修士論文　　　　　1999年度(平成11年度)

Webサーバのための分散ファイルシステムの設計と実装

慶應義塾大学大学院 政策・メディア研究科
修士課程2年

坪井孝弘

tsuboi@tom.sfc.keio.ac.jp

修士論文要旨 1999 年度 (平成 11 年度)

Web サーバのための 分散ファイルシステムの設計と実装

本論文では，分散 Web サーバ環境を支援することを目的とした新しいシステムについて述べる．本システムは Web サーバ上に存在するすべてのプロセスを Web サーバとその他のプロセスに分離し，両者に対し異なったファイルサービスを提供することで，高性能な分散ファイルシステム環境を構築する．

インターネット利用者の増加と光技術の進歩により，ネットワークの高速化はますます進みつつある．しかし一方で，Web サーバなどの動作するエンドノードの計算機はネットワークほど高速化していない．これは技術分野の違いによるもので，今後ますます両者の差は大きくなる．この影響を最も受けるのが Web サーバである．

Web サーバの分散化を行うためには，共有ファイルの管理という問題が発生する．分散ファイルシステムは，ネットワーク透過なファイル空間を提供する．しかし既存の分散ファイルシステムでは，すべてのプロセスに対し同じサービスを提供しており，負荷の大部分が Web サービスによるものである分散 Web サーバ環境には適さない．

本システムでは，Web サーバに対して高速なファイルアクセスを提供し，その他のプロセスには共有ファイルの管理機能を提供する．本システムは，ネットワーク透過なアクセスを提供する分散ファイルシステム，Web サーバに分散ファイルシステムへの高速アクセス機能を提供するライブラリ，協調動作するためのメッセージ管理機構からなる．

本システムの評価には，既存の分散ファイルシステムのファイルアクセス時間を測定し，比較することで行った．その結果から，本システムは分散 Web サーバ環境において，既存システムである NFS，AFS よりも優れた性能を得ることができることを示す．

キーワード

1. 分散システム　2. ファイルシステム　3. ネットワーク透過性
4. ファイル一貫性制御　5. WWW

慶應義塾大学大学院 政策・メディア研究科
坪井 孝弘

Abstract of Master's Thesis
Academic Year 1999

Design and Implementation of Distributed File System for Web Server

This paper describes a new system intended to support distributed Web server. This system implements high performance distributed file system by separating all processes existing on Web server into Web server processes and other processes, and providing different file services for them.

As the number of Internet user has increased and optical technology has developed, network has been increasingly speeding up. However, the advance in technology of end node computer like Web server has not been speeding up as much as network has been. the difference between network and end node is becoming wider and will will most affect Web server.

Distribution of Web server has problem of administration of shared files. Distributed file system provides network transparent file space. However, existing distributed file systems provide uniform service for all processes, and are not adequate to distributed Web server in which the most of loads are caused by Web services.

Our system provides high-speed file access for Web server process and administrative function of shared files for other processes. The system consists of distributed file system which provides function to distributed file system for Web server, and message administrative function for cooperation.

For evaluation of the system, we measured and compared the performance of our system with existing distributed file systems. The result indicates that this system exceed the performance of existing systems in distributed Web server.

Key Words

1. distributed system 2. file system 3. network transparency 4. file consistency control 5. WWW

Keio University Graduate School of Media and Governance
Takahiro Tsuboi

目 次

表目次

図 目 次

第1章　序論

1.1　開発動機

計算機の急速な普及により，計算機をとりまく環境は急激に変化している．また技術革新により性能が大幅に向上した計算機は，今日の社会を構成する上で不可欠な存在となった．また，爆発的な計算機利用者の増加によりサービスを提供するシステムの並列分散化への要求をより強くしている．

マルチプロセッサの計算機で複数同時に処理を行うことを並列処理という．複数のプロセッサから構成されるマルチプロセッサシステムによって，処理速度やスループット，信頼性を向上させ単一プロセッサシステムに比べ高性能の計算機システムを実現できる．マルチプロセッサシステムはプロセッサとメモリ間のデータ転送速度が一定であることから，プロセッサの数をただ増やすだけでは処理性能の向上は難しい．キャッシュメモリを利用することにより，プロセッサとメモリ間のデータ転送量を減らすことで高速化する方法が取られている．

一方，複数の計算機を高速なネットワークで結合し互いに通信しあうようにした処理形態が分散処理である．分散処理の目的として，特に以下の項目を挙げることができる．

- 性能の向上

複数の計算機が利用できる時，計算機の利用効率を上げることでシステム全体としての性能が向上する．1台の計算機に与えられた処理すべてを実行させるのではなく利用可能な計算機に平行して処理を実行させることで，与えられた処理を短時間で実行できる．また，ある計算機が過負荷状態に陥ったとき，他の計算機で処理を実行させることによって負荷分散できる．

- 情報信頼性の向上

複数の計算機が同じ情報を持つことで情報が分散することにより冗長性が得られる．ある計算機で何らかの障害により情報が失われることがあっても，他の計算機でその情報が複製されているため，実際に情報が失われることは少なくなる．

- サービス継続可能性

同じ処理を複数の計算機に実行させせることによりシステムに冗長性を持たせることができる．1台の計算機に障害が発生しても他の計算機が処理を代わりに実行することでシステム全体の機能停止なく処理を継続できる．つまり危険が分散され，システム全体としてのサービス継続性が高まる．

分散処理を実現する上で最も重要な概念として挙げられるのがネットワーク透過性(Network Transparency)である．ネットワーク透過性とはネットワークの存在を利用者が意識する必要性のことを言う．ネットワーク透過性を持ったシステムでは利用者が物理的な分散を意識せずに計算機全体を1つのシステムのように利用することができる．しかし，分散システムにおいて完全なネットワーク透過性を達成することは容易ではない．

より完全な透過性をプログラミングレベルから達成し，より密に統合化された分散処理を行うには，最初から分散処理向けに設計された新しいオペレーティングシステムが不可欠である．このような目的で設計されたオペレーティングシステムのことを分散オペレーティングシステムという．これに対し，従来のオペレーティングシステムに分散処理機能を追加したもの，ノードごとに独立したオペレーティングシステムが存在し利用者がある程度意識して分散処理しなければならないオペレーティングシステムをネットワークオペレーティングシステムという．

既存のオペレーティングシステムを変更することなく，分散処理を支援するソフトウェアモジュールを追加することによっても分散処理は実現可能である．この場合，既存のアプリケーションに影響を与えることもない．分散処理の目的は多義に渡り，システムがどこに重点を置くかで，どういった機能をオペレーティングシステムで提供するかが決まる．しかしこれらの解決方法では，さまざまな特徴あるサービス提供のためのオーバヘッドによる処理効率の低下は避けられない．

1.2　既存の研究

分散環境で実行されるアプリケーションに対して統一されたネームスペース，保証されたアクセスとファイルの保護などを提供するために必要となるのが分散ファイルシステムである．分散ファイルシステムではリモートホスト上のファイルをあたかもローカルファイルのようにアクセスすることが可能となる．

分散オペレーティングシステムはネットワーク透過性を実現するために多くの困難な分散処理機構を実装しなければならい．それに比べ，その構成要素の一部としての役割も果たす，比較的現実的な分散ファイルシステムの実装はこれまでにもいくつか行われ実際に利用されてきた．

- **NFS**

NFS(Network File System)[1] は，Sun Microsystems 社 [33] が開発したリモートファイルシステムへの透過なアクセスを提供するプロトコルである．Sun はプロトコルの発表時に，いくつかのオペレーティングシステムに NFS を実装し，その使用を許可した．そのため広く受け入れられ，UNIX でないシステムでも利用可能であり，事実上の業界標準になった．NFS アーキテクチャはクライアントサーバモデルに基づいている．NFS の設計目標を以下に示す．

- UNIX に依存しない
  どのオペレーティングシステムも NFS をクライアント，サーバ共に実装可能とする．
- 特定のハードウェアを必要としない
  プロトコルはどの特定ハードウェアにも依存しない．
- 既存のアプリケーションがそのまま動作
  アプリケーションは特定のパス名やライブラリを使用することなく，再コンパイルもなしに透過的にリモートファイルにアクセス可能である．
- UNIX ファイルセマンティックスの維持
  UNIX ファイルセマンティックスは UNIX クライアントのために維持されなければならない．

NFS は RPC(Remote Procedure Call 遠隔手続き呼び出し)[3] により，クライアントとサーバ間の相互作用するフォーマットを定義する．XDR(External Data Representation 拡張データ表現)[4] は，ネットワーク上で送られるマシン非依存なデータを提供する．

NFS の特徴として，サーバのステートレス性が挙げられる．サーバはクライアントについての情報を全く持たず，各要求ごとに独立している．ステートレスプロトコルの利点はサーバクラッシュからの復旧を容易にする．

NFS のリリースはバージョン 2[2] からであり，バージョン 3[5][6] も普及しつつある．

- **AFS**

AFS(Andrew File System)[8][9] はカーネギーメロン大学 (CMU)[32] と IBM[34] が共同で開発した分散ファイルシステムである．AFS の目的は 7000 台以上のワークステーションに提供される共有ファイルシステムを実現することであり，プロトタイプシステムは約 400 の組織で利用された．規模の他にも AFS の設計者は次のような重要な目標を示した [10]．

- UNIX 互換性
  動作するクライアントを限定しないため既存インタフェースを利用している

- 位置透過性
  ユーザはどこから利用しても同じ名前空間を利用できる
- 利用者の可動性
  ユーザは気軽に移動できる
- フォールトトレラント
  1つのサーバやネットワークコンポーネントの停止ではシステム全体を停止しない
- 広域ネットワークを考慮したセキュリティ
  信頼できないネットワーク上においても，セキュリティを提供できる

AFSでは他にも以下のようなスケーラビリティを得るためのいくつかの対応が行われている．

1つのサーバが多くのクライアントを扱うことは，サーバの過負荷，ネットワークの過負荷を引き起こす．AFSではネットワークをいくつかの独立したクラスタに分けることで対応している．このクラスタは動的に再構成が可能となっており，各ネットワークセグメントの状況に応じた変更が可能である．また，同じ計算機がサーバとクライアントを同時に行うことがないこともAFSの特徴である．

ファイルのキャッシュを積極的に行うことにより，ネットワークトラフィックを小さくすることができる．AFSのクライアントはローカルディスク上に最近アクセスされたファイルをキャッシュする．初期の実装ではファイル全体をキャッシュしたが，あまりに大きなサイズのファイルをキャッシュすることは実用的でない．AFS3.0ではファイルを64キロバイトごとに分けてキャッシュする．

AFSではクライアントがシンボリックリンクとディレクトリの両方をキャッシュする．パス名ルックアップ操作をサーバからクライアントへ移すことで，名前解決のためのサーバ負荷とネットワーク負荷を減らす．

セッションセマンティクスの一貫性保証はUNIXセマンティクスを実現していない．AFSではread(),write()ごとにデータ有効性を照合する．ファイルが閉じられるまでクライアントはサーバに変更をフラッシュしない．

その後AFSの研究はTransarc[36]に移り，次に説明するDCEのDFSに進化した．

- **DFS**

OSF(Open Software Foundation)はDCE(Distributed Computing Environment)[12]の分散ファイルシステムとして，AFSの技術を取り込んだDCE DFS(Distributed File System)[13]を採用した．DFSはAFSの以下の点を改善した．

- 1つの計算機がクライアントとサーバになることを許す
- より強いUNIXの共有セマンティクスと一貫性保証を提供する

- 他のファイルシステムとの相互操作を可能とする

DFS の特徴を以下に示す．

- スケーラビリティ
  ファイルデータをローカルディスクにキャッシュすることでネットワーク負荷を抑える．特にネットワークが高負荷時に効果を発揮する．
- グローバルディレクトリツリー
  どのホストからアクセスしても同じディレクトリツリーで見える．
- アクセス制御
  ACL(Access Control List) を利用したアクセス制御が可能である．
- 高信頼性
  サーバデータのコピーを作成でき，サーバに障害が発生してもそのコピーをサーバの代わりとして利用できる．

