オペレーティング・システム 58－1
（1993. 3. 19）

地球規模分散ファイルシステム向きのファイルキャッシュプロトコルの設計

弘田 暢幸 † 藤田 聡 † 山下 雅史 † 亀田 恒彦 ‡
†広島大学工学部 ‡Simon Fraser 大学

WAN 上でファイルを共有するためのファイルキャッシュプロトコルを提案し，シミュレーションにより評価する．提案するプロトコルでは，従来のサーバ-クライアントモデルとは異なり，サーバは各 LAN に一つずつ配置され，ファイルの本体を保持せず，その位置を管理する．ファイルの最新版は特別なクライアントによって保持され，その位置はアクセスの状況に応じて変化する．また，複数の LAN に同じファイルの最新版が存在することを許すことによって，WAN 上でのファイル転送とメッセージを減らす．シミュレーションにより，WAN 上での可用性を従来の分散ファイルシステムと比較し，その有効性を示す．

A File Cache Protocol for World Wide Distributed File Systems

Nobuyuki Hirota † Satoshi Fujita † Masafumi Yamashita † Tiko Kameda ‡

†Faculty of Engineering, Hiroshima University
‡School of Computing Science, Simon Fraser University, Canada

A file cache protocol to share files on a WAN, which is geographically distributed in wide area, is proposed and evaluated by simulations. The proposed protocol does not adopt the conventional server-client model. A server exists in each LAN and maintains only the locations in the WAN of the latest versions of files. Multiple "latest" versions of a file are allowed to exist in the WAN to reduce inter-LAN file transmissions. These latest versions are held by clients and the clients holding latest versions may change depending on who accesses to the file. Their effectiveness is demonstrated by comparing with some conventional distributed file systems.

1 はじめに

局所ネットワーク (LAN) や広域ネットワーク (WAN) などの計算機ネットワーク上では, 異なる計算機に分散して存在する様々な資源が共有されており, 各計算機はそれらの資源に対してネットワークを介してアクセスすることができる. それらの資源に対するアクセスは, どの資源に対しても同じ方法で, しかもそれらが物理的にどこにあるのかを意識せずに行なえることが望まれる. 資源に対して, ネットワークの存在を意識せずにアクセスできる時, ネットワークは透過性 (transparency) を持つという. ユーザに対してネットワークの透過性を提供することを目的の一つとして, 多くの分散オペレーティングシステムが開発されている. 特に, 計算機ネットワークで共有されているファイルへのアクセスに関してネットワークの透過性を提供することを主目的として分散ファイルシステム (DFS) が提案されている [2, 8, 9, 10].

他の計算機に保持されている遠隔ファイルに対して効率的にアクセスするためには, まずそのファイルを局所記憶 (あるいはディスク) に転送しなければならないが, この転送には避けられない遅延が伴う. 同じ遠隔ファイルが頻繁にアクセスされる場合, アクセスの度にネットワークを介してそのファイルを転送するならば, ネットワークに負荷をかけ, 時間もかかって効率が悪い. そこで, ネットワークの負荷を低減し, ファイル転送にかかる時間を節約することを目的として (ファイル) キャッシュ機構が導入されている.

ファイルキャッシュ機構は, あるサイトで一度アクセスされたファイルは, 近い将来再び同じサイトでアクセスされる確率が高いというアクセスの局所性 [3, 7] を前提に考案された手法である. これは, ある遠隔ファイルにアクセスした際に, その複製をキャッシュと呼ばれる局所記憶媒体に保持しておき, 次に同じファイルをアクセスする場合には, キャッシュ内の複製に対してアクセスを行うというものである. 一般に, キャッシュには主記憶か局所ディスクが用いられ, それらに対するアクセス速度は, ファイルの転送速度よりもかなり早い. そこで, キャッシュを用いることでファイルのアクセスにかかる時間とネットワークの負荷が低減され, アクセスの効率が向上する.

