

ユーザ・レベル・ライブラリのみによる分散共有メモリサーバ構築の試み

5B-5

中山 泰一 赤澤 忠文 (電気通信大学情報工学科)

1 まえがき

分散共有メモリに関する研究が近年盛んに行われている[1]。本稿では、ネットワークにより結合された複数のUNIXワークステーションにおいて分散共有メモリをサポートするための機構を構築することを試みる。具体的にはこの機構を、数台のSPARCワークステーション上で試作したが、UNIXが提供している通信システム・コールのみを用いて書かれたユーザ・レベル・ライブラリとして実現されているので、他機種への移植、システムの拡張なども容易に行えるものである。

ネットワークにより結合されたシステムの場合、通信のオーバーヘッドがきわめて大きく、できるだけ通信の回数が少なくなるように設計することが求められる。本稿では、サーバが保有している共有メモリのデータを各クライアントがキャッシュするとともに、キャッシュのコヒーレンスを保つための通信もできるだけまとめて通信回数を削減できるようにシステムを設計し、試作、評価を行った。

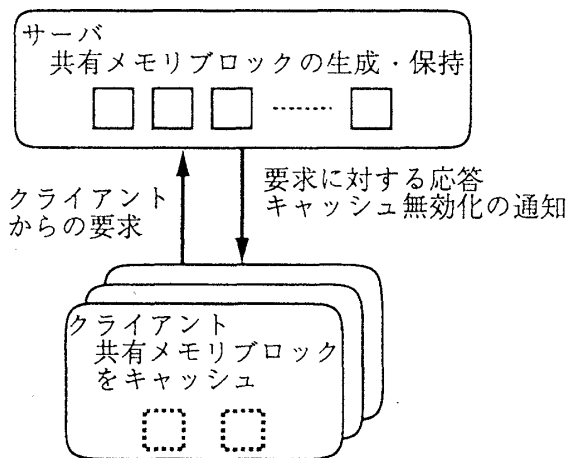


図1: システム構成の概略

A Study of a Distributed Shared Memory Server based on User-level Libraries by YASUICHI NAKAYAMA and TADAFUMI AKAZAWA (Department of Computer Science, University of Electro-Communications).

2 設計

2.1 システム構成

システム構成の概略を図1に示す。サーバとクライアント間の通信にはTCPによる1対1通信を用い、必ずクライアント側からの要求を受けてからサーバはそれの応答を返す形でランデブ通信を行う。ブロードキャストは一切行わない。

ユーザに提供するプリミティブの代表的なものと、それを処理する際のクライアントの要求、サーバからの応答を図2に示す。

```
sh_mem_alloc(alloccsize)
    要求 (a) alloccsize で示す大きさの共有メモリブロックの作成。
    応答 (a') 作成した共有メモリブロックの識別子。

sh_mem_get(mem_blk, bufaddr, bufsize, offset)
    要求 (b) 共有メモリブロックからのデータの読み出し。
    応答 (b') 共有メモリブロックのデータ。

sh_mem_get は、mem_blk で示される共有メモリブロックの
offset バイト目から bufsize バイトのデータを、bufaddr で示
されるアプリケーション内のバッファにコピーする。以下同様。

sh_mem_put(mem_blk, bufaddr, bufsize, offset)
    要求 (c) 共有メモリブロックへのデータの書き込み。
    応答 (c') アクノレッジのみ。

f_a(mem_blk, inc, offset)
    要求 (d) 共有メモリブロック内のデータを fetch_and_add。
    応答 (d') fetch_and_add を行った結果の値。
```

図2: プリミティブと、それを処理する際の要求と応答

2.2 共有メモリブロックのキャッシュ

クライアントとサーバ間の通信回数を削減するため、各クライアントに自分の読み書きした共有メモリブロックのキャッシュを持たせる。sh_mem_get が呼ばれたときにキャッシュがヒットすればその内容を返す(図2の要求(b)を送らない)。