DFS サーバのローカルファイルシステムには Episode ファイルシステム [14][18][19] が開発された．Episode は高い利便性を提供し，論理ボリュームと POSIX 互換の ACL に対応する．DFS は Episode 以外のファイルシステムにも対応しているが，ACL によるアクセス制御が利用できないなどの制限が発生する．

DCE の拡張機能という位置づけであるため，DCE 上のアプリケーションを実行することが目的の場合には，DCE 環境でも導入されないことがある．しかし，逆にファイルが分散されたコンピュータ間で信頼性高く安全に共有できれば特別に DCE 対応のアプリケーションを構築することなく DCE 導入の目的を達成できることも多い．

分散ファイルシステムとしてネットワーク透過なファイルアクセスを提供しようとするシステムの研究は多く存在する．以上に挙げたものはその一例に過ぎないが，すべて一般に普及し実用に耐えてきたシステムであり，システムごとにさまざまな工夫がなされ，それぞれ異なった特徴を持っている．しかし利用されている分だけ問題点も見つかっている．

1.3 既存研究の問題点

既存システムの問題点を以下に挙げる．

• ネットワークインタフェースを通過する情報量の増加

1 台のファイルサーバがすべての情報を管理し，クライアントは必要になるたびにネットワークを介してファイルサーバへ必要な情報を要求するようなシステムでは，高負荷時

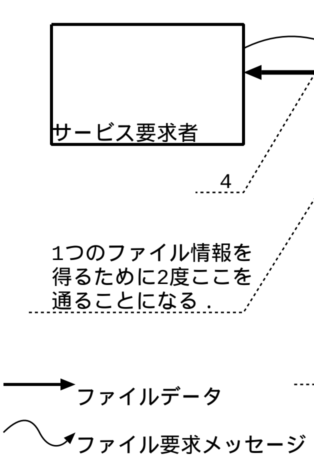

図 1.1: ファイルサーバがすべての情報を保持するシステムの問題

の性能低下が大きい．ネットワークインタフェースはローカルディスクよりも低速なデバイスである．このネットワークを利用するシステムでの高速処理は期待できない．

そのクライアントでサーバ処理を行うとその問題はより深刻である．要求された情報を要求元に送り終えるまでに，同じ情報がネットワークインタフェースを 2 度通るからである．NFS や AFS のキャッシュミス時がこの例である．図 1.1 に処理イメージを示す．

1. 要求元はサーバの動作するクライアントホストへある情報を要求する
2. クライアントホストはファイルサーバへ情報を要求する
3. ファイルサーバは要求された情報をクライアントホストへ返す
   情報がネットワークインタフェースを通って入ってくる
4. クライアントホストは要求された情報を要求元へ返す
   情報がネットワークインタフェースを通って出ていく

また，このシステム構成は高負荷時のファイルサーバへの負荷集中という問題もある．AFS や DFS ではこの問題はかなり改善されている．NFS の高負荷状況でのサーバへの負荷は大きく，性能低下も大きい [11]．

● キャッシュの復元にかかるオーバヘッド

ネットワークを介した処理は分散ファイルシステムにとって重要な要素であり，性能低下の原因となる．上述のような問題も引き起こす．ネットワーク処理を減らすために，ディスクキャッシュが用いられるのが一般的である．しかし，そのディスクキャッシュをファ

イル上に保存すると，その性能はローカルファイルシステムに及ばない．なぜなら，元のファイルへ復元するためのオーバヘッドがかかるからである．このオーバヘッドはキャッシュシステムにより増減する．

AFSではサーバへの負荷集中を抑制するためにブロック単位でファイルをキャッシュする．しかし，このブロックからファイルへの復元のためのオーバヘッドにより高速なローカルファイルシステムと比べるとかなり遅いものとなっている．

当然キャッシュシステムでは，キャッシュヒットしなければネットワークを介した負荷を減らせない．

- ファイルサービスの提供範囲

既存の分散ファイルシステムはどれも無限に存在するファイルアクセスパターンのすべてに対応しようとしている．しかしすべてのアプリケーションに効率の良い実装は困難である．

本研究では以上に挙げた問題点に注目し，解決するシステムの構築を目指す．

1.4　新しい分散ファイルシステムの提案

本研究には以下のような手法を利用する．これらは上述した問題点を解決するための提案である．

- すべてのクライアントがファイルサーバとなる

ネットワークインタフェースを通過する情報量を減らすためには，ファイルサーバへのアクセスを減らす必要がある．すべてのクライアントがローカルディスクに同じ内容の情報を保持し，ファイルサーバ化することにより，ファイルに変更がない限りネットワークを介した処理は発生しない．イメージを図1.2に示す．

- 高速処理が可能なファイルシステムを直接利用する

キャッシュの復元にかかるオーバヘッドは一旦ファイルシステム上に保存した情報を復元することにより発生する．直接高速なファイルシステムに保存することでそのオーバヘッドをなくすことができる．

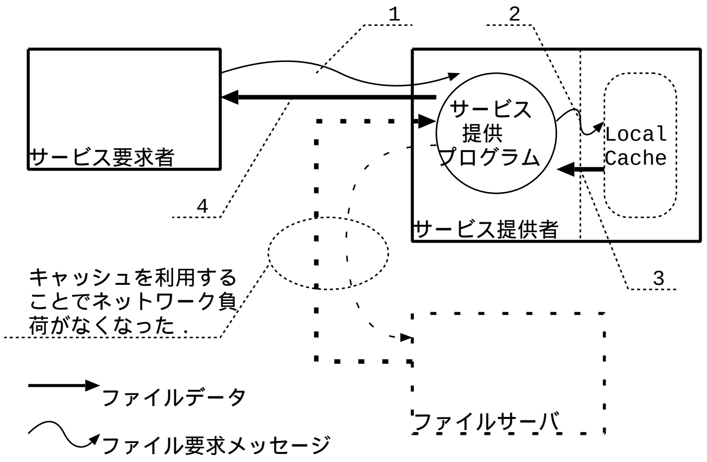

図 1.2: ローカルにキャッシュを持つシステム

- 用途に応じたファイルサービスの提供

どのような性質を持つか推測不明なすべてのプロセスに同様なファイルサービスを提供することは，性能向上のためにならない．ある性質を持つサービスに最適化することで，そのサービスの性能向上が可能である．

本論文ではこれらを実現する分散ファイルシステムを設計，実装し，その性能を評価する．

1.5 本論文の構成

1 章では既存研究の問題点を示し，その解決方法を提案した．

2 章では本研究を行う直接的な動機となったインターネットをとりまく環境の変化について，社会的変化，技術的変化についてそれぞれ説明する．

3 章では本研究で提案するシステムの設計について詳細に述べ，続く 4 章ではその実装について述べる．

5 章では実際に本システム上で行った実験結果を示す．類似システムの測定も行い，その結果を元に分散 Web サーバ環境の比較評価，考察を行う．

最後に 6 章で結論を述べる．

第2章　背景

2.1　インターネットを取り巻く環境の社会的変化

インターネットを取り巻く環境はここ数年急激に変化している．これからも多様に変化し続けるが，どのように変化するかを予測しなければならない．そのためにも現状の把握は必要である．本節では，インターネットを取り巻く環境の社会的変化について述べる．

● PCの低価格化

インターネットを利用するために必ず必要となるのが，PC(Personal Computer)や携帯情報端末など，ディジタル情報を処理する端末である．一般家庭ではPCの端末としての利用が大半である．PCはインターネット利用機器の中で多くの機能を持つ．

最近はPCもインターネット利用端末として使用される割合が大きくなってきているが，元々の事務処理や文書処理，表計算処理などの仕事をする上でも欠かせない存在である．

このPCの低価格化が一段と進んできた．それまでの低価格なPCは，高価格な新製品に比べると安い程度だったが，最近では安さだけを追求したPCも登場し，以前では購入できなかったり，購入しようとしなかった人も購入を考えたり実際に購入したりするようになった．

さらに現在のPCは安価だけではなく性能も伴っている．低価格PCとこれまで通りのPCとを比べると，値段は2倍以上の開きがある機種も処理性能は2倍に届かない．事務処理などの用途になると性能はほとんど変わらない．高級機と低価格PCとの性能差がかなり縮小してきている．

以上のように，PCの低価格化を見ることができる．

● ネットワークの高速化

ネットワークの伝送速度は，インターネットに様々な影響を与える重要な要素である．

第一に挙げられる要素にインターネットの可能性を決めることがある．初期のインターネットではWebなどのハイパーリンクをクリックしたら即座に次の情報を得る必要のあるアプリケーションや，リアルタイムで最新情報を提供することを目的とするサービスのようなソフトウェアを利用することは実現不可能であった．

次に挙げられる要素がインターネットの快適性を決めることである．インターネットの

快適さとは，例えばWebでハイパーリンクをクリックしてから次のページが表示されるまでの時間間隔や，サイズの大きなファイルの転送時間がどの程度小さいかで決められる．この速度の評価は誰が利用するかによっても異なる．速ければ速いに越したことはないが，人が直接利用する場合は，一定以上の高速化は体感できないためあまり意味がない．

このようにネットワークの伝送速度はインターネットに与える影響は大きい．伝送速度向上への要望も大きくなる．その要望に答えるようにネットワークは高速化してきた．

- インターネット利用者の増加

インターネットはコミュニケーション手段という役割が大きい．新規利用の接続相手は現在のインターネット利用者全員である．また，新規利用者が増えた分だけ元の利用者全員の接続相手も増えることになる．利用者が増えはじめると，その増えること自体がさらなる利用者増加の要因になる．

インターネット利用者が増えると，そこでその利用者をターゲットとしたサービスがはじまる．サービスが増えると，そのサービスを利用したいと思う人はインターネット利用者になる．インターネットを利用したサービスは，提供する側のサービス提供コストを抑えることになる．例えばこれまでのカタログ通信販売は，Webを使えばいつでもどこでも最新情報を提供できる．これによりカタログ配布コストを削減でき，浮いたコストでさらなるサービスを提供できる．サービス利用者も最新の情報を得ることができるのはインターネットだけなので，その魅力は大きい．

このようにインターネットの利用者は増えはじめると止まらない構造になっている．インターネット利用者の増加自体がインターネット利用者増加の原因になる．

- **PC**購入者層の変化

以前のPCは白い筐体の大きな箱というようにどこのメーカを見ても決まっていた．しかし，最近のPCの低価格化により性能で利益を得ることが困難になったPC製造業者がデザインで利益を得ようとするようになってきた．この傾向により，PCの低価格化同様にこれまでPCに興味を持たなかった人のPC購入を推し進める結果となった．現在のPC購入者はほぼ間違いなくインターネットを利用する．以前のインターネットを利用するまでのさまざまな面倒な手続きも現在のPCでは不要になった．また，インターネットをとても簡単に利用するアプリケーションであるWebのコンテンツの充実により，インターネットを利用するということ自体の敷居が低くなったこともインターネット利用者増加の原因となっている．

- インターネット接続機器の多様化

携帯電話はすでに多く普及している．最近は携帯電話からさまざまな文字情報をリアルタイムで取り出せるサービスが現れてきた．この文字サービスはインターネットを利用したものであることが多い．すでに無意識のうちにインターネットを利用しているという時代になりつつある．

現在の携帯電話の通信速度はかなり低いが，近い将来この通信速度が200倍以上になる．これは現在以上のサービスを提供できることを意味し，さらなる利用者増加が予想されている．

これらの社会的変化は今後も当分の間継続することは間違いない．それどころか変化要因はますます増えることになる．次に技術的変化について述べる．

2.2 インターネットを取り巻く環境の技術的変化

インターネットを取り巻く環境は今後数年で急激に変化する．本節では，社会的変化と異なり理由が明確である，インターネットを取り巻く環境の技術的変化について述べる．

- 外部ネットワークの高速化

インターネットの普及に伴い，基幹回線(バックボーン)に使用される新しい光通信技術が広がってきた．OTDM(Optical Time Division Multiplexing 光時分割多重)技術とWDM(Wavelength Division Multiplexing 波長分割多重)技術がそれである．