キャッシュ機構は一般にサーバ – クライアントモデルで表現される. クライアントは遠隔ファイルに対してアクセスする際にサーバにファイルの転送を要求し, サーバはクライアントに対してそのファイルの複製を供給する [2, 8, 9, 10]. キャッシュ機構を導入すると, 複数のクライアントが同じファイルの複製を持つ場合がある. この場合, あるクライアントがファイルに変更を加えた時に, 同じファイルをキャッシュしているクライアントがその変更に気付かずにキャッシュに対してアクセスしてしまうと, 本来は 1 つのファイルであるにも関わらず, 異なる内容を持つファイルが存在してしまうという問題が生じる. この問題はキャッシュの一貫性問題 (cache consistency problem) と呼ばれている. この問題を解決するために, サーバ – クライアント間でのファイルの転送, キャッシュへのアクセスを制御するプロトコル, つまり, (ファイル) キャッシュプロトコルが必要になる.

最近, 可伸性 (scalability) が注目され, それに重点をおいた DFS も開発されている. [1, 11] 可伸性とは, サービスの負荷の増大にシステムが適応していける能力である. ここでのサービスの負荷は, クライアントの数に起因するものと考えられている. つまり, クライアント数の増加によって, サーバに対する要求の量が増し, サーバがそれらの要求を効率良く処理できなくなるということである. これは, DFS の性能を制限するボトルネックがサーバ, 特にサーバのディスクアクセスの能力の限界にあると考えていることを意味する. このため, 可伸性を持たせるために, クライアントの数が増加してもサーバに対する負荷ができるだけ増えないようにするため, サーバ – クライアント間での通信量を減らし, クライアントの動作をサーバから隠す工夫がなされている.

これとは別に, DFS の地理的広がりもその可用性を制限する要因になると考えられる. この場合, DFS の可用性を制限するボトルネックは, サーバの処理能力の他に, 長距離通信のためのネットワークの通信遅延にもあると考えられる. このような, 地理的広がりに適応して効率的なサービスを提供できる能力を地理的可伸性と呼ぶことにする. この地理的可伸性を実現するためには, 特に LAN 間の通信を極力減らしたキャッシュプロトコルが必要であり, そのためには従来の DFS とは異なったキャッシングの戦略が必要であると考えられる.

本報告では, 地球規模に及ぶ広域ネットワークに対して地理的可伸性を持ち, 透過的なアクセスを提供するためのキャッシュプロトコルを提案し, シミュレーションによりそれらと従来のプロトコルとを比較し, その有効性を示す. 拠所とする原理は, やはりアクセスの局所性である. 我々は, アクセスには時間的な局所性 (一度アクセスしたファイルは近い将来再びアクセスする確率が高い) と共に, 地理的な局所性 (一度アクセスしたファイルは近い場所で再びアクセスされる確率が高い) も持つことを前提として, キャッシュプロトコルを提案する.

2 ネットワーク

我々が対象とする分散システムは, ネットワークに接続された計算機群上に実現される. ネットワークは, 各地域で LAN を構成し, 更にそれらはインターネット技術により接続され, WAN を形成している.

各計算機上では, 様々なプロセスが実行されているが, どのプロセスも故障はしないものとする. また, 各プロセスはそれぞれ実時間の時計を持っており, それらの指す時刻は常に等しいと仮定する.

各プロセスはトランスポート層のプロトコル [4] を使用することを仮定する. 従って, 全てのプロセスは他の任意のプロセスと論理的に直接通信することができる. しかし, プロセス間通信は同じ LAN 内で行なわれる場合と, 異なる LAN 間, つまり WAN 上で行なわれる場合があり, この違いは通信遅延に大きく現れる.

一般に, LAN 内の計算機は近距離に配置され, 同じイーサネットに接続される. このため, プロセス間通信は直接行なわれ, 通信速度は早く信頼性も高い. 一方, WAN はゲートウェイを介して接続された複数の LAN から構成され, 各 LAN は地理的に広範囲に渡って分散している. 通常各 LAN 間には複数のゲートウェイが存在し, LAN 間の通信は常にそれらのゲートウェイを介して行なわれる. このため, LAN 間の通信遅延は LAN 内のそれに較べて非常に大きく, ゲートウェイの負荷の影響も受けるために不安定である. 文献 [4] では, LAN 内のパケットの転送速度は, $4Mbps$ 〜 $2Gbps$, WAN 上では, $9.6Kbps$ 〜 $45Mbps$ であると報告されている. しかし, 我々が実験の結果得た LAN, WAN でのプロセス間通信におけるパケットの転送速度は以下の通りである.

