シェアド･キャッシュ･モデル

本稿で提案するトランザクション処理モデルである，シェアドキャッシュ(SC)モデルについて説明する。

SC モデルは，図 5 に示すように,プロセスとディスクの間に，プロセス間で共有するデータキャッシュを備えていて，ノード内キャッシュを持つことで性能を上げるという SN モデルの利点と，全てのノードからアクセスできるという SA モデルの利点を兼ね備えたモデルになっている。

ハードウェア構成としては，SA 型のように全ノードが共通のディスクを持つ形でも SN 型のように各ノードが個別に持つ形でも，どちらでも良い。ここでは，SA 型の一番の利点である，“ 全ノードからのアクセス ”は共有キャッシュで実現させるため，SN 型のような構成を取ることとする。

共有キャッシュへは全てのノードからアクセスができるが，キャッシュへのファイル読み込みやキャッシュからファイルへの書き込みは，そのファイルの入出力担当ノードのみが行える。ファイルへアクセスするノードを限定することでノード内にデータをキャッシュでき，性能向上が期待できるし，ディスクではなくメモリの段階でノード間共有をすることができる。ただし，前述の通り，SDSM はノード内外によって共有メモリへのアクセス速度が異なるので，リクエストの処理に必要なファイルを管理しているノードがその処理を行うことが望ましい。そのため，リクエストをクライアントから受け取り，その処理に適切なノードを決定し，そこへ送信するフロントエンド(FE)マシンを置き，そのマシンがリクエストを分配する。図 5 において，FE マシンがクライアントとの通信やリクエストの送信を行い，バックエンド(BE)マシンがキャッシュの管理とリクエストの処理を行う。

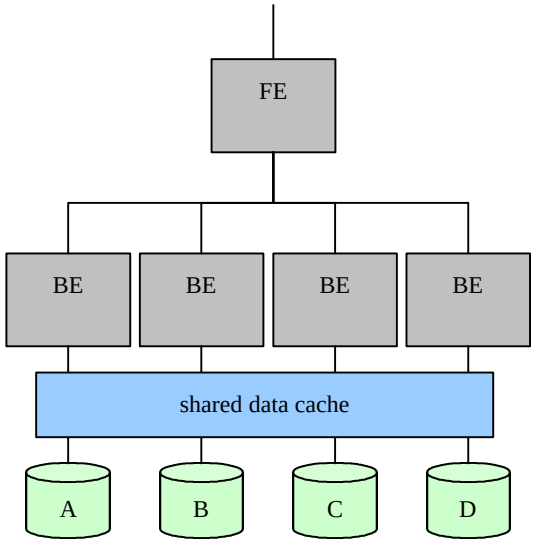


図５　シェアドキャッシュモデル

５．オークションシステムの実装

SC モデルの応用として，オークションシステムを扱うサーバを実装した。このシステムは，外部プロセスによって生成されたリクエストを処理し，その結果を外部プロセスへ返す。このシステムはオークションにおける基本的な操作である，検索・入札・出品を実行できる。検索はキーワードと製品 ID による検索を実行できる。ユーザインターフェースは，オークションシステムとは別に用意されている Web サーバが提供するとして，本システムはそこからリクエストが送られてくることを前提としている。

5．1　リクエスト送信

FE は，送られてきたリクエストを解析して必要なファイルの担当ノードを決定して，通常はそのファイルの入出力を担当しているノードへリクエストを送信する。しかし，担当ノードが一定数以上のリクエストを保持していて高負荷状態の場合は，そのノードへの送信は失敗する。その場合は，FE は全ノードに対してラウンドロビン型に成功するまで順次リクエストの送信を試みる。

5．2　リクエスト処理

BE は，リクエストを受け取ったら必要なファイルをキャッシュへ読み込み，そのリクエストに対応した処理を行って，結果をフロントエンドへ返す。

5．3　キャッシュ管理

SDSM ではノード内外でメモリアクセス速度が異なることを考慮して，図 6 のように，共有キャッシュを分割して各ノードごとに管理する。また，図 6 下部はノード内部でのキャッシュ構造である。このようにキャッシュにはスロットが複数用意され，そこへファイルを入れる。ファイルサイズは固定で，スロットサイズと等しく，必ずスロットに入りきるものとした。

キャッシュはテーブル部とデータ部に分かれ，テーブル部にはスロットに含まれるファイル名を，データ部にはそのファイルの内容を，それぞれ保存する。この分割の結果，データ部に排他ロックをかけた状態でも，テーブル部を用いてスロットに含まれるファイルを把握することができる。