TDMは，1つの波長を使い信号の間隔を出来るだけ短くする方式である．一定の 時間内に光ファイバ内を通過する情報を増やす．これは伝送路を時間で分割し，情報の多重化を図る方法と言える．OTDMでは信号を詰め過ぎると長距離伝送した時，信号が読み出せなくなるため10ギガビット/秒が限界とされる．

WDMは，複数の波長の光を1本の光ファイバに通して信号を多重化する方法である(図2.1参照)．束ねる光の波長の数に応じて容量を簡単に増やしていけるところに大きな特徴がある．現在の技術では50波程度までの多重が可能とされ，既存の光ファイバケーブルをWDM装置に接続するだけで伝送容量を50倍にもすることができる．

これらの技術の注目すべきところは，両者に相乗効果があるところである．TDMは1本の波長を大容量化する．WDMはその本数を増やす．ネットワークは光技術によりギガビット化した．しかし，もうテラビットが見えている．すでに実験段階では50波の光信号にそれぞれ20ギガビット/秒の情報を乗せることで1テラビット/秒の情報を1本の光ファイバで伝送することに成功している．これらの光技術のためにネットワークの高速化はもはやエンドノードにある計算機をはるかにしのぐ伝送容量を得ることになる．TDMとWDMの現状を表2.1に示す．

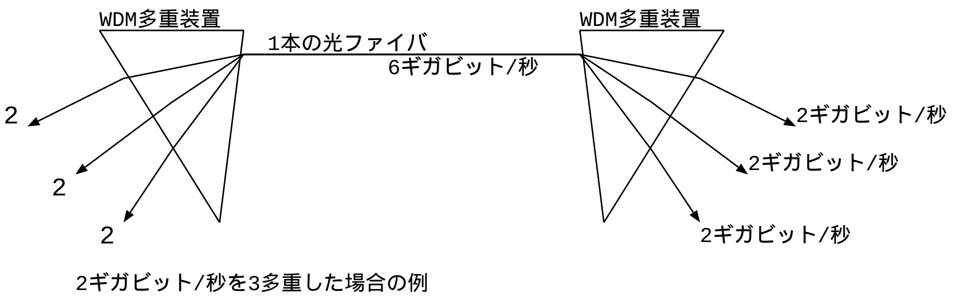

図 2.1: WDM(波長分割多重) 技術

光技術	高速化手法	伝送容量
TDM	1 波の通進路を多重化	~10 ギガビット/秒
WDM	1 本の光ファイバに波を多重化	~50 倍
TDM と WDM	TDM の容量× WDM の本数	~500 ギガビット/秒

表 2.1: 光多重通信技術 (TDM と WDM) の現状

- エンドノードの限界

PCI(Peripheral Component Interconnect) バスは PC 用の汎用バスとして Intel[35] が開発し，現在は PCI SIG(PCI Special Interest Group)[28] が管理している高速なバスアーキテクチャである．PCI バスは複数のデバイスが時分割で転送を行う．ブリッジと呼ばれる回路を使い接続していくことが特徴であり，さまざまなアーキテクチャを統合していくことが可能となっている．具体的には，PC/AT 互換機ではほとんどの機種でアーキテクチャの異なる ISA(Industory Standard Architecture) バスを PCI-ISA ブリッジを設けてサポートしている．ノート PC では，PCI-PCMCIA ブリッジを設けて CardBus もサポートする．CPU のローカルバスへは HOST-PCI ブリッジで接続するので，CPU のバスクロックは関係ない．PCI が足りなくなれば PCI-PCI ブリッジで PCI バス自身の拡張も可能である．この他にもデバイスのオートコンフィギュレーション機能やパワーマネージメント機能，バスマスタリング機能をサポートし，PC/AT 互換機だけでなく，ワークステーションにも採用されている．

現行機種にはバス幅 32 ビットの規格が採用されており，バスクロックは 33MHz のものが主流である．この場合の伝送容量は 1 ギガビット/秒になる．仕様上ではバス幅 64 ビット，バスクロック 66MHz まで対応しており，4 ギガビット/秒の伝送容量に達する．

しかし，外部ネットワークがますます高速化しようという時にエンドノードの伝送容量を決定すると言っても過言ではない PCI バスの伝送容量は仕様上でも 4 ギガビット/秒し

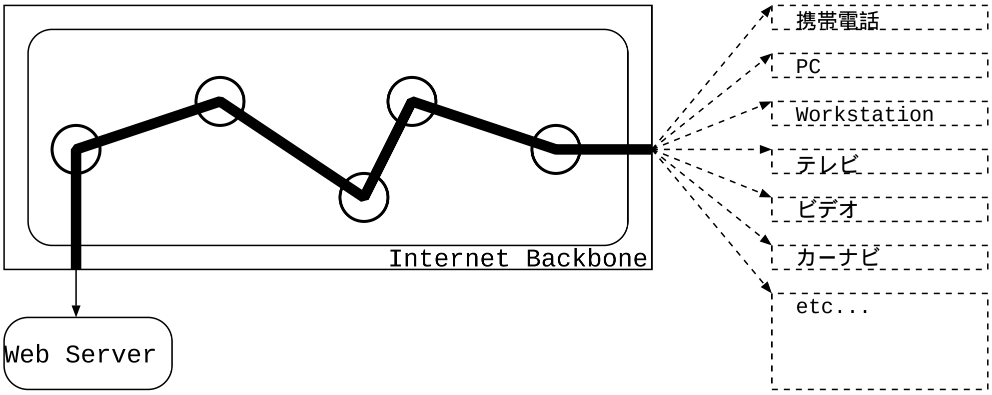

図 2.2: インターネットバックボーンとエンドノードの現状

かなく，3 桁以上の違いである．このようにエンドノードは外部ネットワークほどの高速化は今のところ期待できない．図 2.2 に，光技術により高速化したバックボーンを持つインターネットとそれを取り巻く環境のイメージ図を示す．

2.3 本研究の必要性

光技術の進歩により，外部ネットワークは超高速化する．この高速化するネットワークを使いきるだけのインターネット利用が見込まれるかという問いに対する答えは，インターネットを取り巻く社会的環境変化を見るとはっきりしている．ネットワーク負荷は増加要因ばかりで，今後も変わることはない．エンドノードの計算機は分散化しなければ対応しきれない．

特にこれらの直接影響を受けることになるのが間違いなく Web サーバである．最近になって現れた Web を利用する新たなサービスにオンラインバンキング，手数料自由化に伴うオンライントレード，電子商取引など挙げるときりがない．これらにはセキュリティという重要課題に注目されがちだが，負荷が集中した時対応が出来ていないのが実情である．インターネットを利用した安全なサービス提供が確立されたら，その時はさらに多様なサービスが Web を通して行われるようになる．Web サーバのインターネット上での位置づけはますます高まる一方である．

Web サービス環境にはいくつか特徴がある．その特徴を生かしたシステムを構築する．

2.4 Webサービスの特徴

本研究ではネットワークの高速化に対応できなくなったエンドノードのWebサーバを分散化による負荷分散により対応することにした．分散システムの研究はこれまで多く行われてきた．次世代オペレーティングシステムとして分散オペレーティングシステムが挙げられていたこともあった．しかし，その頃とはインターネット環境が豹変している．特にWeb環境の変化は著しい．本研究はWebサービスに特化することにより，既存システムにない特徴を実現する．そのためにはWebサービスの特徴を把握しなければならない．以下に特徴を挙げる．

- **Webサーバの作業はファイルの読み込みが圧倒的に多い**

Webサービスは基本的に情報を閲覧することが目的で広まった，インターネットを利用したさまざまな情報を発信するための仕組みである．最近ではネットワークの高速化やWebブラウザの機能向上などにより，写真，音，映像なども発信されている．Webサーバから取ってきた情報をWebブラウザを使い，ハイパーリンクされたテキストや画像をポインティングデバイスで押すことにより，無限の情報が得られる．このポインティングデバイスとボタンという容易なインタフェースだけで閲覧可能なWebサービスは，PCの普及にも貢献しており，誰でも理解が容易である．

この特徴からわかるように，Webサーバの仕事はファイルをディスクから取り出し，その情報をWebブラウザに送信するという単純なファイル読み込み作業が大部分を占める．CGIフォームなどを利用して入力を促すページなども存在するが，その処理を見ても，ページをまず表示するためにファイル読み込み作業が発生している．そのCGIフォームを入力してもファイルの書き込みが発生することは少ない．検索サービスなどのプログラムもファイルとサーバとの処理内容は読み込みばかりである．

このようにWebサーバの特徴として，ファイル読み込み処理が大部分で，書き込み処理はまれなことが挙げられる．

- **Webサーバ以外からのアクセスは少ない**

Webサーバの負荷分散は，外部ネットワークの高速化によりWebサーバのネットワークインタフェースをはるかに越える負荷に対応できない状態に陥ることへの解決方法であった．それは，前提からWebサーバのプロセスによってシステムは十分多くのアクセスがあることを意味する．つまり，Webサーバ以外からのアクセスは少ないことをあらわしている．

- **Web サーバは最新のファイルで答える必要はない**

Web サーバの注目すべき大きな特徴であり，本研究に大きく関わる性質である．Web サーバはインターネットを利用することから，元々接続可能性を保証することは不可能である．

このように Web サーバには，近い将来も変化しないことが予想できる特徴を持っている．これらの特徴を利用し，本システムを設計する．

第3章　設計

3.1　設計目標

本研究は，Web サービスに特化することにより高性能な分散計算機環境を低負荷で実現することである．設計目標を以下に示す．

● Web サーバ以外のネットワーク負荷を抑える

エンドノードである Web サーバはネットワークインタフェースの伝送容量が外部ネットワークより小さい．本システムはエンドノードの分散システム化により，外部ネットワークの超高速化に対応する．分散システム化による情報がネットワークの多くを占めると，台数による性能向上効果が低下する．分散システム同士のメッセージ通信量は，台数増加分以上に大きくなる．2 台なら互いに 1 台と通信しさえすればいいが，3 台になると他の 2 台と通信することになり，単純計算で 1 台あたりの負荷は倍増する．

本システムを設計するにあたり，ネットワーク負荷を抑えることは，本システム全体の性能向上を意味し，重要である．

● Web サーバは高速，低負荷で処理

Web サーバの負荷の大部分を占めるのは，Web サーバプロセスの処理時間である．それもファイル入出力処理の多い Web サーバでは，CPU 負荷を抑えるよりもファイル入出力時間を短縮する方がより性能を向上させる場合が多い．

Web サーバのアクセスするファイルシステムには，できる限り高速なファイルアクセス時間を提供できることが，システム性能の向上を意味し，重要である．

● Web サーバ以外のアクセスには一貫性を保証

共有ファイル環境を提供しない分散システムでは，ファイルを変更するたびに他のサーバへコマンドを実行するなどして反映させなければならない．そのような処理を提供してくれるアプリケーションも存在するが，ファイルシステムによる対応を行うシステムと比べると，格段に一貫性制御能力は落ちる．そのようなアプリケーションの実装方法はファイルの更新時間などをすべてのシステムで調べることで，ファイルの変更を知る．この手

法の問題点は多く解決困難である．まず，ファイルの数が大きくなれば，その比較のための情報収集にかかる負荷はかなり深刻である．また比較のためのネットワーク負荷も膨大である．さらにCGIアプリケーションなどでは，変更をすぐに他に伝えなければ動作しないものも確実に存在する．