キャッシュを行うとき、キャッシュのコヒーレンシを保つための処理が必要となる。コヒーレンシは `sh_mem_alloc` で作成された1つの共有メモリブロックを単位として保たれる。

サーバは、要求(c)、要求(d)を受けた時に、要求と同じ共有メモリブロックのキャッシュを持っている別のクライアントに対してキャッシュの無効化を通知してやる必要がある。この無効化の通知はただちに通信されるのではなく、次にそのクライアントから何らかの要求の通信を受けたときに応答(a'~d')と併せて送られる。

2.3 キャッシュの改良

さらに、通信回数を削減するために、以下の改良を試みた。

- 図2の要求(a)、要求(d)については、クライアントは必ずすぐにサーバと通信して送る。
- `sh_mem_put` が呼ばれた場合は、キャッシュにのみ書き込む。すなわち、要求(c)はクライアント側で貯められる。これは次に行われる何らかの通信の時に併せて送られる。
- `sh_mem_get` が呼ばれたときに、もしキャッシュがヒットしている場合でもその共有メモリブロックが上記によりキャッシュに書かれ、しかも要求(c)が貯められたままになっている場合には、クライアントはすぐに通信を行って共有メモリブロックのデータを更新する。

応用プログラムにおいて正しく同期がとられていれば、本方式でコヒーレンシが保てているものと思われる。

3 実験

前章の設計によるシステムを試作し、実験を行った。とくに、2.3節で述べた方式によるキャッシュの効果を確かめるために、表1に示すの3種類のシステムを作成して比較した。B構成とC構成との大きな相違点は、図2の要求(c)をクライアント側で貯めないか貯めておくかである。

表1: 試作した3種類のシステム

A 構成	キャッシュを行わない、すなわちプリミティブが呼ばれるたびにサーバとの通信を行うもの
B 構成	2.2節で述べた方式でキャッシュを行うもの
C 構成	2.3節で述べた方式でキャッシュを行うもの

また、これらのシステム上でアクティビティ方式[2]のスケジューラを動作させた。アクティビティ方式は多数の並列実行処理単位(タスク)を効率よく処理するために筆者らが開発してきたものであるが、共有メモリ型並列機を対象としたものなので、スケジューラ自体もかなりの頻度で共有メモリにアクセスをする。

応用プログラムとしては、10都市巡回セールスマン問題を解く並列プログラムを動作させた。この問題において枝刈りをする場合でも約40,000個のタスクを生成する。その生成、切替に前述のスケジューラの機能を用いている。

表2: 実行結果(クライアント数は3台)

	通信回数	通信データ量 (Mbyte)	処理時間 (sec)
A 構成	640040	68.2	336
B 構成	343116	37.6	199
C 構成	210434	25.6	133

3台のクライアントにより並列実行させた場合の実験結果を表2に示す。キャッシュを全く行わないA構成のものに比べてキャッシュを行うB構成C構成では大きく性能向上が図れることが確かめられた。

図2の要求(c)をクライアント側で貯めておくC構成は、要求(c)を貯めないB構成に比べてさらに性能が改善された。C構成では通信回数と通信データ量が削減できており、このことが性能に与える効果が大きいことがわかる。

4 まとめ

ネットワークにより結合された複数のUNIXワークステーションにおいて分散共有メモリをサポートするための機構を試作した。通信回数を削減できるようなキャッシュの方式を設計することにより性能の向上を図れることが確かめられた。

参考文献

- [1] Nitzberg, B. and V. Lo: Distributed Shared Memory: A Survey of Issues and Algorithms, *Computer*, Vol.24, No.8, pp.52-60 (1991).
- [2] 中山泰一, 永松礼夫, 出口光一郎, 森下巖: 共有メモリ型並列機のための新しいアクティビティ方式並列実行機構, 情報処理学会論文誌, Vol.34, No.5, pp.985-993 (1993).