Lucas オペレーティングシステムにおけるメモリマップ技術を用いたストリーム型プロセス間通信

5B－3

宮澤 元　猪原 茂和　上原 敬太郎　益田 隆司
東京大学大学院 理学系研究科 情報科学専攻

1　はじめに

我々が開発中の64bit オペレーティングシステム Lucas では分散協調アプリケーションの開発のために、メモリマップ技術を用いて分散ファイルシステムと分散共有メモリを融合したリージョンという仮想記憶管理機構を採用している。このリージョン機構をプロセス間通信に応用することで、従来の UNIX 上のプロセス間通信に必要だったユーザプロセスとカーネルの間の送信データのコピーを減らすことができるために、通信の高速化が期待できることに加え、構造をもったデータを扱うことができる。しかし、リージョンの一貫性保持のための通常の分散共有メモリプロトコルでは、主にメッセージ数を減らすことによる性能向上をはかっているので、UNIX で頻繁に用いられるストリーム型のプロセス間通信を実装した場合、読み手の latency が大きくなってしまう。本研究ではメモリマップファイルを用いた効率的なストリーム機構を提案する。特に送信データに対してメモリのアクセスパターンに応じた prefetch を行なうことで読み手の latency を下げることを目指したプロトコルについて述べる。

2　ストリームの概要

メモリマップファイル上のストリーム通信でも、従来の UNIX のストリーム通信と同様に一つのストリームに対して書き手となるプロセスと読み手となるプロセスが一つずつ存在する。書き手が書き込んだデータを送信する時は、通信用リージョン上の区間を指定して send() を明示的に発行することにより、システムに送信データを示す。同様に、読み手はデータ読み出しの前に recv() を発行する。

この理由は、メモリマップファイルであるリージョンを通信に用いて構造を持ったデータを扱うと、書き手と読み手が通信用に割り当てられたリージョンに通常のメモリアクセスとしてデータの書き込み、読み出しを行なうため、システムはメモリ上のどのデータを送受信するべきかを自動的に判断することは困難になるからである。転送データの粒度を大きくしてページ単位で転送する方式も考えられるが、自由度が低くなってしまうという問題がある。本研究のストリーム機構では send(),recv() を使用することで転送データを明示することができるとともに、ページ境界に縛られない通信が可能である。

例えば、このストリームを用いてツリー構造を転送する場合、ツリーの各ノードに対して send(),recv() を発行して通信を行なうことになる。このようなデザインにすることで、本研究のストリーム機構は UNIX のストリーム機構の拡張となっており、時間的に増大していくデータ構造の通信を行なうことができる。

3　prefetch を行なうストリームプロトコル

通常の分散共有メモリプロトコルでは、通信に要するメッセージ数を減らすために、読み手のデータ要求に従ってデータを転送する。このため、このプロトコルを用いてストリームを実装した場合、読み手の latency が大きくなる (図1(a) request driven)。本研究では分散環境上で、読み手の latency を減らす prefetch プロトコルを提案する。送信データが構造的かそうでないかに応じて、ユーザは線形 prefetch と構造的 prefetch の2種類のプロトコルを選んで用いることができる。

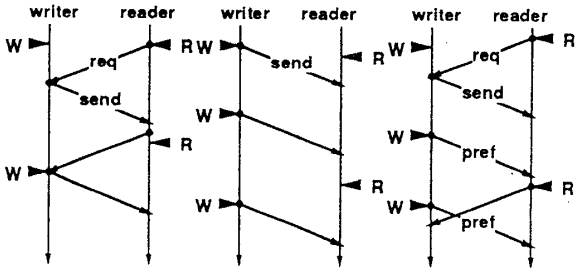


図 1: Protocol

3.1　線形 prefetch のプロトコル

構造のないストリームは、send() が発行された区間をすぐに読み手に転送する eager なプロトコルを用いて通信を行なう (図1(b) non-typed)。読み手は書き手からデータが転送されるのを待つだけであり、データの要求は行なわない。これは、従来の UNIX で使われている TCP/IP ストリームのプロトコルをメモリマップファイル上で実現したものである。書き手がデータを生成した順で読み手もアクセスするため、書き手が生成したデータを即座に転送することで失敗のない prefetch を行なうことができる。

3.2　構造的 prefetch のプロトコル

ポインタを含む複雑なデータ構造をストリームで送るような場合、書き手がデータ構造を生成していく順序と読み手がポインタを参照してデータ構造を読んでいく順序が違ってしまう場合があると考えられる。このような場合、3.1節の protocol では性能的に不利になると予想される。そこで、アプリケーションがシステムに対して送信データに含まれるポインタや配列の構造情報をヒントとして与え、データ構造に従って prefetch を行なう。

読み手では recv() 発行時にアクセス通知を書き手に送る。書き手は、読み手からの通知を受け取ると、まずアクセス通知の生じたデータを調べ、そのデータがまだ転送されていなければ転送する。次に、アクセス通知の起こったデータと構造的に関連したデータを prefetch し、読み手からのアクセス通知がなくても優先的に読み手に送る (図1(c) typed)。構造を持ったデータはあらかじめ与えた構造情報に従ってアクセスされるので、データ構造に応じた prefetch により性能向上が期待できる。

Stream communication using memory-mapping in the Lucas operating system
MIYAZAWA Hajime　INOHARA Shigekazu　UEHARA Keitarou　MASUDA Takashi
The Department of Information Science, Graduate School of Science, the University of Tokyo