キャッシュの管理手順は以下の通りに行う。

① テーブルに対して共有ロックをかけ，各スロットに含まれているファイルを調べる。
② ファイルが既にキャッシュされていることが分かったらそのスロットのデータ部へ，実行する操作に適したロック，つまり，読み込み操作に対しては共有ロックを，書込み操作に対しては排他ロックを，かける。(処理終了)
③ どのスロットにも含まれていなかった場合は，テーブルのロックを排他ロックに変更する。
④ 前回のキャッシュ読み込みからアクセス回数が最も低かったスロットを新しい読み込みスロットに決定する。
⑤ この読み込みスロットにファイルが入っていて，既に書き込みが行われていたときは，スロット内のデータをファイルへ書き戻し，必要なファイルをそのスロットへ読み込ませる。(処理終了)

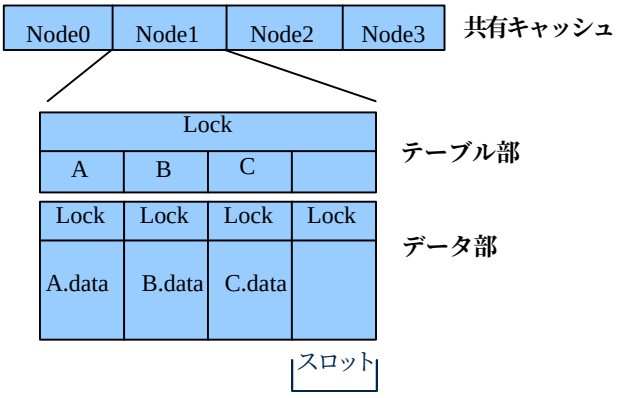


図６　キャッシュ構造

5．4　実装に用いる SDSM

著者の手元にある，実行可能な SDSM は，SMS と TreadMarks の 2 種であるが，両者はメモリの一貫性モデルが違い，提供している機能も異なる。両者を本システムの実装の際に必要となった機能に関して比較すると以下のようになる。

- ロックの多重化

本システムでは複数ノードに含まれるプロセスに対するキャッシュの制御のために，ロックの多重化が必要であるが，SMS はこれをサポートしていない。また，効率的な動作のために排他ロックのみならず共有ロックもあることが望ましいが，TreadMarks は排他ロックのみしかない。

- 外部通信機構

実装するオークションシステムは，外部プロセスによって生成されたリクエストを受け取り，処理結果を外部プロセスへ返す。従って，socket のような，外部通信機構が必要となる。socket は，そこへ入力があったことをシグナルによって知る。一方 SDSM でも，他ノードが共有領域にアクセスしたことを知るために，シグナルを用いているので，それらのシグナルを区別できなくてはならない。

表 1 は，これらの機能について，両者の対応状況をまとめたものである。いずれも望まれる機能全てを満足していないので，ここでは，キャッシュ管理には多重ロックが殆ど不可欠なことを考慮して TreadMarks を利用することにした。それに伴い，共有ロックや socket の利用はせずに排他ロックやファイルとのやり取りで代替したので性能的には不利な状況になっている。

表 1　SDSM の機能対応

SDSM バージョン	SMS 0.4.19a	TreadMarks 1.0.3.3
ロックの多重化	×	○
共有ロック	○	×
socket	○	×

6．評価

6．1　比較対象

比較対象として，SN 型と SA 型を擬似的に実行できるオプションを実装したシステムに追加した。SN 型実行では，各ノードのローカルディスクにファイルを保存し，リクエスト送信は必ずファイル入出力の担当ノードへ送る。キャッシュ構造はそのまま利用するが，リクエストが担当ノード以外に送られることがないため，排他制御はノード内だけで行なう。SA 型実行では，NFS を用いてディスクへファイルを保存し，リクエスト送信は，全ノードにラウンドロビン型に送る。また，排他制御は，ファイルごとに，SDSM を利用したロックをかけるようにした。

6．2　リクエストパターン

表 2 に示すような 4 種類のそれぞれ偏り方が異なるリクエストパターンについて測定を行った。表 2 では，○が偏りがあることを示し，×が偏りがないことを示す。

ここで，ノード偏りとは，BE ノード間で処理を行うリクエスト数に偏りがあることを示し，ファイル偏りとは，同一ノード内において特定のファイルにアクセスが集中することを示す。

表２　リクエストパターン

	ノード偏り	ファイル偏り
Base	×	×
Node	○	×
File	×	○
Both	○	○

6．3　測定環境

測定は，表 3 の環境であるノードのクラスタを利用し，表 4 に示すパラメータを用いて行った。表 4 におけるスロットサイズとは 1 スロット中に含まれる製品データの個数を，キューサイズはキューの中に置くことができるリクエストの個数を表す。