LAN 内の通信は, 広島大学工学部情報工学専攻の LAN を用い, WAN 上の通信は, 同 LAN と Simon Fraser 大学 (バンクーバ, カナダ) の LAN 間で行なった. 信頼性を保証する通信プロトコルである TCP [4] を用いた場合, 転送速度は LAN 内で約 $1Mbps$, WAN 上で約 $9Kbps$ であった. そこで, 我々のモデルでは, WAN 上の通信遅延は, LAN 内の約 100 倍であると想定する.

3 分散ファイルシステム

3.1 共有の意味

キャッシュの一貫性を議論するには, あるファイルに対して複数のプロセスが同時にアクセスした時の正しい結果を規定する必要があり, 従来, 主に UNIX セマンティクスとセッションセマンティスクが正しさの基準として採用されてきた.

UNIX セマンティクスは Sprite などで採用されており, 以下のように定義される [8, 10]. DFS のファイルの全ての読み込み操作において, それまでになされた全ての書き込みが反映される. 特に, あるクライアントがオープンしているファイルへの書き込みは, 同じ時に同じファイルをオープンしている他の全てのクライアントにも直ちに読めるようになる. このセマンティクスは, DFS の全てのファイルに対して, 全てのクライアントが常に同じ内容を参照できることを保証しているが, このセマンティクスをキャッシュ機構を用いて実現するためには, 多くのメッセージを必要とし, ネットワークに負荷をかける.

セッションセマンティクスは AFS で採用されており, 次のように定義される [8, 9]. あるファイルに対する一連のアクセスはファイルのオープンからクローズまでを一単位とし, これをセッションと呼ぶ. セッション中は自由にファイルに対して読み書きができ, セッション中に起こった変更はそのセッション内では直ちに反映されるが, そのセッションが終了するまでは他のクライアントには一切反映されない. ファイルセッションが終了すると, そのセッションで起こった変更はある遅延を伴って他のクライアントに反映されるが, すでに開始されているセッションに対しては反映されない. (遅延のためにその変更の伝わり方にも差が出るかも知れない.) このセマンティクスはファイルセッション中はアクセスしているファイルの内容が他のクライアントのアクセスにより干渉されないことを保証するが, あるファイルが変更を伴ってクローズされた場合, そのファイルをすでにオープンしている他のクライアントは古い版を使っていることになってしまう. しかしながら, このセマンティクスは UNIX セマンティクスよりはるかに少ないネットワークの負荷で実現可能である.

3.2 従来のキャッシュプロトコルの WAN 上での問題点

現在広く知られている DFS としては, NFS, AFS, Sprite などが挙げられる [2, 8, 9, 10]. [1] これらはいずれもサーバ－クライアントモデルの DFS で, LAN を対象として設計されている. 特に, AFS と Sprite はキャッシュの一貫性を重視して設計されている. これらの DFS は, それぞれのファイルに対してそれを

[1] 以下では, AFS あるいは Sprite と言ったとき, AFS あるいは Sprite で採用されているキャッシュプロトコルを意味すると解していただきたい.

管理するサーバが一意に定まる．これらを本研究で対象とする WAN 上で動作させることを考える．この時，WAN 上ではある特定のファイルに対して，WAN の中のどれか 1 つの LAN 内にのみ特定のファイルのサーバが存在することになる．

このような条件の下で，AFS, SPRITE について一定時間に各クライアントが繰り返しファイルを要求する場合のファイルに対するアクセス回数をシミュレーションにより調べた．その結果，サーバから地理的に離れたクライアントは WAN の大きな通信遅延のため，サーバと同じ LAN 内のクライアントの 1 % 未満しかアクセスができないことが分かった．(ただし，この実験ではファイルアクセスの局所性を 0.5，すなわち 2 回に 1 回は直前のファイルを再び要求するものと仮定した.) 従来の DFS を WAN 上で使用した場合の問題点として以下のものが挙げられる．

- クライアントに対して公平なサービスが提供できない．
- WAN 上にかなりの数のメッセージが流れる．
- サーバから離れたクライアントがファイル要求をする場合，有効なキャッシュを持つクライアントが同じ LAN 内に存在してもそれを利用できず，サーバからファイルを転送することによって高価な通信を用いなければならない．