性能向上を目指す分散Webサーバ環境にとって，Webサーバ以外のアクセスに一貫性を提供することは，重要である．

- **Webサーバ以外のアクセスには既存環境を提供**

すべての既存アプリケーションを変更することで，高速なアクセスを提供するシステムも構築可能かもしれない．しかし，それはすべてのアプリケーションのソースコードが手に入る場合であるが，それは現実的ではない．バイナリパッケージでしか提供されないアプリケーションは商用の製品に多い．

また，もしソースコードが手に入ったとしても，頻繁に作業環境としては利用されないことが前提のWebサーバシステムのために，必要なものだけとはいえ再コンパイルを要求することは難しい．コンパイルがすべて簡単に通る保証もない．

以上の理由から，Webサーバ以外のプロセスへ現在の環境を提供することは，重要である．

これらの目標を実現するためには，Webサーバのアクセスと Webサーバ以外のアクセスに対して，異なったサービスを提供することが必要である．

3.2 設計方針

ネットワークインタフェースの負荷を抑えるためにはローカルディスクにキャッシュを持つことが効果的である．しかしキャッシュシステムの場合，キャッシュへのアクセス時間と元のファイルへの復元コストの和が実際のアクセス時間となる．元のファイルへの復元コストの分だけローカルファイルよりもアクセス時間がかかる．

そこで，本システムではすべてのファイルをローカルディスクにファイルとして持つことにする．ネットワークインタフェースの負荷を抑えるためにすべてのホストはファイルサーバとする．これにより，キャッシュファイルからの復元コストをなくすことができるが，同時に共有ファイルの一貫性制御という処理が必要になる．

共有ファイルの一貫性を提供するためには，ファイルシステムレベルによる対応が効果的である．ファイルの変更があった時，その変更されたファイルがどれなのかを，まさに変更時に知ることができ，すぐに反映させることが可能である．ユーザ空間でなくファイルシステムにより一貫性制御を提供することは，既存のアプリケーションへの変更が必要ないという利点もある．しかし，同時に通常のファイルアクセスよりも一貫性制御処理の

ための時間が追加され，全処理時間が確実に大きくなる．

ここでWebサービスの特徴を利用することが本システムの特徴である．ホスト上で動作するすべてのプロセスに対して同等のサービスを提供することは，これまで既存の分散ファイルシステムが実現してきており，研究も進んでいる．同じ条件で大きな性能向上を実現することは困難である．

Webサービスの特徴は，必ずしも最新のファイルを提供する必要はないことである．これを利用することで，Webサーバの負荷の大部分を占める処理に対してWebサーバに適した一貫性制御を提供可能である．この一貫性制御を利用したファイルアクセス時間を既存の分散ファイルシステムのファイルアクセス時間より小さく出来れば，サーバ全体の高性能化につながる．なぜなら，Webサーバの特徴によりWebサーバ以外の処理時間は十分小さいからである．

そこで本システムは3つの機構にわけることで，共有ファイルの一貫性制御を実現する．

- **メッセージ管理機構**

共有ファイルの一貫性制御とは，ファイルの変更操作が起こると他のサーバへその変更を反映させることである．変更を管理するために変更内容を通知するメッセージを利用する．このメッセージを管理し，必要な場所へ送受信し，そのメッセージの通り処理することがメッセージ管理機構の役割である．

すべてのメッセージには操作時間が付けられ，メッセージを受け取った時間ではなく操作時間により，メッセージが有効かを判断する．そのためには，各ホスト間の時刻が正確でなければならない．時刻調整もメッセージ管理機構の役割である．

各サーバのメッセージの送受信も各サーバ上で動作するメッセージ管理機構同士で行われ，受信したメッセージに従ってファイル反映処理を行うのもこの機構である．

- **ライブラリ**

Webサーバに適した一貫性制御を提供するためのソフトウェアモジュールであり，一貫性制御処理の代わりに高速なアクセスを提供する．このライブラリを利用することで，ファイル変更操作が起こった時のみ，ファイル変更メッセージを送信する．ヘッダファイルとライブラリで提供されており，本ライブラリを利用するためには，ヘッダファイルをインクルードし，Webサーバを再コンパイルする必要がある．

ファイル変更時にメッセージを生成するのは，分散ファイルシステムの一貫性制御を支援するためである．本ライブラリを利用した場合のファイル変更操作は一定時間経過するか，負荷の小さい時まで反映されない．しかし分散ファイルシステムからそのファイルへのアクセスがあれば直ちに反映するようになっている．

処理機会は非常に多いが，書き込み処理の割合はごくわずかであり，パス名変換コストを小さく実装することが性能向上のためには重要である．

項目	分散ファイルシステム	ライブラリ
ターゲット	Web サーバ以外のプロセス	Web サーバ
主目的	共有ファイルの一貫性を提供	高速なアクセス処理時間を提供
アクセス時間	分散ファイルシステム程度	ローカルファイルシステム程度
ファイル変更時	変更はすぐに反映	変更はいつか反映
処理機会	ごくまれ	非常に多い
書き込み操作の割合	比較的多い	非常に少ない
利用準備	必要なし	ソースコードの再コンパイル
メッセージ	常に生成	書き込み時のみ生成
パス名変換コスト	非常に小さい	小さい

表 3.1: 分散ファイルシステムとライブラリの特徴

- 分散ファイルシステム

本システムの中心となるファイルシステムであり，Web サーバ以外からのアクセスを処理する．アクセス数自体が少ないことを前提としており，実用に耐える程度の処理速度は必要だが，基本的に低速で構わない．共有ファイルの一貫性制御を提供することが重要である．ファイルシステムであるから，ここにアクセスするプログラムに何ら変更を必要としない．

このファイルシステム上でファイル変更操作が起こると，その変更はただちに他のサーバへ反映される．Web サーバ以外のプロセスによるファイル変更はすぐに反映されるため，コンテンツの作成など，どのサーバで作業しても良い．

分散ファイルシステムとライブラリの特徴を表 3.1 にまとめ，メッセージ通信とファイル通信のイメージを図 3.1 に示す．

3.3 マウント操作

ファイルシステムとしてディレクトリツリーへ張り付けることをマウントという．分散ファイルシステムはカーネルに組み込まれているだけでは何もできない．ローカルファイルシステムをマウントすることで分散ファイルシステムとして機能するようになる．

分散 Web サーバ環境として共有ファイル空間を構築するためには，分散 Web サーバ環境で 1 つの共通のマウントポイントと，実際のローカルファイル空間が必要になる．

例えば，/www_shared ディレクトリ以下に共有ファイル空間を構築するとして，そのためのローカルファイル空間を/export/www として分散 Web サーバ環境を構築したとする．

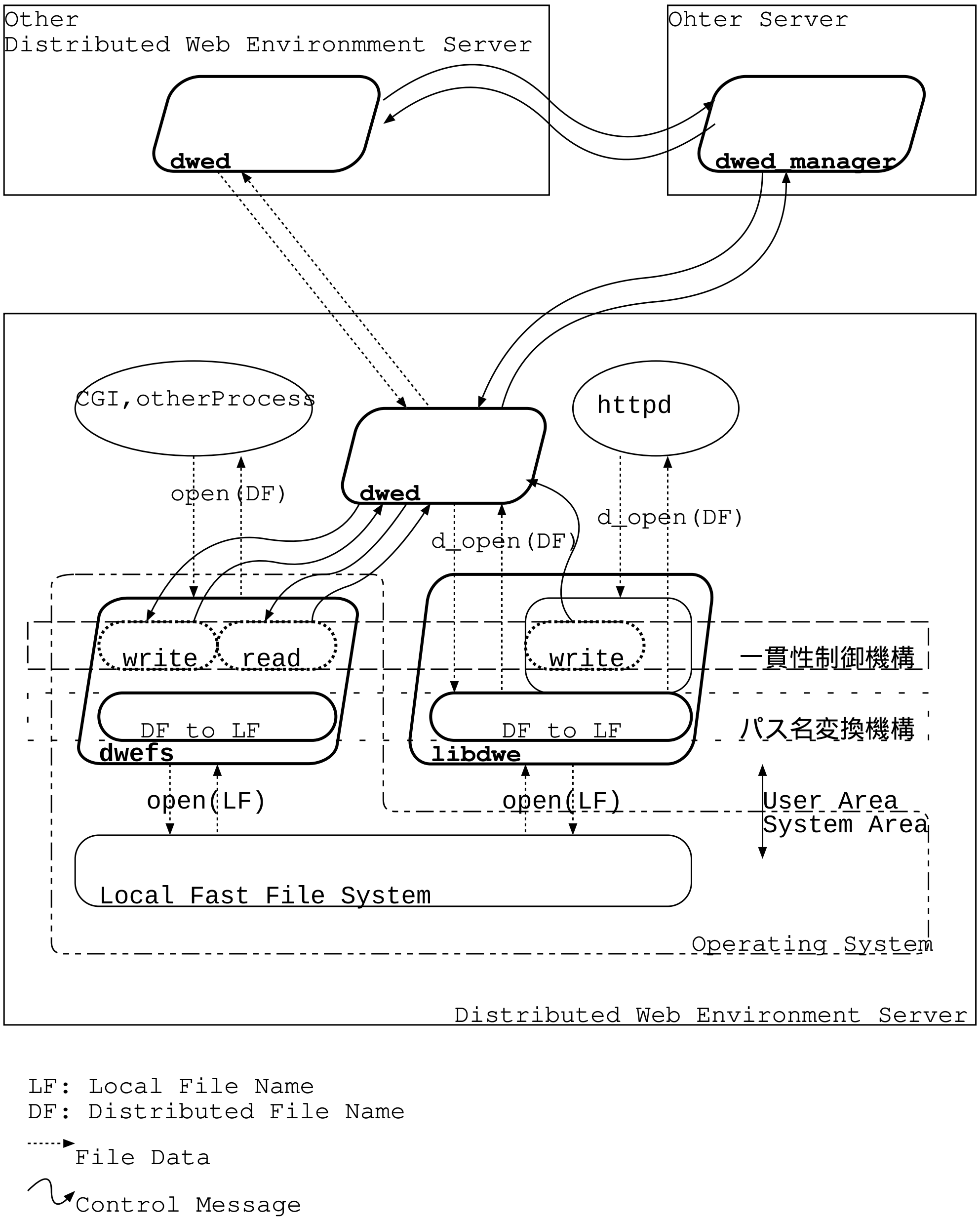

図 3.1: 分散 Web サーバ環境におけるシステム構成

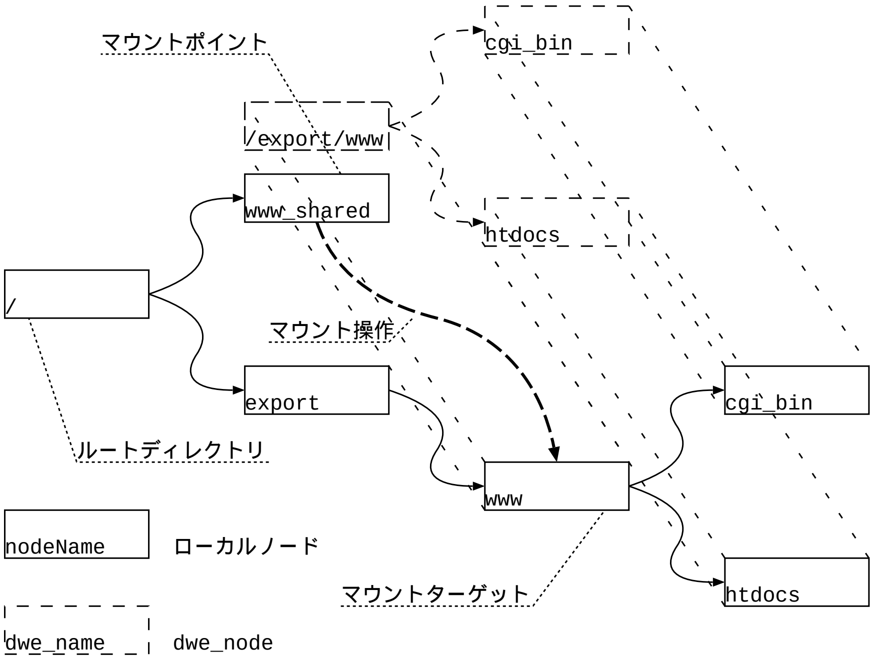

図 3.2: 本システムのマウント操作処理イメージ

分散ファイルシステムとして有効なアクセスは/www_shared ディレクトリ以下である．操作のイメージ図を図 3.2 に示す．

一般のプロセスで/www_shared 以下にアクセスすると，分散ファイルシステム機構を利用することになる．また，ライブラリを利用してコンパイルしたソフトウェアで/www_shared 以下にアクセスすると，ライブラリ機構を利用することになる．