表３　実行環境

CPU	Intel PentiumIII 1.13GHz　*2
メモリ	1GB
OS	RedHat Linux 9.0
カーネル	2.4.20-6smp
SDSM	TreadMarks 1.0.3.3
ネットワーク	Gigabit Ethernet

表４　パラメータ設定

ノード数	フロントエンド	1
	バックエンド	2
キャッシュスロット数		2
スロットサイズ		1000 個
リクエストキューサイズ		100 個
リクエスト数		1000

6．4　実行結果

- 総実行時間

全リクエストを処理するのに要した時間の平均値を表 5 に示す。この表から SC 型と SN 型は，SA 型に比べて実行時間が長く，大きなばらつきがあることが分かる。

実行時間については，SA 型がファイル 1 つに対して 1 つのロックを取っているのに対して，SC 型，SN 型はテーブル部とデータ部という 2 つのロックを取って処理を行うという違いが現れている。つまり，共有ロックがないという実装上の制約のために，同じノード内の異なるファイルに対して同時にアクセスすることができなくなってしまって，実際にはほぼ逐次に動作しているため SA 型に比べて遅くなっていると考えられる。

具体例を図 6 に示す。図 6 では，異なるプロセスが同時期にそれぞれファイル A とファイル B にアクセスしようとした場合である。ファイル A もファイル B もすでにキャッシュに含まれているため，本来ならばテーブルには共有ロックをかければよい。しかし，実装 SDSM の制約から排他ロックがかけられるため，本来ならば競合せずに並行処理できたはずだった二つのアクセスが逐次化されてしまう。

また，実行時間が SA 型に比べて大きいのは，SA 型がファイルの一部だけを読み書きしているのに対し，SC 型と SN 型は全体の読み書きをしていることも理由として考えられる。すなわち，キャッシュにおいてファイルを頻繁に変更する場合にはほとんど変更していないファイルを書き戻すという無駄な動作によって余分な時間をかけてしまっている可能性がある。

リクエストの送信方法や結果の返し方はすべての型で同一の方法を用いていること，SC 型と SN 型がテーブルを用いた多重ロックを使っていること，これらの理由からこの結果も多重ロックによるものだと考えられる。総実行時間の比較によって，SA 型が SC 型と SN 型とは明らかに異なることが分かったので，今後は，主に SC 型と SN 型とを比較する。

表５　総所要時間

(秒)

	SC		SN		SA	
	平均	偏差	平均	偏差	平均	偏差
base	43	9	47	20	21	13
node	79	30	85	40	13	1
file	44	29	80	74	10	0
both	118	47	107	41	12	1

