マルチメディア通信と分散処理　85－1
（1997．11．6）

キャッシュコヒーレンスプロトコルにおけるオーナーシップに関する考察

山本伸介*，中嶋卓雄*，中村良三*

*熊本大学工学部

近年，複数の異機種計算機をネットワーク上で有機的に結合し，計算機資源の有効活用と複雑な問題に対する解決能力を目的とした分散処理技術が活発に研究されている．分散共有メモリ上におけるローカル・キャッシュモデルは，分散共有メモリシステム上のプロセッサの結合型に依存するが，実際のネットワーク環境において効率の良いローカル・キャッシュモデルとその評価はなされていない．

様々に結合されたローカル・キャッシュを持つ分散共有メモリシステムおいて，効率の良いルーズなキャッシュコヒーレンスプロトコルの開発を行うため，本稿では特に，データブロックのオーナーシップに注目してバス結合型について新しいモデルを提案し，その評価を行った．

Discussion about Ownership in Cache Coherence Protocol

Shinsuke Yamamoto*,
Takuo Nakashima*,Ryozo Nakamura*

*Faculty of Engineering, Kumamoto University

In a distributed system, the shared memory technique has been useful for the natural extension of single computation. A distributed shared memory system with local cache depend on the connection type of it's processors. But in a practical network system, there is no efficiency model of local cache on a distributed shared memory system. We propose the design and development of efficiency cache coherence protocol with relax consistency model on a distributed shared memory system. In this paper, we notice the ownership of data-block, show the new protocol model and evaluation of bus connection type system.

1 はじめに

近年，コンピュータの小型化・高性能化にともない，複数の異種計算機をネットワーク上で有機的に統合し，計算機資源を有効に活用し，より複雑な問題を解決する分散処理技術が活発に研究されている．

特に、分散共有メモリ (DSM) に関する研究の中で，実際のネットワーク環境でのローカル・キャッシュモデルにおける効率の良いキャッシュコヒーレンスプロトコルに関する研究が行われている．ローカル・キャッシュモデルは，プロセッサのネットワーク結合型に依存して動作し，その主な種類として，バス結合型，トークン型，メッシュ型などがあげられる．

一般的には共有データを保持するメモリとキャッシュ間のデータ転送はキャッシュ-キャッシュ間のデータ転送よりも時間がかかる．そこでプロトコルによっては，各データブロックにオーナーシップを設けて，共有データを保持するメモリへの書き戻しの頻度を低下させ，システム全体の速度向上を図っている．このオーナーシップは，代表的な分散共有メモリモデルである read-replication モデルにも採用されているが，データブロックの保持に関して，read-replication モデルは分散型，ローカル・キャッシュモデルは，集中・分散型となる．また，オーナーシップの管理に関しては，read-replication モデルは特定のマシンが行うのに対し，ローカル・キャッシュモデルはオーナー自身が行う．

本研究では，様々な形で結合された分散共有メモリシステムにおけるローカル・キャッシュのモデルについて考察し，実際のネットワーク環境において，効率の良いルーズなキャッシュコヒーレンスプロトコルを新たに開発し，その評価を行うことを目的とする．

本稿は，その前段階として，分散共有メモリシステムの結合型として最も単純であるバス結合型に着目し，そのキャッシュコヒーレンスプロトコルについて考察する．従来のバス結合型におけるキャッシュコヒーレンスプロトコルは，厳しいデータ一貫性を保持するもので，そのデータ一致のタイミングにより (1) ライトスルー，(2) ライトバックに，更新されたデータブロックに対する処理により (a) データ無効化型，(b) データ更新型とに分類される．今回は，ライトバック，データ無効化型プロトコルである Berkeley プロトコルおよびライトバック，データ更新型プロトコルである Dragon プロトコルについて考察し，データブロックのオーナーシップに注目して，新しいモデルを提案する．また，従来のプロトコルとの比較・考察を行うとともに，キャッシュコヒーレンスプロトコルの評価法についても考察を加える．