4 ストリームの性能測定

以上で説明した二つの方式の有効性を調べるためにストリームのシミュレータをSunOS上で製作し、性能を測定した。転送データとしてストリームを要素とするストリームを仮定し、書き手は全部に書き込みを行ない、読み手は全ストリームから選択的にいくつかのストリームのみをアクセスすることとした。(1) request driven, (2) non-typed prefetch, (3) typed prefetch の三つのプロトコルを比較した結果を図2に示す。

グラフの横軸は書き手が書いたデータ要素のうち読み手が必要とするデータ要素の割合を示す選択率である。この選択率が高い場合は、non-typed が総じて良い結果を出している。これは、選択率が高い場合、シミュレーションで用いたストリームを要素とするストリームという構造は普通のストリームに近似できることに起因している。

選択率が低い場合には書き手と読み手の相対的な速度によって3通りの結果がでた。読み手が相対的に速い図2(a)の場合、request drivenが有利である。これは、書き手のデータ書き込みが遅いために、読み手からのリクエストが、書き手がデータを書き込む前に届いてしまい、実質的に各プロトコル間の差がなくなってしまうためと思われる。双方が同程度の速さの図2(b)では、typed prefetchが有利である。この場合にはtyped prefetchのコストが、non-typed prefetchで余分なデータを送ってしまうオーバーヘッドより低くなっているためと思われる。書き手が相対的に速い図2(c)の場合では、型のあるなしにかかわらずprefetchを用いたプロトコルが有利であった。これはnon-typed prefetchで余分なデータを送るオーバーヘッドやtyped prefetchでのprefetchミスが読み手と書き手の速度差に吸収されてしまうからであろう。

5 まとめ

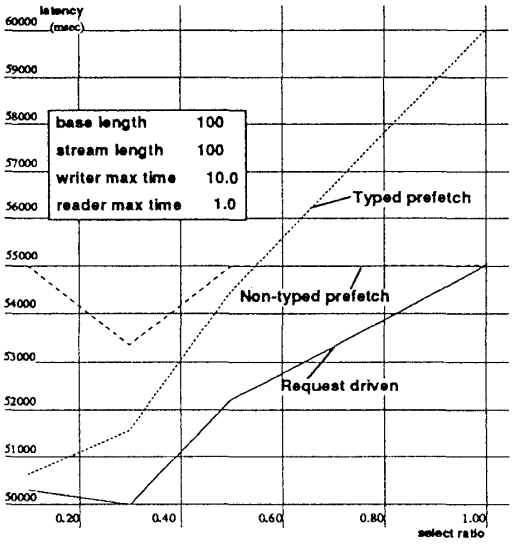
64bit オペレーティングシステムLucas上で複雑なデータ構造をプロセス間で通信するためのメモリマップファイルを用いたストリーム通信機構を提案し、そのシミュレータを製作して性能を測定した。このストリームは従来のUNIXにおけるストリームの概念を構造を持ったデータに対し拡張したものであり、効率的なプロセス間通信を可能とする。現在DECstation上のMachを用いて開発を行なっているLucasの上にこの機構を実装中である。

また現状では、構造をもったストリームのprefetchに用いられる構造情報を与える作業はアプリケーション毎に行なわねばならないが、将来的にはコンパイラが適切なサポートをすることで自動化できるだろう。

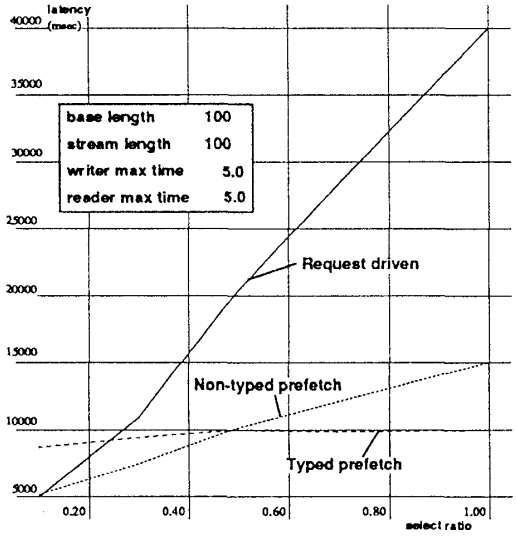
参考文献

[1] Brian N. Bershad, Thomas E. Anderson, Edward D. Lazowska, and Henry M.Levy. User-level interprocess communication for shared memory multiprocessors. *Transactions on Computer Systems*, 9(2):175–198, May 1991.

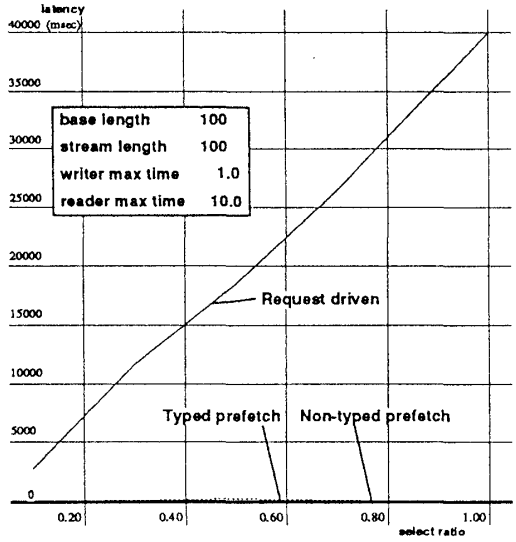
[2] Orran Krieger, Michael Stumm, and Ron Unrau. Exploiting the advantage of mapped files for stream I/O. In *Proceedings of the USENIX 1992 Winter Conference*, pages 27–42. USENIX Association, January 1992.



(a) 読み手が速い場合



(b) 双方の速度が同程度の場合



(c) 書き手が速い場合

図 2: 実験結果