我々は，WAN 上の通信を極力減らし，効率良く，しかも公平なサービスの提供できる新しいキャッシュプロトコルを提案する．

4 WAN のためのファイルキャッシュプロトコル

地理的に離れた比較的小数の LAN から構成される WAN 上の DFS を対象として，セッションセマンティクスを保証するファイルキャッシュプロトコルを提案する．キャッシュデータの粒度は，キャッシュプロトコルの効率を左右する重要な項目の一つであるが，(取りあえず簡明のため) 本プロトコルでは，ファイル全体をキャッシュする．そこで，キャッシュには局所ディスクが用いられるが，その容量は十分にあるものとする．

セッションセマンティクスでは，あるファイルに対する同時アクセス，特に変更を伴うアクセスが頻発した場合，UNIX セマンティクスの意味でのファイルの内容の一貫性は保証できない．このため，データベースなどのように厳密な一貫性制御を必要とするアプリケーションは，そのセマンティクスに頼らずにそのアプリケーション内で一貫性制御を行なわなければならない．しかしながら,Kent [7] によると，日常の UNIX ファイルシステムの使用状況ではファイルの同時アクセスの起こる割合は 4 % 程度であり，書き込みを伴う同時アクセスは，更に同時アクセス全体の約 4 % であることが報告されている．我々は，一部のアプリケーションのためにより厳密な一貫性を保証するよりも，DFS 全体の効率を優先する．

4.1 キャッシュプロトコルの概要

従来のキャッシュプロトコルでは，各ファイルに対してある固定されたサーバが対応しており，クライアントによってなされた変更は，タイミングは異なるが，必ずそのサーバに書き戻される．その結果，最新版はサーバに存在する．そこで，あるファイルを要請するには，そのファイルを管理するサーバに要請すれば，ことは足りる．

我々が提案するキャッシュプロトコルもサーバとクライアントから構成されるが，従来のキャッシュプロトコルと違ってサーバはファイルの本体を管理しない．サーバは各 LAN に一つずつ置かれ，すべてのファイルに関して，ファイルの最終変更時刻，サイズ等の情報と，“最新版” の位置を管理している．(提案するプロトコルでは，複数の “最新版” が存在する可能性がある.) ファイルの最新版はファイナルライター (FW) と呼ばれる特別なクライアントが保持しており，FW はファイルのアクセスの状況に応じて移動する．各ファイルの FW は，DFS 全体で少なくとも 1 つ存在し，また，各 LAN 上に高々 1 つしか存在しない．サーバの仕事の一つは，すべてのファイルに関して FW の一つにたどり着くための情報を記憶しておくことである．これは，おおよそ以下のようにして行われる．

対象となる WAN において，各 LAN を節点,LAN 間の物理的接続関係を枝と考えることで，グラフが定まる．LAN 間の接続にかかる通信遅延を対応する枝のコストと考えたとき，グラフのコスト最小生成木を Tとする．すべてのサーバ間の通信は Tの枝を介して行われる．サーバは，Tを認識している．そして，各ファイルについて，FW がその LAN 内に存在すればそのことを，もしか存在しなければ，自分の属する LAN に接続している Tの枝で，FW が存在する LAN に向かう Tの枝 (すなわち，FW の存在する方向) を記憶する．目的の FW には，記憶されている枝を順々に辿ることによって到着できる (図 1を参照せよ).

セッションセマンティクスでは，クライアントによって変更されたキャッシュデータの内容は，そのファイルがクローズされた時点から有効になる．そこで，我々

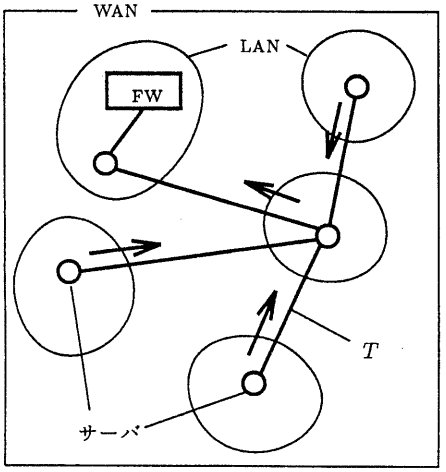

図 1: WAN, 生成木 T, FW, および FW の位置の記憶法

のプロトコルでは, クローズが発生した時点で, その事実を自分が属する LAN のサーバに伝え, サーバが知る限りそれが最新であれば, そのクライアントにそれが最新版であることを告げる. その結果, そのクライアントは FW となり, そのキャッシュデータは, サーバに書き戻されることなしにファイルの最新版であるとみなされる. この後, サーバは他のサーバにそのファイルの FW の位置が変更されたことを告げる.[2]