ライブラリを利用したソフトウェアで，一般のファイルシステムにアクセスした時は，特に何もなく普通にコンパイルしたときと同様に動作する．

共有ファイル空間を構築した後も，直接ローカルファイルシステムをアクセスすることは可能とする．しかし，その場合の一貫性は保証しないし，直接変更後の分散 Web サーバ環境の正常動作も保証しない．読み込みしかないことがわかっており一貫性制御の必要もないアプリケーションにはこの直接アクセスが有効である．

第4章　実装

4.1　実装環境

本システムの実装には，ソースコードの公開されたオペレーティングシステムが適している．候補にはNetBSD[29]，FreeBSD[30]，Linuxなど多く挙げられる．その中でも，さまざまなプラットフォームで動作可能であるNetBSDは，測定時のハードウェア環境を構築する場合に便利である．BSDオペレーティングシステムは研究用として発展してきた歴史もあり，研究成果も多い[22]．普段からも生活環境として利用しているオペレーティングシステムでもあるなどの理由から，オペレーティングシステムにはNetBSDを選択することにした．

開発用のマシンには，作業効率を考え高速なコンパイルの可能なi386系PCを選択した．

コンパイラは，NetBSD 1.4に標準で付属しているgcc version egcs-2.91.60 19981201 (egcs-1.1.1 release)[31]を利用した．

表4.1に実装環境をまとめる．

項目	説明
ハードウエア	IBM ThinkPad 560E
CPU	MMX Pentium 166MHz
オペレーティングシステム	NetBSD 1.4
コンパイラ	egcs-1.1.1 release

表4.1: 実装環境

4.2　mount_dweの実装

mount_dweはdwefsをマウントするためのコマンドである．オプションにはdwefsとして共有するディレクトリとマウントポイントを指定する．以下に例を示す．

```
lestat(80)% mount_dwe
usage: mount_dwe [-o options] target_fs mount_point
lestat(81)% su
lestat(82)# mount -t dwe /export/local/etc/httpd/htdocs /dwefs
```

これで/dwefs 以下が dwefs 空間としてアクセス可能となる．

mount_dwe が動作するためには，dwefs の情報をあらかじめカーネルに組み込む必要がある．以下のように dwefs についての情報を mount.h と vnode.h に追加した．

```
/usr/src/sys/sys/mount.h:
#define MOUNT_DWE       "dwe"        /* dwe Filesystem Layer */
#define CTL_VFS_NAMES { ... { MOUNT_DWE, CTLTYPE_NODE }, ... }
#define VFS_MAXID       20+1         /* number of valid vfs ids */

/usr/src/sys/sys/vnode.h:
enum vtagtype   { ... VT_DWE, ... }; /* Vnode tag types */
```

これらの情報からカーネルは mount_dwe コマンドにより，dwefs 内の dwe_mount(), dwe_unmount() が呼び出せるようになる．dwe_mount(), dwe_unmount() については 4.5 節で述べる．

4.3 libdwe の実装

libdwe はライブラリ libdwe.so.1，ライブラリの定義ファイル dwe_lib.h により構成される．ライブラリが dwefs へのバイパスアクセスを実現するために d_realpath() を利用する．相対パスを指定された時，カレントワーキングディレクトリを知る必要がある．システムコール LKM の d___getcwd() によってカレントワーキングディレクトリを得る．分散 Web サーバ環境上で一意なパス名を生成する d_transpath() もこれらを利用して実装している．またシステムコール LKM の dwed_kinfo() はファイル変更時のメッセージ通知を行う．以下でそれぞれについて詳しく説明する．

- **libdwe.so.1**

libdwe.so.1 は libdwe の共有ライブラリである．スタティックライブラリの libdwe.a も提供している．このライブラリを利用することで，もし dwefs 上のファイルをアクセスしようとすると，ローカルディスクへのアクセスへ変換される．提供される関数は，ファイル操作に関わるシステムコールと標準 C ライブラリのすべてであり，合計 54 個[1] 存在する．これらの関数は関数名に d_をつけた名前で提供される．d_つきの関数は既存のファイルシステムにアクセスしようとすると，通常の関数として機能する．

- **dwe_lib.h**

dwe_lib.h は libdwe ライブラリに含まれる関数の定義ファイルである．このファイルに上述した 54 個の関数が定義されており，libdwe を利用する時はこのファイルをインクルードする．

これでは数多く存在するシステムコールと標準 C ライブラリのうち，どの関数がファイル操作関数かを常に意識しなければならない．既存コードを libdwe に対応させることも大変な作業となる．そこで，既存の関数宣言をプリプロセッサによって該当関数を自動的に d_つきの関数に置き換えるコードを追加した．ソースコードの合計行数が 99349 行の apache-1.3.9 において，このファイルをインクルード利用した結果，1 ヶ所の修正を必要とするのみで問題なく動作した．この結果からもわかる通り，よほど特殊なコードを書かない限り，コードの変更はこのファイルのインクルード行を追加するだけで libdwe を利用可能である．

- **d___getcwd() システムコール**

dwefs の vnode 情報を利用し，dwefs 上のパスをローカルファイルの情報にアクセスすることなしにカレントワーキングディレクトリを得るシステムコール LKM である．

dwefs 上のパスに限られるが，パス名や親 vnode へのポインタはパス名ルックアップ時に vnode にキャッシュされているためディスクアクセスすることもなく，高速な処理を提供する．

親 vnode へのポインタが NULL になると，マウントポイントまで辿ったことになる．マウントポイントは，マウントターゲットのパス名までのフルパスである．マウントターゲットのパス名に，これまで辿ったパス名を逆からつなぐことでカレントワーキングディレクトリが得られる．

---

[1] getcwd, open, close, chmod, lchmod, fchmod, dup, dup2, read, readv, pread, preadv, write, writev, pwrite, pwritev, stat, lstat, fstat, chown, lchown, fchown, access, chdir, fchdir, chroot, chflags, fchflags, statfs, fstatfs, truncate, ftruncate, utimes, lutimes, futimes, getfh, link, mkdir, mkfifo, mknod, rename, readlink, symlink, unlink, revoke, rmdir, pathconf, fpathconf, dlopen, opendir, fopen, fclose, creat, remove,

dwefs上を通る相対パスをフルパスに変換する時に，開始ポイントを知るためにd_realpath()から呼ばれる．

- **d_realpath()**

指定したターゲットパス名がdwefs上にあれば，実際に存在するファイルシステムのフルパス名を返す．シンボリックリンクをたどりながら処理するため，ターゲットパス名にシンボリックリンクを指定することはできない．その場合はd_lrealpath()を利用する．分散Webサーバ環境に一意のフルパス名を生成するためにd_transpath()から呼ばれる．

- **d_transpath()**

指定したターゲットパス名がdwefs上にあれば，dwefs上のフルパス名を返す．メッセージ管理はこの名前によって行われる．

- **dwed_kinfo() システムコール**

カーネル空間とユーザ空間でファイル更新情報や，ファイル利用状況などの情報をやりとりするためのシステムコールLKMである．呼び出す条件によって以下の2通りの役割がある．

  - d_dwed_kinforecv()
    バッファを確保し，そのバッファへのポインタとサイズをオプションとして指定する．d_dwed_kinforecv()はカーネル内にたまっているメッセージを返す．
  - d_dwed_kinfosend()
    libdweライブラリによってファイルが変更されると，その変更をdwefsは認識できない．libdweライブラリからd_dwed_kinfosend()によってカーネルに通知することでdwefsはその変更を知る．厳密な一貫性制御機構を利用する時，ファイルをopen()可能かを確認メッセージで確認し，close()したことを確認メッセージで通知する．

- **d_isnotdwed()**

libdweライブラリは次節で述べるdwedからも利用される．しかし，libdweライブラリの標準の動作ではファイルの変更操作はチェックされ，ファイル変更メッセージが通知され，無限ループが発生する．そのため，dwe_isnotdwed変数を用意することでdwedからの利用を区別する．この変数を設定するのがこの関数であり，呼び出すことでlibdweライブラリはメッセージ通知を行わなくなる．2度以上実行しても結果は変わらない．

- **libdwe ライブラリ利用例**

以上が libdwe ライブラリを構成するモジュールである．上述したように libdwe ライブラリはシステムコール LKM を利用するため，先にカーネルにロードしなければならない．以下に手順を示す．

```
lestat(13)% su
lestat(14)# modload mod___getcwd/mod_d___getcwd.o
Module loaded as ID 0
lestat(15)# modload mod_dwed_kinfo/mod_dwed_kinfo.o
Module loaded as ID 1
lestat(16)# modstat
Type     Id  Off Loadaddr Size Info     Rev Module Name
SYSCALL   0 210 f4bce000 0008 f4bcf008   1 sys_d___getcwd
SYSCALL   1 211 f4bd3000 0008 f4bd4008   1 sys_dwed_kinfo
lestat(17)#
```

ファイル名 sample を削除するサンプルプログラム sample_unlink.c を以下に示す．

```
#include <unistd.h>
#include "dwe_lib.h"
int main(void)
{
  unlink("sample_file");
}
```

dwe_lib.h がインクルードされているので，unlink() は d_unlink() に展開される．このコード sample_unlink.c のコンパイル例を以下に示す．

```
lestat(92)% cc -I/usr/src/lib/libdwe -c sample_unlink.c
In file included from sample_unlink.c:2:
/usr/src/lib/libdwe/dwe_lib.h:4: warning: #warning "include dwe_lib.h"
lestat(93)% cc -L/usr/src/lib/libdwe -R/usr/src/lib/libdwe \
                -o sample_unlink sample_unlink.o -ldwe
lestat(94)% ldd sample_unlink
sample_unlink:
        -ldwe.1 => /usr/src/lib/libdwe/libdwe.so.1 (0x4001a000)
        -lc.12 => /usr/lib/libc.so.12.40 (0x40020000)
lestat(95)%
```

4.4 メッセージ管理機構の実装

本節では，メッセージ管理機構として，メッセージの意味，そのメッセージを扱うdwed，dwed_mamagerについて述べる．

- **dwed_msg 構造体**

メッセージはファイル変更操作がdwefs内で起こった時に生成され，カーネル内のメッセージバッファに入れられる．libdweライブラリ内でファイル変更操作が行われた時にも生成され，その時はd_dwed_kinfosend()によりカーネル内のメッセージバッファに入れられる．以下にメッセージの内容を示す．

```
struct dwed_msg {
  int op;
  struct timespec op_ts;
  char path[MAXPATHLEN];
  int len;
  char srcpath[MAXPATHLEN];
  int srclen;
  char ip[DWED_IPSIZE];
};
```

int op の 32 ビットの役割は，確保された他のデータ領域が有効かという情報や，それ自身に割り当てられたバッファを解放できるかなどの情報が含まれている．またファイル操作内容など，情報量の小さな静的情報もここに含まれる．以下に詳しく説明する．

- 利用されるパス名の数 (2 ビット)
  定義名: KDWED_PATH1,KDWED_PATH2
  ファイル操作は 1 つのパス名に対する操作がほとんどであるが，名前変更のように変更前と変更後の 2 つのパス名を必要とする操作がある．このビットを利用して，その操作がパス名をいくつ持つかを区別する．