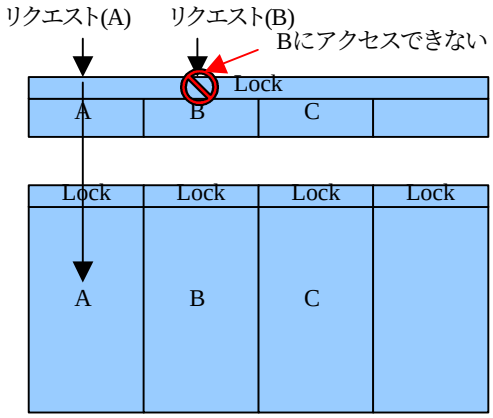


図６．１　テーブルの競合による逐次化

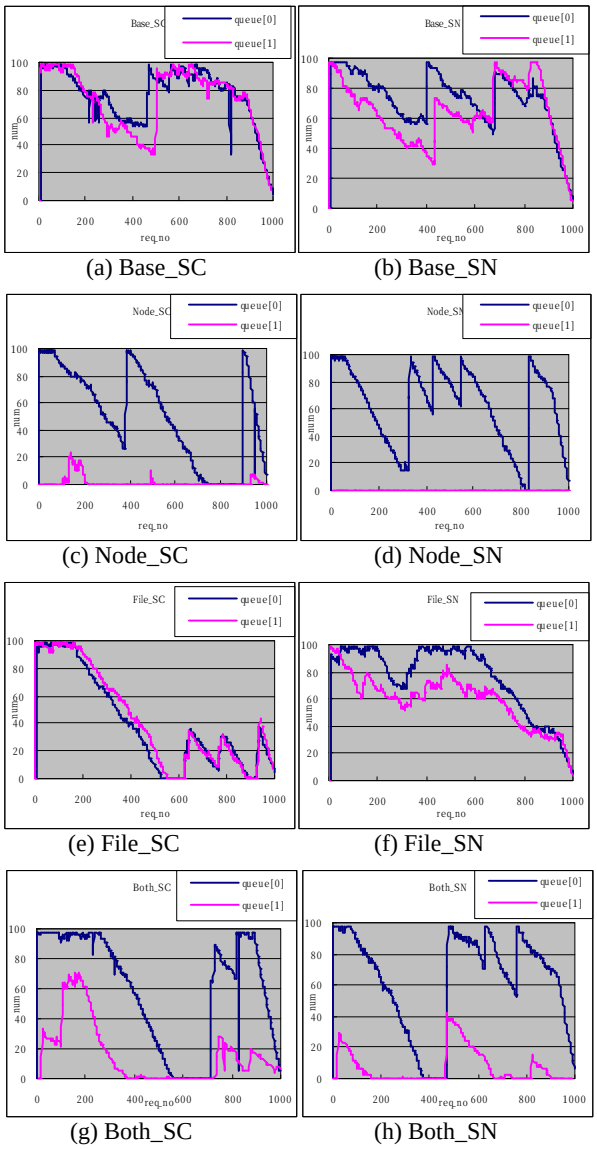
- 処理の分散

各バックエンドにおける保持リクエスト数の動きを図 7 に示す。この図は，バックエンドのプロセスが，リクエストを取り出した直後に保持しているリクエスト数を示している。

リクエストパターンが Node の場合，SN 型では片方のノードがほとんどの処理を行っているのに対して，SC 型では片方のノードのリクエストキューが埋まった直後にもう 1 つのノードへ送信が行われていることが分かる。同様に，Both の場合も 1 台に処理が集中したところでもう 1 台へ負荷を分散できている。

次に，表 6 に FE と BE 両方について再転送の回数を示した。表 6 は，FE において転送を行ったリクエストの個数と BE において担当ノードへ再転送を行ったリクエスト数を示している。ただし，SN 型は必ず担当ノードが処理するため，また，SA 型はどのノードも全てのファイルにアクセスできるため，どちらも BE 間での転送は起こらない。

SC 型では読み込みが必要なファイルが別ノードにある場合に BE 間でのリクエストの転送が有って，この BE 間の転送が多ければそれだけ無駄な処理が発生していることになる。表 6 では，Base と Node といった，アクセスファイルに偏りがない場合には約 20%の割合で再転送が行われていることが分かる。しかし，ファイルに偏りがある File と Both では確率は約 5%と小さい。これらの値が大きいか小さいかに関しては，BE で行われる処理によって評価が異なってくるため，詳細な評価は改めて後述する。



(a) Base_SC (b) Base_SN

(c) Node_SC (d) Node_SN

(e) File_SC (f) File_SN

(g) Both_SC (h) Both_SN

図６．２　SC 型と SN 型における保持リクエスト数

表６　転送されたリクエスト数

	SC		SN		SA	
	FE	BE	FE	BE	FE	BE
Base	61	17	18	0	120	0
Node	90	16	17	0	123	0
File	42	0	30	0	114	0
Both	80	6	26	0	134	0

● ターンアラウンド時間

1 つのリクエストに対して，FE がそのリクエストを受け取ってから，BE へ送信し，BE での処理結果をクライアントへ返すまでの時間について表 7 に示した。

SC 型は，Base では SN 型よりも時間がかかっているが，Node，File，Both のようにリクエストに何らかの偏りが生じている場合には安定して早い反応をしている事がわかる。

File，Both のように，ファイルに偏りがある場合は，担当ノード以外に送られた場合でもキャッシュ内に必要なファイルがある可能性が高いため，SN 型に比べて非常に速くなっている。Node の場合，キャッシュ内に必要ファイルがあるかどうかが不確かな状態で送られてくるが，SN 型に比べて明らかに速くなっている。これらのことから，前述の再転送の効果は得られていると考えられる。

表７　ターンアラウンド時間

(秒)

	SC		SN		SA	
	平均	偏差	平均	偏差	平均	偏差
base	7.87	4.93	5.83	2.81	3.55	1.20
node	11.21	5.12	19.74	10.60	3.81	1.73
file	4.27	2.81	14.30	18.46	2.62	0.80
both	5.94	2.64	7.12	4.04	3.36	1.04