クライアントによるファイルの要請は, 同じ LAN に属するサーバに対してなされる. サーバは, その LAN に FW が存在するならば, それに対して, そうでないときには, 他の LAN に属する FW に対して, ファイルをそのクライアントに送信することを要請する. その FW は, ファイルを直接そのクライアントに送信し, それが自分の複製を持つことを記憶し, (後述の) 無効化命令に備える. (従って, ファイルはサーバを経由されることなく, クライアント間を直接転送される.)

キャッシュプロトコルにおけるクライアントの重要な仕事は, 自分のキャッシュデータが最新版であるのかどうかを判断することである. キャッシュが最新であるとき, そのキャッシュは有効であるという. キャッシュの有効性の確認をする方法として, クライアント主導型と, サーバ主導型が提案されてきた [8]. クライアント主導型は, クライアントがファイルアクセスの必要に応じてサーバに対してキャッシュの有効性の確認を行なう. この方法では, キャッシュが有効な場合にも, アクセスの度に有効性の確認を行なう必要があるので, ネットワークとサーバに負荷をかける. サーバ主導型は, ファイルの変更が起こった時に, サーバが古い版を持つクライアントのキャッシュを破棄させることにより, 古い版へのアクセスを未然に防ぐ. この動作をキャッシュの無効化という. キャッシュが無効化されるまで, そのキャッシュは有効であるとみなされる.

我々は, サーバ主導型と似た FW 主導型を採用する. ここでは, クライアントにキャッシュの破棄を命令するのは FW である. すなわち, サーバがあるファイルのより新しい版の存在に気付いたとき, (もしそのファイルの FW が同じ LAN 内に存在すれば) そのことを FW に告げ, その FW が FW と同じ版を持つ同じ LAN 内のクライアントに対し, キャッシュの無効化命令を送信する. どのクライアントが FW と同じ版を持つか, サーバは関知しないので, 無効化命令の送信は FW の仕事である.

本プロトコルは, LAN の内部だけでユーザの要求を満たすことができる場合には, その LAN から外にメッセージを一切出さないように設計されている. 他の LAN との通信が必要な時も, メッセージ数を必要最小限にとどめるためにサーバ間の通信はブロードキャストを用いず生成木 Tの枝に沿って伝達される.

4.2 キャッシュプロトコル

あるファイルのオープンとクローズにおける, DFS 全体での動作を以下で説明する.

4.2.1 オープン

ユーザからファイル fのオープンの要求を受け取ったクライアント Cは, まず自分のキャッシュ内に有効なものが存在するかどうか調べ, 存在するならばそれをユーザに提供する. そうでなければ, 自分が属する LAN のサーバ Sに対してファイル要求のメッセージを送信し, ある FW からファイルが転送されて来るのを待つ.

クライアント Cからファイル要求のメッセージを受け取ったサーバ Sは, 自分の属する LAN 内に FW Fが存在していれば, Fに対してファイル要求のメッセージを転送する. 自分の属する LAN 内に FW が存在しない場合には, ファイル fの FW が存在する方向にファイル要求のメッセージを転送し (図 1参照), クライアント Cを新しい FW であるとみなす. (なぜなら, Cには, ある FW からファイル fがいずれ到着

[2] この手法は, Non Write Back 方式 [6] の応用であり, 変更時にファイル転送を行なう必要がないので, ネットワークの負荷が軽減できると期待できる (文献 [5] を参照せよ).