2 バス結合型におけるプロトコル

バス結合型マルチプロセッサの特徴は，全てのデータ転送が 1 本の共有バスを介して行われる点である．つまり，共有バス上のデータの流れを監視 (snoop) することで，システム全体の挙動をほとんど把握できる．そこで，ローカル・キャッシュが共有バスを監視し，主記憶およびローカルキャッシュの間で内容の一致を保証する．

2.1 従来のプロトコルの分類

従来のキャッシュコヒーレンスプロトコルは，主記憶に最新のデータを書き戻すタイミングとその後の処理によって表 1 のように分類できる．

表 1 ：従来のプロトコルの分類

ライトスルー	データ無効化型	
	データ更新型	
ライトバック	データ無効化型	Synapse プロトコル Illinois プロトコル Berkeley プロトコル Write Once
	データ更新型	Firefly プロトコル Dragon プロトコル

2.2 オーナーシップ

Illinoisプロトコルにおいては，データブロックに書き込まれた場合，必ず1度主記憶に対して書き戻しを行う．しかし，一般的には主記憶-キャッシュ間のデータ転送はキャッシュ-キャッシュ間のデータ転送よりも時間がかかるため，データが更新されるたびに主記憶に最新のデータを書き戻すのはロスが大きい．そこで各データブロックに「オーナーシップ」を設けて，主記憶への書き戻しの頻度を低下させ，システム全体の速度向上を図る．

なお，各データブロックのオーナーは，そのブロックに対する他からの要求に対してデータブロックを供給し，主記憶への書き戻しについても最終的な責任を持つ．

3 基本プロトコル

3.1 特徴

表1に示すように，Berkeleyプロトコルはライトバック，データ無効化型，Dragonプロトコルはライトバック，データ更新型のキャッシュコヒーレンスプロトコルである．

Berkeleyプロトコル，Dragonプロトコルはオーナーシップを利用し，少ない状態数のもとで，主記憶との転送頻度を最小限におさえ，データブロックの転送によるロスを低減している．また，Dragonプロトコルについては，データ更新型であることから，一度でもアクセスのあったキャッシュにおいてI状態になることはなく，立ち上がり時以外はI状態を設ける必要がない．

3.2 状態

Berkeleyプロトコルでは，それぞれのキャッシュについて，データブロックとオーナーシップの有無により以下の4つの状態に分類する．

- I(Invalid). データブロックを保持しない．
- SN (Shared-Nonowner). データブロックを持ち，そのコピーが他に存在するかもしれず，オーナーでない．
- EO (Exclusive-Owner). データブロックを持ち，そのコピーが他に存在せず，オーナーである．
- SO (Shared-Owner). データブロックを持ち，そのコピーが他に存在するかもしれず，オーナーである．

また，Dragonプロトコルでは，上記のI, SN, EO, SOの4状態にENを追加した5状態に分類する．

- EN (Exclusive-Nonowner). データブロックを持ち，そのコピーが他に存在せず，オーナーでない．

3.3 状態遷移

Berkeleyプロトコルの状態遷移図を図1に示す．なお，図1における実線はローカルなプロセッサからのアクセスを示し，破線は共有バスからのアクセスを示す．

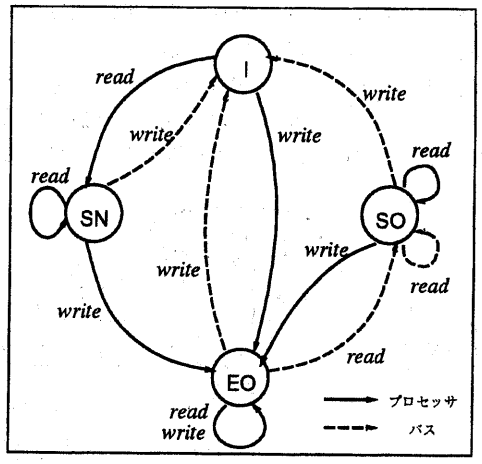

図 1: Berkeleyプロトコルの状態遷移図

Dragonプロトコルの状態遷移図を図2に示す．なお，図2における実線はローカルなプロセッサからのアクセスを示し，破線は共有バスからのアクセスを示す．また，(Y)は他のキャッシュにデータブロックのコピーが存在することを示し，(N)は存在しないことを示す．