- メッセージの生成元 IP アドレス情報の有無 (1 ビット)
  定義名: UDWED_IP
  IP アドレスが設定されていればこの値が設定される．ネットワーク通信を行うメッセージには IP アドレスが必要だが，確認メッセージなどネットワーク通信を必要としないものもある．

- メッセージのリファレンスカウント (7 ビット)
  dwed_manager によって利用される．メッセージ用バッファは複数あるサーバにすべて送り終えるまで解放できない．初期値としてサーバの数が指定され，メッセージが転送されるたびに 1 ずつ小さくなる．0 になると，そのメッセージ用に利用されたバッファは解放される．この情報量が 7 ビットで表現されることから，最大接続可能台数は 2^7-1=127 台に制限される．それ以上の値が設定されてもチェック機能がないため正しく動作しない．

- 属性情報の有無 (1 ビット)
  定義名: UDWED_ATTR
  ファイル変更操作によって属性を持つものとそうでないものがある．属性情報を持つものはこのビットが設定されている．

- メッセージの優先度 (1 ビット)
  このビットが設定されていると，メッセージの処理順序が優先される．libdwe ライブラリから生成されるファイル変更操作にはこのビットが設定されない．

- ファイル変更操作名 (11 ビット)
  全内容を表 4.2 に示す．

struct timespec op_ts はファイル変更操作が起こった時間を記録する．

char path は操作対象の dwefs パス名で，この名前が分散 Web 環境の一意の名前として利用される．int len はその長さであり，ネットワーク転送時はバッファサイズではなく実際の長さが転送される．

char srcpath は，KDWED_PATH2 を持つメッセージのみ有効であり，その意味はメッセージにより異なる．srclen はその長さである．

操作名	パス名の数	優先度の有無	属性情報の有無
DWED_CHECK	KDWED_PATH1	DWED_PRIO	
DWED_CREATE	KDWED_PATH1	DWED_PRIO	UDWED_ATTR
DWED_MKDIR	KDWED_PATH1	DWED_PRIO	
DWED_SYMLINK	KDWED_PATH2	DWED_PRIO	UDWED_ATTR
DWED_MKNOD	KDWED_PATH1	DWED_PRIO	UDWED_ATTR
DWED_LINK	KDWED_PATH2	DWED_PRIO	
DWED_REMOVE	KDWED_PATH1	DWED_PRIO	
DWED_RENAME	KDWED_PATH2	DWED_PRIO	
DWED_SETATTR	KDWED_PATH1	DWED_PRIO	UDWED_ATTR
DWED_RMDIR	KDWED_PATH1	DWED_PRIO	
DWED_WRITE	KDWED_PATH1	DWED_PRIO	UDWED_ATTR

表 4.2: ファイル変更操作ごとのパラメータ一覧

char ip はメッセージの生成元 IP アドレスである．このアドレスはネットワークを介した転送が行われるメッセージにのみ記録される．

メッセージバッファは dwed の実行する d_dwed_kinforecv() により定期的にこれらの情報がユーザ空間へ回収される．ユーザ空間に取り出されたメッセージには，さらに以下の属性情報を追加されるものもある．

```
struct udwed_attr {
  mode_t    mode;
  u_int32_t flags;
  uid_t     uid;
  gid_t     gid;
  dev_t     rdev; // for mknod
  struct timeval tv[2]; /* tv[0] = atime, tv[1] = mtime */
};
```

こうして生成されたメッセージは dwed_manager へ送られ，まとめられる．dwed_manager は送られてくるメッセージと引き替えに，他のホストで生成されたメッセージを dwed へ返す．その返されたメッセージによって dwed は他のホストによる変更操作を知ることができ，メッセージに従ってファイル変更操作を行う．

このようにメッセージ管理機構を構成するモジュールは，各ホストに1つずつ動作するdwedと，分散Webサーバ環境にただ1つ動作するdwed_managerの2つにより管理されている．

- **dwedの実装**

dwedは各ホストに1つずつ動作する必要がある．dwedはd___getcwd()システムコールとdwed_kinfo()システムコールを使うので，あらかじめロードしなければならない．またdwed_managerよりも先に実行を開始する．dwedの動作を以下に示す．

1. カーネル空間からメッセージを受信
   d_dwed_kinforecv()を使い，メッセージをユーザ空間に取り出す．
2. 欠けた情報の補完
   属性情報など，カーネル内部で設定されない情報を取得する．また，ネットワーク転送するメッセージのIPアドレスを設定する．
3. dwed_managerへメッセージを送信
   ネットワーク通信量を必要最低限に抑えるため，メッセージサイズの情報を送信してから，メッセージを送信する．
4. dwed_mamagerからメッセージを受信
   他のホストでファイル変更された時に生成されたメッセージを受信する．
5. 受信メッセージの処理
   パス名，操作名などの情報をもとに，libdweライブラリの機能を利用して，他ホストでの変更内容を実行する．

この作業を繰り返す．

- **dwed_managerの実装**

dwed_managerは分散Webサーバ環境に1つだけ動作する必要がある．dwedとは異なり特別なシステムコールを利用しない．先にdwedが動作していることを確認するだけで実行可能である．dwedの動作ホストを記述したファイルを/etc/dwed.confに置く必要がある．記述例を以下に示す．

```
#dwed host ip address
133.27.175.43 # zion
133.27.175.44 # kinka
```

dwed_manager の動作を以下に示す．

1. /etc/dwed.conf を読み込む
   このファイルがないと起動しない．dwed の IP アドレスを取得する．
2. ファイル変更メッセージを dwed から受信
   取得した IP アドレスに自分の存在を知らせ，メッセージを受信する．
3. 受信したメッセージを各 dwed へ送信
   この時，操作時間を確認し，操作順序が逆にならないようにしている．
4. 不要になったメッセージバッファの解放
   メッセージ受信時に dwed の数だけリファレンスカウントを設定しており，値が 0 になれば解放する．メッセージの同時転送数に制限があるため，状況によっては先送りされるメッセージがあるからである．

/etc/dwed.conf は起動時に一度だけ読み込まれる．dwed の動作ホストを増やす場合は，dwed_manager に kill -HUP を実行することで，dwed_manager から操作時間つきの停止命令が dwed へ送られ，dwed は操作時間以降のメッセージを送らずに一旦停止し，確認メッセージを返す．すべての dwed が停止したことを dwed_manager が確認すると/etc/dwed.conf を読み込み，リスタートする．

4.5 dwefs の実装

本節では dwefs の実装について，dwe_node 構造体を構成するデータの意味，ファイル変更操作，dwe_node メモリ管理方法について述べる．

- **dwe_node 構造体**

dwefs を構成する重要な役割を果たす dwe_node 構造体について述べる．dwe_node は，すべてのタイプのファイルが参照されたときに生成されるノードである．dwe_lookup() により生成され，dwe_reclaim() によって解放される．dwe_node 構造体の一部を以下に示す．

```
struct dwe_node {
  struct vnode    *dwe_lowervp; /* VREFed once */
  int             dwe_rc;       /* reference count */
  char            *dwe_name;    /* node name */
  struct dwe_node *dwe_pvnode;  /* parent vnode pointer */
  int             dwe_isdirty;  /* dirty check */
  ...
};
```

struct vnode *dwe_lowervp は，dwefs の下層のファイルシステムにあたる FFS(Fast File System)[16] のノードへのポインタである．この値を使うことで，実際にデータが保存されている FFS にアクセスすることができる．

int dwe_rc は，dwe_node 構造体が他のいくつのノードから参照されているかを示す．この構造体が参照されるたびに 1 ずつ値を大きくし，参照がなくなると値を小さくすることで，自身のメモリを解放しても良いかを判断する整数型変数である．初期値は 1 で，この値が 0 になるとメモリを解放する．

char *dwe_name は，ノードの名前が入っており，realpath() から利用される．

struct dwe_node *dwe_pvnode は，親ディレクトリのノードへのポインタで，realpath() から利用される．

int dwe_isdirty は，ファイルが open() されてから close() されるまでに，そのノードの指すファイルが変更されたかを記録する．初期値は FALSE で，write() の呼び出しによってファイルが変更されると TRUE が設定される．close() が呼ばれた時点でこの値が FALSE なら，このファイルへの変更はなかったことになり，TRUE なら変更されたことになる．

- ファイルタイプ

dwefs によって扱えるファイルのタイプに制限はないが，本実装環境で扱わないファイルタイプは未実装である．

ファイルタイプによってデータブロックを使用するものとディレクトリエントリにだけ存在するものの 2 通りがある．表 4.3 にデータブロックの有無とファイルタイプの関係を示す．

ファイルタイプはファイル変更を反映させるアクションを決定する．データブロックを持たないファイルタイプを持つファイルの反映は 1 つのメッセージ転送のみで完了する．データブロックを持つファイルは，メッセージ受信後データブロックを別に転送する．

vnode type	データブロックの有無	説明
VNON	-	タイプなし
VREG	1 バイト以上の時は使用する	通常ファイル
VDIR	使用する	ディレクトリ
VBLK	使用しない	ブロック型デバイス
VCHR	使用しない	キャラクタ型デバイス
VLNK	使用しない	シンボリックリンク
VFIFO	使用しない	FIFO

表 4.3: ファイルタイプとデータブロックの有無

- ファイル変更を伴う操作

ファイル操作のすべてがファイル変更に関わるとは限らない．UNIX のファイルシステムでは最後にファイルにアクセスした時間を記録する．ファイル変更ごとにメッセージ通信が発生する dwefs でこの時間を管理することは，オーバヘッドが大きいため行わない．以下にファイル変更メッセージの発生する操作を説明する．

- 内容変更
  dwe_write()
  通常ファイルのデータブロックを変更する．ディレクトリへの名前空間に関する変更を伴わない唯一の操作となる．dwe_write() は 1 度の操作で複数回呼ばれるため，dwe_write() が呼ばれると dwe_node の dwe_isdirty を TRUE に設定する．dwe_close() が呼ばれた時，dwe_isdirty を確認し，TRUE ならメッセージを生成する．

- 名前空間変更
  dwe_rename()
  すべてのファイルタイプの名前を変更する．dwe_node が変更されるのはこの操作時のみである．

- 属性変更
  dwe_setattr()
  すべてのファイルタイプのファイル属性や時間を変更する．すべてのファイル変更操作の直後にはファイル変更時間 (mtime) の変更メッセージが必ず発生する．メッセージの半分以上はこの属性変更メッセージであり，このメッセージの集合をいくつかまとめて最適化できれば，システム全体の性能向上につながる．

- 削除

    - **dwe_rmdir()**
      ディレクトリを削除する．
    - **dwe_remove()**
      ディレクトリ以外を削除する．通常ファイルの場合は，データブロックも削除する．

- 新規作成
    - **dwe_create()**
      0バイトの通常ファイルを作成する．
    - **dwe_mkdir()**
      ディレクトリを作成する．データブロックも割り当てる．
    - **dwe_symlink()**
      シンボリックリンクを作成する．
    - **dwe_mknod()**
      スペシャルファイルを作成する．
    - **dwe_link()**
      ハードリンクを作成する．

- **dwe_nodeのメモリ管理**

dwe_nodeはvnodeが生成された時，可能な限り同時に生成される．以下ではvnodeの生成，解放に関わる操作について述べる．

- **dwe_lookup()**
  dwefs上のファイルを触ると，はじめにパス名ルックアップが行われ，vnodeを生成し，同時にdwe_nodeも生成する．ルートノードを除くすべてのファイルのvnodeはこれにより生成される．

- **dwe_reclaim()**
  dwe_lookup()により生成されたvnodeは，プロセスに使われなくなると解放対象となる．vnodeを解放するためにこれが呼ばれ，同時にdwe_nodeを解放しようとする．dwe_nodeのリファレンスカウントを1減らし，もし0になればdwe_nodeを解放するが，そうでなければ別のdwe_nodeに参照されていることになる．つまり，vnodeが解放されてもdwe_nodeが解放されるとは限らない．残されたdwe_nodeは，参照先dwe_nodeがすべて解放された時に解放される．解放に失敗するdwe_nodeの実体はディレクトリのみである．