する．従って，Cは，少なくともその FW よりは古くない fの版を持つことになる．) これにより，次にファイル fに対する要求が同じ LAN 内で発生した場合には，LAN 内だけで処理が可能となる．

他のサーバ S'からファイル要求を受け取ったサーバ Sは，自分が属する LAN 内に FW Fが存在するならば Fに対してファイル要求メッセージを転送し，そうでなければ，FW が存在する (S'以外の) 方向にファイル要求のメッセージを転送する．いずれの場合も，ファイル要求の転送後，Sは S'の方向 (すなわち，Cの方向) を新たに FW の方向の一つとして記憶する．

最終的にサーバからファイル要求メッセージを受け取った FW Fは，クライアント Cに直接ファイル fを転送する．Fが Cと同じ LAN に属する場合には，Fは Cを fの転送先として記憶し，fの無効化命令に備える．Fが Cと異なる LAN に属する場合には，Fは Cを記憶せず，ファイルを受け取った Cはその LAN 内で FW として動作する．

4.2.2 クローズ

ユーザからファイル fのクローズの要求を受け取ったクライアント Cは，サーバ Sに対してキャッシュの無効化命令を送信し，その成功を示すメッセージ Success あるいは，他により新しい版が存在することを示す無効化命令が到着するのを待つ．なお，Cが発行する無効化命令には，C, fと共に，クローズを起こした時刻 tが記載されている．メッセージ Success が到着したときには，Cは最新版を持つ FW として以後動作する．新しい版が存在することを示す無効化命令が到着したとき，すなわち，クライアント C'がファイル fを時刻 $t' > t$ にクローズしたことを記載した無効化命令が到着したとき，自分のキャッシュを破棄すると共に，(Cがもともと FW であった場合には) Cからファイルを複製した C'以外のクライアントの全てに対して無効化命令を転送する．

クライアント Cから無効化命令を受け取ったサーバ Sは，その時刻印 tを調べ，そのクローズが自分が知る限り最近のものであるならば，メッセージ Success を Cに返送し，(C以外の) WAN 内の知っている FW の全てに対してこの無効化命令を転送する．そして，以後 Cを FW と認識する．もし，Sに，あるクライアント C'がファイル fを時刻 $t' > t$ にクローズしたことを記載した無効化命令が到着していたならば，それを Cに転送する．

他のサーバ S'から無効化命令を受信したサーバ Sは，S'以外の FW の存在する方向の枝に対して，この無効化命令を転送し，それらの枝を FW の存在する方向を示す枝のリストから削除する．自分が属する LAN 内に FW が存在するときには，それに対してこの無効化命令を転送する．

最後に，FW から無効化命令を受信したクライアントは，現在セッションをオープンしているならば，セッションをクローズするまで待ち，FW から転送されたままの版が残されているなら (すなわち，FW から転送した後，セッションでは書き込みが起こらなかったならば) 自分のキャッシュを破棄する．なお，書き込みを含むセッションがクローズされたならば，その時点で，無効化命令が発行されサーバに送信されていたはずである．

4.3 キャッシュプロトコルの改良

本プロトコルでは以下の事態が起こると予想され，それがプロトコルの効率を引き下げる可能性があると考えられる．(事実，次節で述べるシミュレーション実験において，そのような事態が起こっていることが確認される．)

二つのクライアント Cと C'が同じファイル fをほぼ同時に更新しようとしたとする．Cと C'が別の LAN に属し，それぞれサーバ Sおよび S'によって管理されているとしよう．その LAN 間の通信遅延が大きいならば，C'が時刻 t'に起こしたクローズが S'を経由して Sに伝わる前に，Cが時刻 $t > t'$に起こしたクローズが Sで処理され，その結果，C'での変更が Sの LAN では反映されないということが起こる．(この事実は，セッションセマンティクスには抵触しない．) 従って，Cと C'での更新が頻発すると，それぞれの LAN で，整合性の無い版を最新版として長く保持する可能性が生じ，これは (セッションセマンティクスには抵触しないが) 都合が悪い．この状況を改善するために，以下の改良を施す．(紙面の都合から，概略だけを説明する．)

基本的な方針は，ファイルの変更に関する特権を持つサーバである，特権サーバを考えることである．特権サーバ S^*は WAN 内に唯一つ存在し，変更を伴うクローズをクライアント Cから要請されたサーバ Sは，(このクローズが最近のものであるならば) 先ず S^*から特権を譲り受けた後，メッセージ Success を返す．このことで，すべての変更を伴うクローズを一意に順序付けることができ，上記のような問題が回避できる．(この順序は，クローズが起こった時刻による順序付けと必ずしも一致するとは限らない．)

現在，この方針を次のように実現している．変更を伴うファイル fのクローズをクライアント Cから要

請されたサーバ Sは, Sから特権サーバ S^*までの経路上にある全てのサーバを, 無効化命令が通過してから S^*からの無効化完了のメッセージが戻ってくるまでの間, ロックし, ファイル fに関わる操作をその間行なえないようにする.