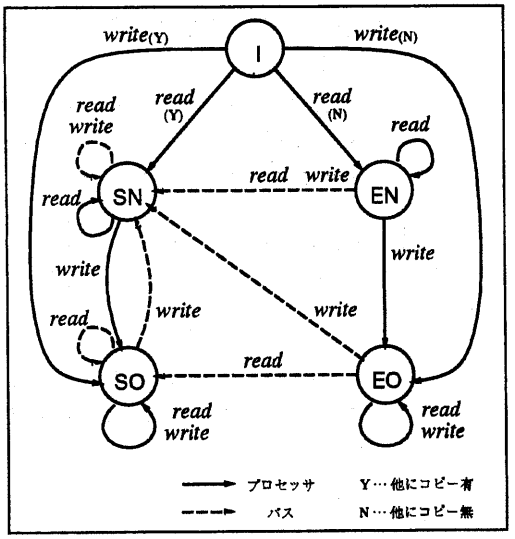

図 2: Dragon プロトコルの状態遷移図

3.4 問題点

一般的に，主記憶-キャッシュ間の転送は，キャッシュ-キャッシュ間の転送より時間がかかる．Berkeley プロトコルや Dragon プロトコルでは，オーナーシップを用いることで，他のプロトコルより主記憶への書き戻しの頻度を低くし，プロトコル全体の効率向上を図っている．

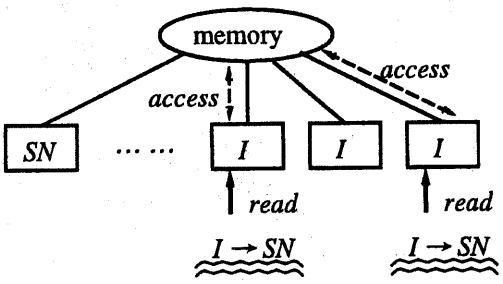

図 3: 従来のプロトコルの問題点

しかし，初期状態からの連続した読み込み要求が発生した場合や，オーナーを持つキャッシュにおいて，ブロックの追い出しなどによりオーナーが主記憶に移動した後の連続した読み込み要求が発生した場合には，図 3に示すように，SN と I による状態となる．ここで I 状態のプロセッサからの読み込み要求が繰り返し発生した場合，主記憶-キャッシュ間の転送が毎回行われることになる．

4 提案するモデル

4.1 基本概念

本稿では，SN と I による状態からなる場合における I 状態からの読み込み要求に対するアクセスの高速化を目的として，代表的なキャッシュコヒーレンスプロトコルである Berkeley プロトコルと Dragon プロトコルを改良する．

一般的にキャッシュにおけるアクセス速度を向上させるためには，出来るだけ多くのコピーブロック (SN 状態) が存在すればよい．従来の Berkeley プロトコルおよび Dragon プロトコルにおいて，SN 状態が増加する場合は，

1. 主記憶から直接読み込む (SO 状態が存在しない場合)
2. SO 状態のキャッシュから読み込む

の 2 通りである．しかし，一般的に，主記憶-キャッシュ間の転送は，キャッシュ-キャッシュ間の転送より時間がかかるため，本稿では主記憶からの読み込み (1 の場合) をキャッシュからの読み込み (2 の場合) で置き換えることにより，システム全体の速度向上を図る．

4.2 SNO と SNN

従来のプロトコルにおける SN 状態を，以下のように SNO と SNN の 2 つの状態に分類する．

- SNO (Shared-Nonowner-Other)

  データブロックのコピーを持ち，そのオーナーではない．また，他キャッシュにそのブロックのオーナーが存在する．
- SNN (Shared-Nonowner-No)

  データブロックのコピーを持ち，そのオーナーではない．また，他キャッシュにそのブロックのオーナーが存在しない．

以上のようにSN状態を分類することにより，あるキャッシュにおいて，目的のデータブロックのオーナーシップを意識した処理が可能となる．これによるオーナーシップの効率の良い移動は，主記憶-キャッシュ間のデータ転送をキャッシュ-キャッシュ間のデータ転送に置き換えることを可能とする．