6．5　トランザクション処理のための SDSM 機能

6.4 の結果を受け，本システムのようなトランザクション処理に適用するために望まれる SDSM の機能について述べる。

(1)　必要不可欠と思われる機能

外部通信機構と多重ロックが必要なことが分かった。

● 外部通信機構

外部と通信するためにはソケットのような機構が必要となるが，SDSM では，入力があったことを知らせるシグナルを含めてユーザが取り扱うシグナルは全て SDSM システムによって感知される。これは共有メモリの管理にシグナルを利用しているためであるが，ソケットを用いた外部通信を行うには，SDSM のシステムが共有メモリを管理するためのシグナルとユーザ側の応用へ渡すためのシグナルとを区別して，外部通信等の SDSM の制御用ではないシグナルを応用へ適切に渡すことが，この種の応用のサーバに SDSM を用いるために不可欠となる。

● 多重ロック

多重ロックは，実装する応用によっては必要とは限らないが，多くのトランザクション系応用で必要になる可能性があり，その場合に SDSM がサポートていないとその応用の実装ができなくなってしまう。SMS と TreadMarks では，SMS が未対応，TreadMarks が対応である。

また，SMS では，ロックを得たときに共有領域のコピーを作り，次のプロセスがロックを得たときにそのコピーと自身が変更した内容を比較して，変更内容だけを他プロセスに知らせる仕組みとなっている。これは，ロックをかけた場合のコピーの取り方という，メモリの一貫性モデルのポリシーの問題となるため，単純に対応させることは難しいと考えられる。

(2) 効率化のための機構

- 共有ロック

共有ロックの実現は効率的な実行のために最も重要な要素だと考えられる。特に読み込みの多いシステムを実装することを考えたときに，この機能が使えないと，本研究で実装したオークションシステムのように効率が非常に悪くなる。これについては，通常のロック機構が実現できていれば簡単に実現できる機能であると思われる。

- 条件変数

SC モデルでは，1 ノードごとにリクエストを保存するから，リクエストキューの管理のための条件変数はノードの数だけ必要となる。これに対して SMS では 8 個，TreadMarks では 4 個の条件変数を用いることができるが，この程度の数ではこの個数がシステムの拡張性のボトルネックとなる可能性がある。

- 一部プロセスによる一貫性同期

最後に，一貫性同期について述べる。本研究で対象としたようなページベース SDSM では，すべてのプロセスが 1 つもしくは少数の共有データに対して処理を行うことを前提としている。そのため，バリア機構はすべてのプロセスの参加が前提となり，必要のないデータも受け取らなくてはいけないという余分な作業が含まれる可能性がある。MPMD 型プログラムとの関連を考えたときに，一部のプロセス群による一貫性同期機能が性能向上のために必要になってくるのではないかと考えられる。

７．まとめ

本稿では，クラスタを用いたトランザクション処理サーバの負荷分散を目的に，SDSM を効率的に使える処理モデル(SC)を提案し，オークションを題材としたその実装結果を示した。擬似的に SN 型や SA 型を実行できるようにして，ターンアラウンド時間や負荷の分散を評価軸として，簡単な初期実験を行った。

SC 型の実行結果は，実装の制約から SA 型よりも劣る結果となったが，条件が等しい SN 型に比べると，高速に動作した。また，共有キャッシュを利用することで，ノード間の負荷の偏りを軽減させることができた。

また，SDSM を用いたトランザクションシステムの実装では，多様なロック機能やソケット機構などが重要になるものの，現状の SDSM のこれらの機能サポートは不充分であり，機能拡張の余地があることが明らかになった。

このような応用を扱う場合，外部的には 1 つのシステムのように見せながら内部に複数のディスクドライブとキャッシュメモリなどを備えたファイルサーバも開発されているため，単純にファイル入出力の高速化というだけでなく，付加的な処理を加えたシステムとしての特徴を出す必要がある。

また，本稿では既存の SDSM を全く手を加えずに用いたが，このようなファイルアクセスを主体とした処理には SDSM 内部で用いている，ファイルのメモリマップ機構を直接使ったほうが効率的であることが分かっており，今後このような，応用を意識したファイルのキャッシュ機構を主軸に置いた SDSM の開発が期待される。

参考文献

[1] 緑川，飯塚 : "ユーザレベル・ソフトウエア分散共有メモリ 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.SIG9(HPS 3), pp.170-190 2001

[2] P. Keleher, et al.: TreadMarks : Distributed Shared Memory on Standard Workstations and Operating Systems, Proc. Winter 94 Usenix Conf., pp.115-131, (1994).

[3] M.R.Eskicioglu, et al : Evaluation of the JIAJIA Software DSM System on High Performance Computer Architectures, Proc. 32nd Hawai Int. Conf. System sciences, 1999

[4] T.Schroeder, S.Goddard, and B.Ramamurthy: Scalable Web Server Clustering Technologies, IEEE Network•May/June 2000