予想されるとおり (そして, 次節で述べるシミュレーション実験でも確認できるように) この実現法にするとアクセスの局所性が低くなり, ロックされるサーバが増すにつれて, アクセス効率が低下する.

この問題の解決法は現在検討中であり, 今後の研究課題とする.

5 シミュレーションによる比較評価

提案したプロトコルの有効性を示すために, シミュレーションによる実験を行なった. 比較の対象として, AFS と Sprite を用いた.

シミュレーションは論理時間で進められ, 各プロセスは 1 単位時間に 1 つのメッセージを受信して処理することができる. 但しファイルの受信に関しては 10 単位時間かかるものとする. 採用している共有の意味論の差を軽減するため, ファイルのアクセスはオープン, ファイル全体の読み込み, クローズが 1 回のアクセスで完了するものとする. 但し, クローズは一定の確率で書き込みを伴う. それぞれのクライアントでは, 先に出した要求が満たされ次第, 次の要求が発生する.

シミュレーションの対象である分散システムは 2 つの LAN から構成される WAN で, それぞれの LAN には 1 台のサーバと 10 台のクライアントが属している. DFS 全体のファイル数は 20 とする. 通信遅延は, LAN 内が 1 単位時間, LAN 間が 100 単位時間である. AFS と Sprite も各 LAN にサーバがあり, それぞれ DFS 全体の半分のファイルを管理し, クライアントはファイルに応じてサーバを選択するものとする.

このような条件の下で, 各プロセスが 10000 単位時間動作した時の, ファイルアクセスの総数と, アクセス 1 回あたりに必要としたメッセージ数の平均を計測した. アクセスの局所性 (同じファイルを続けてアクセスする確率) が 80 % の場合に, リード比 (アクセスが読み込みのみである確率) を変化させた場合の結果を図 2, 3 に示す. 図 2 は, クライアントの総アクセス数を表しており, 図 3 は, 一回のアクセスに必要としたメッセージ数の平均を表している.

アクセスの局所性が高い場合には, 提案プロトコルは AFS や Sprite のプロトコルの 2 倍程度のアクセス数を得ることができるが, リード比が低い場合, 4.3 節で述べたように, それぞれの LAN で同一ファイルに対する変更が頻発して, 互いを無効化し合うという状況が生じ, その結果, 総アクセス数は予想したほど良くなかった. 提案プロトコルの改良はこの問題を解消するために行なわれたのであるが, その結果アクセスの局所性が 50 % 以下になると, AFS とアクセス数が同程度に落ちてしまう. (図 4 : アクセスの局所性 40 % のときの総アクセス数を参照せよ.) この点の改良が, 残された課題である. しかし, いずれの場合も, 図 3 から分かるとおり, LAN 間で交換されるメッセージの数は従来のものよりかなり低く押えられており (しかしながら, LAN 内でのメッセージ数は従来のプロトコルの 1.5 倍程度必要である.) 初期の目的は達成されている. (AFS と Sprite では, LAN 内通信数と LAN 間通信数に差はなかったので, 図 3 では両方が同じ実線で表示されている.)

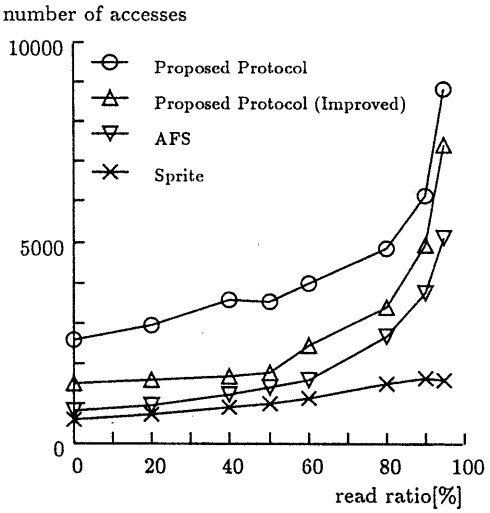

図 2: 総アクセス数 (アクセスの局所性 80 %)

参考文献

[1] M.Blaze and R.Alonso, "Dynamic Hierarchical Caching in Large-Scale Distributed File Systems," IEEE Computer, pp.521-528, 1992

[2] U.M.Borghoff, "Catalogue of Distributed File / Operating Systems," Springer - Verlag, 1992.

