5B−2

Lucasオペレーティングシステムにおける分散ファイルシステムと分散共有メモリの融合

上原 敬太郎　猪原 茂和　宮澤 元　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

近年注目を集めているグループCADやソフトウェア開発などの協調作業を行なうアプリケーションでは、ポインタを含む複雑なデータ構造を複数のプロセスが共有して使用する。そのようなアプリケーションを効率良く実装するための枠組として、当研究室では64ビット仮想空間を用いたオペレーティングシステムLucasの研究開発を行なっている[2]。本稿ではLucas上のファイルシステム(以下LFS)について述べる。

従来のUNIX等で使われているファイルではread()やwrite()を用いてファイルの内容をメモリにコピーして使用するため、内容の変更を複数のプロセス間で即時に伝播させることが困難である。このためLFSではより協調作業に適したmemory-mapped fileのインタフェースを用いている。一方分散環境上の変更伝播については各ホスト上にキャッシュされたmapped fileの一貫性を保つ必要がある。LFSではこのキャッシュ間の一貫性を保証するためのプロトコルとして、従来の分散共有メモリの技術に分散ファイルシステムとしての特徴を活かした最適化を導入し、実行時効率の改善を目指した。以下、LFSの設計とプロトコルの最適化について述べる。

2 Lucasファイルシステムの概要

上で述べたように、分散した環境で変更を伝播するためには各ホスト上にキャッシュされたmapped file間の一貫性を保つ必要がある。LFSでは各サイト上のディスクを管理しているストレージサーバ(SS)の他に各サイト上でキャッシュを管理するキャッシュサーバ(CS)を置いている。各サイト上のクライアントプロセスはファイルを仮想空間にマップすることでそのサイト上のCSと通信をする。CSはキャッシュがサイト上にあればそれをクライアントのアドレス空間にマップし、なければそのファイルが存在するサイト上のSSと通信し、キャッシュを要求する。(図1)

従来の分散ファイルシステムでは、同時に複数のクライアントがファイルを共有するケース(concurrent write sharing)は稀であるとして、効率と一貫性のどちらかを犠牲にして処理を行なっている。例えばSprite[3]では書き手が複数になるとキャッシュの使用をやめて遠隔サービスに切り替える。Andrew[1]ではそもそも厳密な一貫性を保証しておらず、もっと緩い一貫性(セッションセマンティクス)に基づいている。

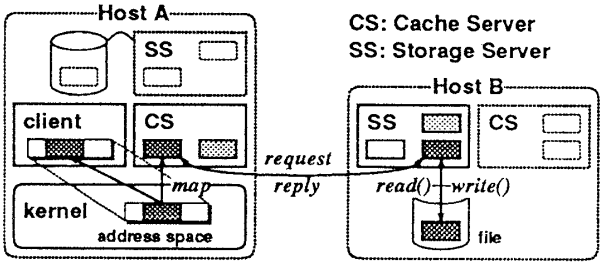

図1: LFSの構成

一方LFSでは協調作業を念頭に置いている。そこで厳密な一貫性を保証する分散共有メモリのセマンティクスを採用している。これは、Multiple readers or one writerと呼ばれるセマンティクスで、常に複数の読み手(reader)か唯一の書き手(writer)のみを認めるというものである。複数の読み手にキャッシュが共有されている状態をshared、書き手にキャッシュが占有されている状態をdirtyという。sharedからdirtyへ、またはその逆へと状態が遷移する際にはキャッシュを無効化(invalidation)するメッセージが送られる。

3 プロトタイプ版の性能測定

LFSのプロトタイプをDECstation5000(CPU: MIPS R3000)上のMachの上に実装した。CSとクライアントとの間のメモリマップにはMachの外部ページャーインタフェースを使用している。マップ、およびページフォールトのオーバーヘッドを同一ホスト上のファイル(local)／遠隔ホスト上のファイル(remote)のそれぞれについてキャッシュのあるなしでどの程度差が出るか計測してみた。その結果を図2に示す。これによれば、ページフォールトは1ページ当たりローカルで約5.85ms、リモートでは約46.0msのオーバーヘッ

Integration of a Distributed File System and a Distributed Shared Memory System in Lucas
UEHARA Keitaro, INOHARA Shigekazu, MIYAZAWA Hajime, and MASUDA Takashi
The Department of Information Science, Graduate School of Science, the University of Tokyo

	local	remote
map(CS cache なし)	16.72	41.78
map(CS cache あり)	2.12	2.12
page fault(CS cache なし)	5.85	46.02
page fault(CS cache あり)	1.95	1.95
page access	0.36	—

図 2: 1 ページ当たりのアクセス時間 (単位:ms)