4.3 提案するプロトコル

SNOとSNNの状態遷移に関するシステム全体の流れとして，図4における状態遷移(1)を図5に，状態遷移(2)を図6に示す．

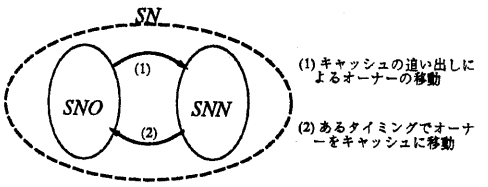

図 4: SNO と SNN の状態遷移

まず，図4における(1)の状態遷移のタイミングは，キャッシュにオーナーが存在した状態から存在しない状態への遷移である．つまりオーナーであるキャッシュにおいて，そのデータブロックに追い出しが発生した場合にSNNメッセージを発行し，状態が遷移する(図5)．

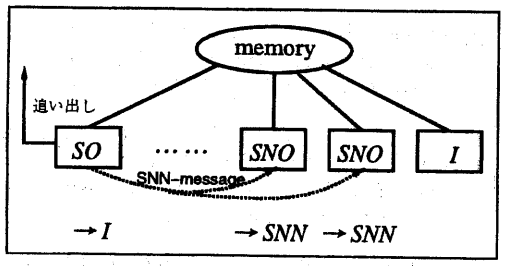

図 5: SNN メッセージ

次に，図4における(2)の状態遷移のタイミングは，キャッシュにオーナーが存在しない状態から存在する状態への遷移である．つまり，あるキャッシュにおいて読み込み要求が発生した時，そのキャッシュがSNN状態である，すなわちキャッシュにオーナーが存在しない状態であれば，キャッシュにオーナーを移動させることが可能となる．そこで，そのキャッシュにオーナーを移動させ，SNOメッセージを発行することでオーナー移動が完了する(図6)．

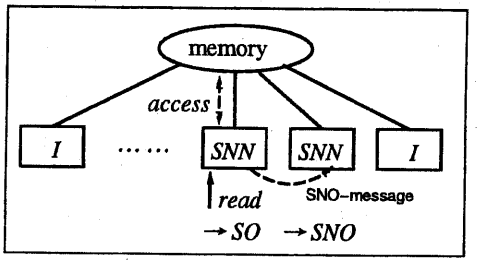

図 6: SNO メッセージ

5 評価

まず，Berkeley プロトコルの状態遷移マトリックスを導出する．更に，提案するプロトコルの状態遷移マトリックスを導出し，2つのプロトコルの効率について，比較・考察する．

ただし，

- S : 共有メモリシステム上のホストの数．
- n : データブロック数．
- r : 読み込みと書き込みの割合．r 回のアクセス (読み込み，書き込み) に対して1回の書き込みが起こる．
- e : データブロックの複製がすでに存在する確率．
- d : データブロックのオーナーシップをキャッシュが持つ確率．

5.1 Berkeley プロトコル

以下に，導出した Berkeley プロトコルの状態遷移マトリックスを示す．

$$\begin{bmatrix} I \\ SN \\ EO \\ SO \end{bmatrix} = \begin{bmatrix} \frac{S-1}{Sn} & \frac{S-1}{Snr} & \frac{S-1}{Snr} & \frac{S-1}{Snr} \\ \frac{r-1}{Snr} & \frac{r-1}{nr} & 0 & 0 \\ \frac{1}{Snr} & \frac{1}{Snr} & \frac{1}{Sn} & \frac{1}{Snr} \\ 0 & 0 & \frac{(S-1)(r-1)}{Snr} & \frac{r-1}{nr} \end{bmatrix} \begin{bmatrix} I' \\ SN' \\ EO' \\ SO' \end{bmatrix}$$