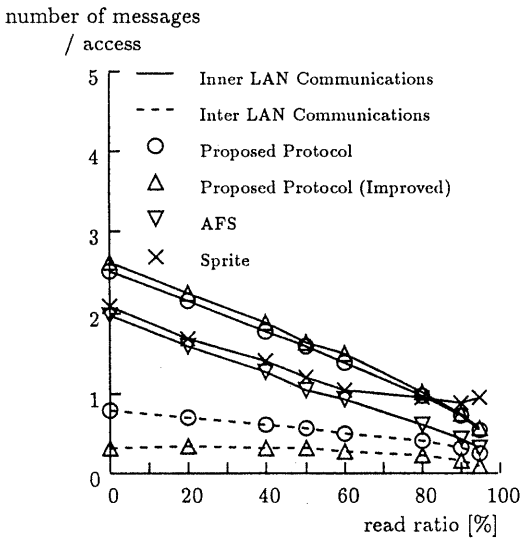

図 3: アクセス 1 回当たりの平均メッセージ数 (アクセスの局所性 80 %)

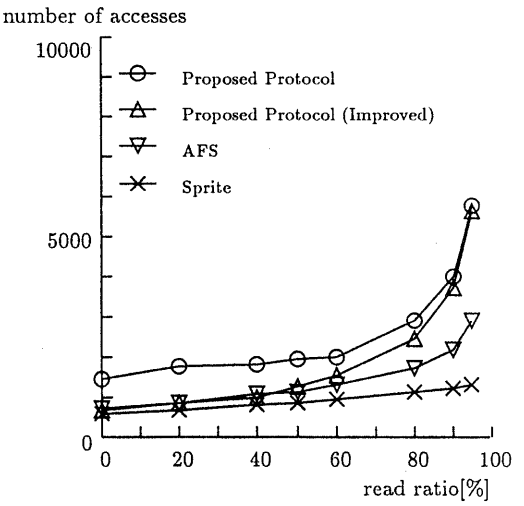

図 4: 総アクセス数 (アクセスの局所性 40 %)

[3] P.J.Denning, "Working Sets Past and Present," IEEE Trans. on Software Engineering, Vol. 6, No. 1, Jan. 1980, pp.64-84.

[4] D.Comer (村井 純, 楠本 博之 訳), "TCP/IP によるネットワーク構築," 共立出版, 1990.

[5] 稲荷 智英, 相原 玲二, 山下 雅史, 阿江 忠, "分散ファイルシステムにおけるキャッシュプロトコルの実験的評価," 日本ソフトウェア科学会ソフトウェア研究会 (関西) 資料, Apr. 1992, pp.39-46.

[6] 金沖 充也, 相原 玲二, 山下 雅史, 阿江 忠, "NWB 方式によるネットワークファイルシステム Bartch," 電子情報通信学会秋期全国大会 (D-86), 1990.

[7] C.A.Kent, "Cache Coherence in Distributed Systems," WRL Research Report 87/4, Dec. 1987.

[8] E.Levy and A.Silberschatz, "Distributed File Systems: Concepts and Examples," ACM Computing Surveys, Vol. 22, No. 4, Dec. 1990, pp.321-374.

[9] J.H. Morris, M.Satyanarayanan, M.H.Conner, J.H.Howard, D.S.Rosenthal and F.D.Smith, "Andrew: A Distributed Personal Computing Environment," Communications of the ACM, Vol. 29, No. 3, Mar. 1986, pp.184-201.

[10] J.K.Ousterhout, A.R.Cherenson, F.Douglis, M.N.Nelson and B.B.Welch, "The Sprite Network Operating System," IEEE Computer, Vol. 21, No. 2, Feb. 1988, pp.134-154.

[11] M.Satyanarayanan, "The Influence of Scale on Distributed File System Design," IEEE Trans. on Software Engineering, Vol. 18, No. 1, Jan. 1992, pp.1-8.