ドがかかる (オーバーヘッドの内訳はカーネルのページフォールトの検出、CS に対する要求、SS に対する要求、実際のファイルの読み込み、CS に対するページ供給、カーネルに対するページ供給、である)。なお、このページフォールトのコストはシステム組み込みのページャーによるページフォールトのコスト (約 3.9ms) に比べてもあまり大きくはない。リモートの際には単純なメッセージの送信だけでも約 10ms のコストがかかるため、数十 ms のオーダーになるのは避けられない。

4 オープンモードによる最適化

上の測定結果より、特に遠隔ホストとの通信を伴う場合にはメッセージを送る部分がかなり大きなオーバーヘッドを占めていることがわかる。従って、送るメッセージの数を減らすことが全体としての性能向上に大きく貢献することになる。そこで分散共有メモリのプロトコルにファイルシステムのインタフェースを利用した最適化をすることを考える。単純な分散共有メモリと違い、mapped-file には map と unmap(ファイルでいう open, close に当たる) による明確なセッションの区切りがある。また、ファイルのオープンの際には必ずアクセスモードを指定する。これらを利用することで、将来的に起こるであろうアクセスの先読み (prefetch) による効率化が図れると考え、検討した。具体的には write モードでオープンされているクライアントからの read によるページの要求は、その直後に write が来ることに備えて、write コピーを供給することにする。これによって一旦 read コピーを供給し、直後にまた write コピーが要求されるという冗長な処理を省くことが出来る (図 3参照)。

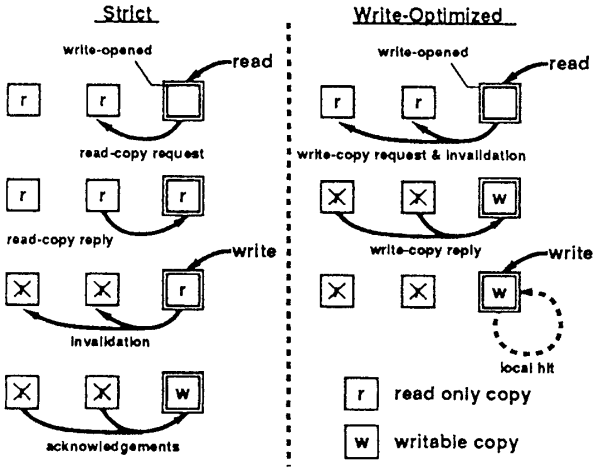

図 3: 最適化された write プロトコル

この最適化を行なったシミュレーションの結果を図 4に示す。このグラフは全体のうち読み手が占める割合とそれに対応するメッセージの数を示している。この最適化によって平均 20.5% 最高 47.6% のメッセージ量の減少が見られる。特に書き手の占める割合が多い時に効果が大きいことがこのグラフから見てとれる。

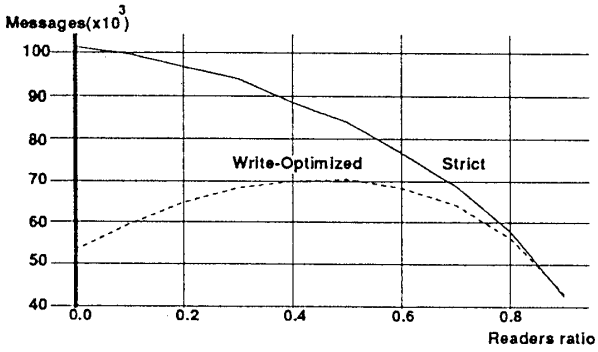

図 4: write 最適化によるシミュレーション結果

5 まとめ

本稿では協調作業に適したファイルシステムとして内容をメモリに直接マップする mapped-file を使用した分散ファイルシステムを提案し、プロトタイプ版を用いて性能測定を行なった。

現在のところ性能はほぼ予測した通りの数値が出ているが、まだ満足できる値というわけではない。特に、常に厳密な一貫性を保証することは、キャッシュ無効化のメッセージを数多く発生させることになり、効率が良いとは言えない。将来的には段階的に緩くなる複数の一貫性プロトコルを用意し、ファイル単位あるいはセッション単位でユーザーが指定することができるようにすることも考えている。

参考文献

[1] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, and R. N. Sidebotham. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems*, Vol. 6, No. 1, pp. 51–81, Feb 1988.

[2] 猪原茂和, 上原敬太郎, 宮澤元, 益田隆司. オペレーティングシステム Lucas における 64 ビットアドレス空間の管理. In *6th SWoPP*, Aug 1993.

[3] M. N. Nelson, B. B. Welch, and J. K. Ousterhout. Caching in the Sprite Network File System. *ACM Transactions on Computer Systems*, Vol. 6, No. 1, pp. 134–154, Feb 1988.