5.2 改良 Berkeley プロトコル

以下に，導出した改良 Berkeley プロトコルの状態遷移マトリックスを示す．(左辺省略)

$$\begin{bmatrix} \frac{S-1}{Sn} & \frac{S-1}{Snr} & \frac{S-1}{Snr} & \frac{S-1}{Snr} & \frac{S-1}{Snr} \\ \frac{(r-1)(1-d)}{Snr} & \frac{(S-1)(r-1)}{Snr} & \langle SNN \rangle & 0 & 0 \\ \frac{d(r-1)}{Snr} & \langle SNO \rangle & \frac{r-1}{nr} & 0 & 0 \\ \frac{1}{Snr} & \frac{1}{Snr} & \frac{1}{Sn} & \frac{1}{Snr} & \\ 0 & \frac{r-1}{Snr} & 0 & \frac{(S-1)(r-1)}{Snr} & \frac{r-1}{nr} \end{bmatrix} \begin{bmatrix} I' \\ SNN' \\ SNO' \\ EO' \\ SO' \end{bmatrix}$$

5.3 状態遷移マトリックスの考察

改良 Berkeley プロトコルは，従来の Berkeley プロトコルよりも状態数が 1 つ増えているため，改良 Berkeley プロトコルのマトリックスの 2 行および 2 列を除いて比較すると，共通部分が非常に多い．

まず，2 列目に注目すると，状態数の増加により，I，EO に遷移する確率が妥当に増加しているのに対して，SNO(従来の SN) に遷移する場合は SNO メッセージが発行された場合のみの遷移となり，明らかに確率が減少している．また，3 行 1 列のパラメータも，従来のものより低下している．これより改良 Berkeley プロトコルにおいては，既存のプロトコルに比べて，SNO(従来の SN) に状態遷移する確率が低くなったといえる．

また，5 行 2 列のパラメータにおいて，従来のプロトコルでは，SN 状態から SO 状態に状態遷移する場合はなかったのに対し，改良 Berkeley プロトコルでは，SNN から SO となる状態遷移が可能となっている．これより，既存のプロトコルに比べ，SO に状態遷移する確率が高くなったといえる．

以上より，SNO(従来の SN) に遷移していた特定の状態が，SO に状態遷移することが可能となり，結果的にプロトコル全体で SO に状態遷移する確率が高くなったといえる．これにより，従来よりも SO になる確率が高い，すなわちオーナーがキャッシュに移動しやすくなったといえ，これにより特定の場合の主記憶-キャッシュ間のデータ転送をキャッシュ-キャッシュ間のデータ転送により置き換えた処理が可能となる．これは他キャッシュのオーナーシップを意識した処理を実現したともいえる．

6 おわりに

今回は，分散共有メモリシステムにおけるローカル・キャッシュモデルについて考察した．特に，バス結合型のシステムにおいて，厳しいデータ一貫性を保つ従来のキャッシュコヒーレンスプロトコルについて改良し，その評価を行った．これにより，ある特定のアクセス傾向の強いシステムにおいて，改良したキャッシュコヒーレンスプロトコルの有効性を評価でき，さらにはキャッシュコヒーレンスプロトコルの評価法についても検討できた．

7 今後の課題

今後の課題として，今回検討したバス結合型システムのみならず，実際のネットワーク環境を念頭においたシステムについての考察があげられる．また，現実のネットワークシステムにおいて有用であると思われる，効率の良いルーズなキャッシュコヒーレンスプロトコルについて考察・評価していきたい．

参考文献

[1] Micahel Stumm, Songnian Zhou:Algorithms implementingdistributedsharedmemory,*IEEE Comput. Mag.*, vol.23, No.5, May(1990).

[2] S. Zhu, M. Stumm, K. Li, and D. Wortman:Heterogeneous Distributed Shared Memory, *IEEE Trans. on Para. and Distri. Sys.*, vol.3, No.5, September(1992).

[3] 小柳滋，久保田和人，川倉康嗣:並列プログラムデバッグのための可視化ツール，情報処理，vol.37, No.7, pp1219-1307(1996).

[4] 天野英晴:並列コンピュータ (Parallel Computers)，情報系教科書シリーズ第 18 巻，昭晃堂，1996.

[5] R.H.Katz, et.al:Implementing a Cache Consistency Prot ocol, ISCA85, pp276-283(1985).