- **dwe_mount()**
  マウントポイントの dwe_node はマウントされる時に生成する．ルートノードだけは dwe_unmount() 以外に解放されないために，リファレンスカウントの初期値には 2 が設定される．また，ルートノードに限り，dwe_name はパス名であり，文字/が含まれる．

- **dwe_unmount()**
  生成されたすべての dwefs の vnode が解放される．解放に成功した dwe_node は直後に親ノードにあたる dwe_pvnode を解放しようとする．こうして dwe_node は再帰的に解放される．

第5章　評価と考察

5.1　時間測定方法

オペレーティングシステムによって，またハードウェアによって提供される時間測定インタフェースはさまざまである．本節ではさまざまな性能の時間測定に用いるルーチンについて説明する．

- **gettimeofday()**

性能評価を行う場合一般に良く用いられる手法がgettimeofday()ルーチンを利用したものである．gettimeofday()ルーチンはさまざまなハードウェア環境で利用可能であるが，その精度は機種に依存している．また長時間実行する場合は，他のプログラムの時間の影響を受ける．本システムの測定環境となるNetBSD1.4ではマイクロ秒単位までの測定が可能である．しかし，本システムの測定にはナノ秒単位の測定も一部必要となるためNetBSD1.4のgettimeofday()ルーチンの精度ではやや不足している．gettimeofday()ルーチンを利用した測定のためのサンプルソースコードを以下に示す．

```
#include <sys/time.h>
main()
{
  struct timeval /*{
    long    tv_sec;         // seconds since Jan. 1, 1970
    long    tv_usec;        // and microseconds
  }*/ tvs, tve;
  gettimeofday(&tvs, NULL); // 測定開始時間の保存
  performance_routine();    // 測定対象処理
  gettimeofday(&tve, NULL); // 測定終了時間の保存
}
```

- **getrusage()**

長時間の測定を行う場合は，他のプロセスなどの影響を排除したい場合がある．getrusage() ルーチンを用いることによって，指定したプログラムで消費されたユーザ時間とシステム時間を得ることができる．getrusage() ルーチンを利用した測定のためのサンプルソースコードを以下に示す．

```
#include <sys/time.h>
#include <sys/resource.h>
int main()
{
  struct  rusage /*{
    struct timeval ru_utime;   // user time used
    struct timeval ru_stime;   // system time used
    ...
  }*/ rus, rue;
  getrusage(RUSAGE_SELF,&rus); // 測定開始時間の保存
  performance_routine();       // 測定対象処理
  getrusage(RUSAGE_SELF,&rue); // 測定終了時間の保存
}
```

サンプルコードでは RUSAGE_SELF を指定しているが，RUSAGE_CHILDREN を指定することにより，子プロセスの CPU 消費時間を測定することができる．NetBSD の time コマンドではこれが利用されている．time コマンドの使用方法とその情報の意味を以下に示す．

```
lestat(129)% /usr/bin/time -p sleep 1
real         0.99 /* real time */
user         0.00 /* user time used */
sys          0.00 /* system time used */
```

real	そのプロセスを呼び出してから終了するまでの経過実時間 (秒)．
user	そのプロセスが実際に消費したユーザ CPU 時間 (秒)．
sys	そのプロセスが実際に消費したシステム CPU 時間 (秒)．

time コマンドをフルパスで指定し実行しているのは，csh の built-in コマンドとしても提供されているからである．また，-p オプションは出力フォーマットを指定している．sleep コマンドの処理時間は実時間では 1 秒となるが CPU の消費はほとんどないことを示している．

- タイムスタンプカウンタ

最近の高性能プロセッサには SunMicrosystems 社の UltraSparc や Intel 社の Pentium プロセッサなど，タイムスタンプカウンタと呼ばれるサイクルレベルのカウンタを内蔵するものが多い．これは計算機の処理できる最小の単位であり，プロセッサクロックにこれ以上の精度を求めることは難しい．このカウンタは CPU がリセットされてから何ステップ経過したかをサイクル単位でカウントする．

Pentium の RDTSC

Pentium のタイムスタンプカウンタを利用するためには RDTSC(Read Time-Stamp Counter)[7] 命令を使用する．RDTSC 命令はプロセッサ内部にある 64 ビットのタイムスタンプカウンタをユーザモードで読む命令で，システム起動時からの総サイクル数を返す．ここで得られる値は時間ではないことに注意する必要がある．測定したい処理の開始時にこのカウンタを保存し，処理の終了時にも再度このカウンタを保存してその差を取ることで，タイムスタンプカウンタによる処理時間を得ることができる．カウンタと時間と動作周波数の関係は以下のようになる．

$$\text{時間} = \text{サイクル数}/\text{動作周波数}$$

例えば，RDTSC によって 1,000,000 クロックという値が得られたとする．このサイクル数から得られる実際の時間はプロセッサの動作周波数によって異なる．AMD 社のプロセッサである動作周波数 560MHz の Athlon の場合，以下ように求めることができる．

$$1,000,000(cycles)/560,000,000(Hz) \fallingdotseq 1,785,714(\text{ナノ秒})$$

RDTSC 命令はパイプライン実行可能であるため，前後の命令による実行状況によって

得られる値が左右される．パイプライン処理を止めるには，RDTSC 命令の直前にパイプライン処理を止める命令を挿入する必要がある．しかし，この命令に該当する命令自体にも数十クロック消費するため，細かい測定を行う場合はその影響を考慮しなければならない．表 5.1 に Intel PentiumII333MHz と AMD Athlon560MHz による RDTSC 命令のみを実行した場合，およびパイプライン処理を止める命令である CPUID 命令を追加した場合にかかるサイクル数と時間を示す．

プロセッサ名 (動作周波数)	RDTSC	RDTSC+CPUID	CPUID
Intel PentiumII (300MHz)	44(146nsec)	136(453nsec)	92(306nsec)
Intel PentiumII (333MHz)	45(135nsec)	152(457nsec)	107(322nsec)
AMD Athlon (560MHz)	28(49nsec)	73(130nsec)	45(80nsec)

表 5.1: RDTSC に CPUID を併用した場合の処理時間の変化

本章で示すデータはすべてパイプライン処理について考慮し，その影響の出ないような測定を行った．Pentium プロセッサのタイムスタンプカウンタを利用した測定のためのパイプライン処理を考慮しない場合のサンプルソースコードを以下に示す．

```
#define RDTSC(x) __asm __volatile(".byte 0x0f,0x31" : "=A" (x))
int main()
{
  unsigned long long start_count, current_count;
  RDTSC(start_count);     // 測定開始時のタイムスタンプカウンタの保存
  performance_routine(); // 測定対象処理
  RDTSC(current_count);  // 測定終了時のタイムスタンプカウンタの保存
}
```

5.2 測定環境

本節では実際に測定を行う環境についてハードウェア的な環境，ソフトウェア的な環境に分けて説明する．

ホスト名	役割	CPU	メモリ	OS	NIC
lestat	情報統括サーバ	PentiumII(333MHz)	192MB	NetBSD1.4	100BaseTX
kinka	Web サーバ	PentiumII(300MHz)	64MB	NetBSD1.4	100BaseTX
zion	Web サーバ	PentiumII(300MHz)	64MB	NetBSD1.4	100BaseTX

表 5.2: 測定環境

5.2.1 ハードウェア環境

本測定を行った測定環境を表5.2 に示す．Web クライアントについてはWeb サーバの役割を果たすすべての計算機をあわせた性能を得る必要がある．その他に測定に関わるハードウェア構成上の制約は特にない．

5.2.2 ソフトウェア環境

以下のファイルシステムを測定対象とする．

- FFS(Fast File System)
- NFS(Network File System)
- AFS(Andrew File System)
- dwefs(Distributed Web Environment File System)
- libdwe(Distributed Web Environment Support Library)

NetBSD1.4 において FFS，NFS は標準でサポートされている．Transarc の AFS は商用であるだけでなく，現時点ではNetBSD への対応は行われていない．そのため，フリーの AFS クライアントとして The Arla Project[26] による AFS クライアントの実装である arla-0.26 を使用する．また，AFS のセキュリティ機構を提供する Kerberos4 パッケージは KTH 版 Kerberos4[27] を使用する．

5.3 定量的評価

本節では本システムをさまざまな側面から測定する．その測定結果をもとに関連研究と比較し定量的に評価する．

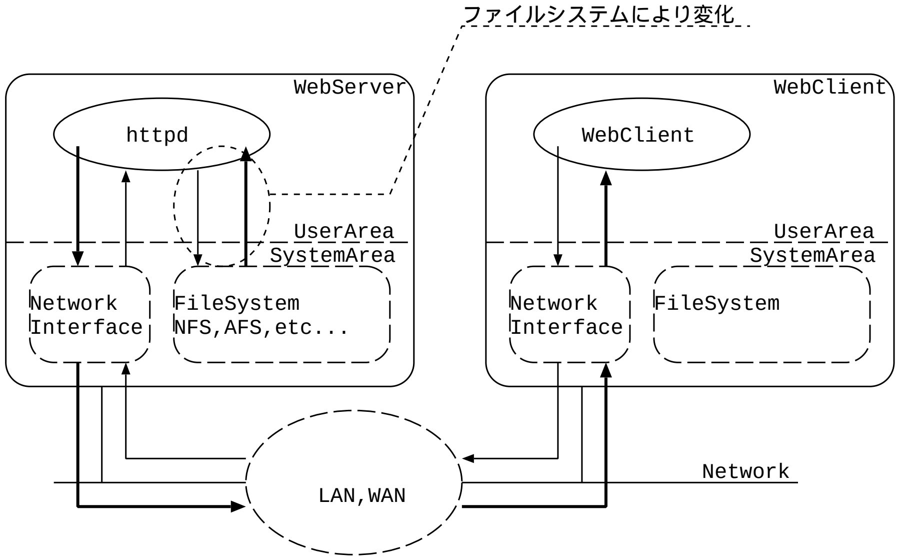

図 5.1: Web サーバの処理

5.3.1 評価想定環境

本システムの目的は分散 Web サーバ環境において既存システムより同一ハードウェア環境において，環境に応じた高速な処理を実現することである．本システムを利用するための想定環境は，分散 Web サーバ環境である．その中でも特にファイル入出力を測定する．ファイル入出力の測定により，さまざまな分散 Web サーバ環境の性能比較を行うことが可能である．これはファイルシステムのアクセス時間以外の条件は全く同じことによる．図 5.1 にイメージ図を示す．ファイルシステムの変更により大きく変化するのは，httpd からファイルシステムへのアクセス時間である．よって，測定は以下のような環境について行う．

- ローカルからのファイルシステムの測定
  FFS, libdwe, dwefs, AFS, NFS を利用した場合の読み込み，書き込み時間をキャッシュ効果などをふまえて測定する

5.3.2 ファイルシステムの測定

各環境ごとに以下の項目において比較測定が必要となる．

- ファイル読み込み時間の比較
- ファイル書き込み時間の比較

サイズ	FFS	libdwe	NFS	AFS	dwefs
0.1K	871	5,896	7,755	11,846	31,396
10K	969	5,945	9,893	22,684	31,405
1000K	70,064	124,711	379,437	1,116,262	143,121

表 5.3: 各ファイルシステムの書き込み時間(単位=マイクロ秒)

サイズ	FFS	libdwe	NFS	AFS(hit)	AFS(miss)	dwefs
0.1K	192	289	1,262	3,222	7,266	10,583
10K	648	745	3,066	3,906	10,296	53,990
1000K	109,916	110,064	169,219	512,605	512,887	120,144

表 5.4: 各ファイルシステムの読み込み時間(単位=マイクロ秒)

ファイルシステムにはさまざまなキャッシュシステムが組み込まれている．各ファイルシステムの性能を測定することが目的のため，これらの影響を受けない，あるいは平等な効果が得られる環境を作り測定を行うことにする．キャッシュヒット時の測定には直前にターゲットファイルの読み込みを行ってから測定を行った．また，キャッシュミス時の測定には読み込んだことのないファイルのみ測定を行った．また，vnodeのバッファキャッシュ効果があらわれないようにするため，測定に関係のあるファイルシステムは各測定ごとに直前までマウントするのを避けた．ローカルディスクへのキャッシュを利用するシステムのキャッシュ領域に関しても同様な操作を行った．

AFSにおける読み込みはキャッシュヒット時とキャッシュミス時の2通りを測定しなければならない．キャッシュヒット時とはローカルディスクにファイルが保存されている状態のことで，AFSはファイル読み込み要求をローカルキャッシュを利用して処理する．キャッシュ最大サイズはすべての測定1回分のファイルがキャッシュされる容量にしなければならない．本測定では1000キロバイトのファイルを200回測定する場合が最大であるため，最低200メガバイト必要となる．本測定では余裕を持たせて240メガバイトとした．

本システムにおけるファイル読み込み操作にはWebサーバからとその他のプロセスからの2通りを測定する必要がある．本システムを利用した場合のWebサーバからのアクセスにはlibdweを利用し，一般プロセスからのアクセスはdwefsを利用する．

まず，すべての測定結果を示し，その結果を元に本システムを評価する．

これらの性能測定にはPentiumプロセッサに内蔵されているタイムスタンプカウンタを利用した．全項目において，200回の測定のあとそのデータを並べ替え，中間の100回分の平均値を出した．測定結果を元に各ファイルシステムの比較評価を行う．

- **FFS**

本システムは，FFS がどのファイルシステムよりも高速にアクセス可能であることを前提としている．本測定結果によってもその前提が正しいことが証明されている．

- **NFS**

NFS はすべてのファイル処理操作をネットワークを介して行うため，ネットワーク処理にかかる時間の参考になる．本測定において NFS の処理時間はすべてにおいて FFS, libdwe よりも低速な処理結果となっている．これはローカルファイルの処理時間の方がネットワークファイルの処理時間よりも高速であることをあらわしている．また，AFS,dwefs よりは高速な処理結果が出ている．これは本測定の負荷が比較的小さいからであり，NFS の性能は高負荷になると，極端に低下することを考慮する必要がある．NFS と AFS の性能を同じハードウェア構成を使って比較すると，一つのサーバにおいて，NFS は低負荷 (15 クライアント以下) では高速だが，高負荷になると急速に悪化する [11]．

- **AFS**

AFS の書き込み時は NFS と同程度かそれ以上の時間がかかる．しかし，読み込み時はキャッシュヒット時とキャッシュミス時とで大きく差が出る．サイズの比較的小さなファイルの場合は，キャッシュ効果があらわれ，キャッシュミス時の半分以下の時間になっている．キャッシュヒット時はネットワークを介した処理が発生しないことから，NFS を越えるパフォーマンスが出ることも期待できたが，結果は NFS の処理時間が上回った．上述したように測定内容が高負荷なものであれば，結果は変化する．キャッシュミス時は dwefs を除くと，予想通り最低の結果である．

サイズの大きなファイルの読み込み処理にキャッシュ効果があらわれないのは，AFS が大きなファイルをキャッシュしないためである．

- **libdwe**

libdwe の読み込み処理時間，FFS の読み込み処理時間，dwefs 上のパス名から FFS 上のパス名への変換処理時間の関係を以下に示す．

$libdwe$ の処理時間 $=$ FFS の処理時間 $+$ パス名変換処理時間

1つのファイルごとにある一定のバイパス処理時間が加えられることから，小さなファイルを処理する場合はFFSと大きな差としてあらわれるが，ファイルサイズが大きくなるにしたがって，その影響は小さくなる．

本測定結果において，libdweの読み込み時のパス名変換処理時間は約100マイクロ秒程度かかっていることがわかる．

本システムにおけるlibdweの役割は他の分散ファイルシステムよりも高速な読み込み処理を実現することである．次に分散ファイルシステムとlibdweを比較評価する．

ネットワークを介した処理を行うNFSと比較する．読み込み時はNFSも高速な結果となっているがlibdweはさらに高速である．高負荷時になると，この差はさらに大きくなると予想される．

AFSのキャッシュヒット時とlibdweとの比較は，キャッシュシステムのローカルファイル処理時間とlibdweのパス名変換処理時間とを比較することになる．キャッシュシステムのファイル復元のためのオーバヘッドがlibdweのパス名変換のためのオーバヘッドより十分大きいことが本システムの前提である．結果はAFSのキャッシュシステムよりlibdweのパス名変換処理時間は圧倒的に小さいものとなり，前提が正しいことを証明できた．

次に，libdweの書き込み処理時間について考察する．

測定結果からもわかるようにFFSのみの処理時間との差が読み込み時よりも書き込み時の方がかなり大きくなっていることがわかる．読み込み時と書き込み時のパス名変換処理時間に関わる大きな違いとして，ファイル変更通知処理が発生していることが挙げられる．この処理自体はシステムコールを1回呼び出すだけなので大きくはないが，そのメッセージにより遅いdwefsが動作しはじめるためである．dwefsの動作を止めることが可能なら読み込み時と書き込み時にこれほどの違いは出ない．

- **dwefs**

分散ファイルシステムの中でも遅い部類に入る結果になっている．これは，一貫性を提供する処理やパス名変換処理などの影響である．しかし，dwefsに求められるのは処理速度よりもすべてのアプリケーションへファイルの一貫性を提供することであり，他のシステムより極端に処理速度が遅くなければ，目的は達成される．

まず読み込み時について比較する．AFSのキャッシュヒット時と比較すると，サイズが10キロバイトのファイルで最大で50ミリ秒程度の差になるが，サイズの大きなファイルになれば逆にAFSより高速になるという結果が出ている．このことからも，小さなファイルを数多く操作する場合の速度低下は，比較的大きなものとなるが，それを除けば十分実用に耐えうる性能を得られることが予想される数値である．

また，本システムは他のシステムと違いまだプロトタイプ版であり，デバッグコードやメモリ管理コードの最適化，パス名変換コードの最適化など，性能向上の可能性を十分に残していることも考慮したい．

評価項目	dwefs	AFS	NFS	rdist	ミラー
任意のディスクサイズで利用可	○	−	−	○	×
どこでも書き換え可	○	○	○	○	×
CGI プログラムによる書き換え可	○	△	○	×	×
変更は即反映	○	○	○	×	○
ネットワーク透過性	○	○	○	×	×
規模拡張性	○	○	×	×	×

表 5.5: 他の分散 Web サーバ環境との比較評価

これらの結果をまとめると，AFS のようなローカルディスクへのキャッシュを利用する方式は，キャッシュミスした時はネットワークを介した処理となり，明らかに低速である．また，キャッシュヒットした時でもファイルシステムへの直接アクセスを行う libdwe と比較するとオーバヘッドは大きくなる．libdwe を利用した場合は，比較的低負荷であるパス名変換のオーバヘッドのみでありこれはキャッシュシステムを利用するより高速であることが予想される．またファイルのサイズが大きくなればなるほど FFS に近い性能となり，ファイル単位でキャッシュしないシステムとの差はより大きくなる．

5.4 定性的評価

本節では分散 Web サーバ環境としての本システムおよび類似システムを定性的に評価する．

AFS，NFS 共に機能の面においては本システムとほぼ同様であるが，サーバを別に持つことが相違点として挙げられる．

特に AFS は分散 Web サーバ環境における性能を除いては良くできた分散ファイルシステムである．しかし分散 Web サーバ環境に限れば性能において本システムが勝っている．

NFS は規模拡張性に問題があり，Web サーバの分散を考えなければならない環境では高負荷が予想される．分散 Web 環境に利用するファイルシステムとして NFS よりも AFS が適しており，AFS よりも本システムが適している．

rdist を用いたミラーリングやミラーでは分散 Web サーバ環境として運用していくには読み取り専用などの制限をつけなければ運用不可能である．

分散 Web サーバ環境としての本システムと類似システムとの定性的比較評価を表 5.5 に示す．

5.5　評価と考察のまとめ

本章では，本システムのさまざまな面においての測定を行い，その測定結果を元に評価を行った．測定結果はおおよそ予測していた値に近い値が得られた．

次章では，結論として本論文をまとめる．

第6章　結論

本研究では，分散Webサーバ環境を支援することを目的とし，Webサーバ上に存在するすべてのプロセスをWebサーバプロセスとその他のプロセスに分け，異なったサービスを提供することによって，高性能な分散Webサーバ環境を構築する手法を提案した．

既存の分散ファイルシステムでは，すべてのプロセスに対して一様なファイルサービスを提供している．それに対し本システムでは，処理機会の多いWebサーバプロセスをその他のプロセスと分けることで，分散Webサーバ環境の高性能化を実現した．具体的にはWebサーバ以外のプロセスに対しては共有ファイルの管理機能を提供するために，分散ファイルシステムdwefsを実装した．また，Webサーバに対して高速なアクセスを提供するために，分散ファイルシステムのバイパス機能を提供するライブラリlibdweを実装した．そして，これらの機構を協調動作させるためのメッセージ管理機構を実装した．

これらのモジュールを用いることで，既存の分散ファイルシステムを利用した分散Webサーバ環境よりも高性能化することが可能であることを示した．libdwe対応化したapacheを用いることで，分散Webサーバ環境として運用可能であることがわかった．

本システムはまだ試作段階であり，より最適化可能な段階だが，既存の分散ファイルシステムの例として挙げたAFSを用いたシステムよりもはるかに高性能な結果を示した．さらなる最適化は今後の課題である．本論文で提案したシステムは分散Webサーバ環境において，既存システムよりも優れた性能を得ることを示した．

謝辞

本研究の指導教員として，多くの助言と御指導を賜わった慶應義塾大学環境情報学部 萩野達也助教授に深く感謝致します．また，本研究をはじめる機会を与えて下さった慶應義塾大学常任理事 徳田英幸教授に深く感謝致します．

京都大学総合人間学部 立木秀樹助教授にはオペレーティングシステムの研究をはじめるきっかけを与えて頂きました．萩野研究室の博士 斎藤鉄也氏，安田絹子氏には本研究を行うにあたり多くの有益な御意見，貴重な助言を数多く頂きました．共に深く感謝致します．また，度々助言を下さった遠山緑生氏をはじめとする萩野研究室の方々，共に研究を行った由良淳一氏，岩井将行氏をはじめとする徳田研究室の方々に感謝します．

最後に，本研究に限らず常日頃よりお世話になった，中澤仁氏に深く感謝致します．

2000年1月17日

坪井 孝弘
tsuboi@tom.sfc.keio.ac.jp

付録A　AFS環境の構築

- **AFS 環境の構築**

AFS 環境を構築するために利用したソフトウェアは以下の2つである．

- arla-0.26.tar.gz
  AFS クライアントパッケージ
- krb4-0.10.1.tar.gz
  KTH-Kerberos4 パッケージ

krb4-0.10.1 のインストール方法を以下に示す．もし，autoconf, gm4 が無い場合は，先にインストールする必要がある．この作業により，/usr/athena ディレクトリにプログラムやライブラリがインストールされる．

```
lestat(574)% gunzip < krb4-0.10.1.tar.gz | tar xfp -
lestat(575)% cd krb4-0.10.1
lestat(576)% sh configure --enable-shared
lestat(577)% make
lestat(578)% su
lestat(579)# make install
```

現時点の arla パッケージは Kerberos4 のライブラリのみを必要とするため，これ以上の設定は AFS のみを利用する場合，必要ない．共有ライブラリオプション付きでコンパイル環境を構築しているため，コンパイル時の環境によっては共有ライブラリの設定が必要となる．その一例を以下に示す．

```
lestat(600)% su
lestat(601)# echo /usr/athena/lib >> /etc/ld.so.conf
lestat(602)# ldconfig
```

arla-0.26のインストール方法を以下に示す．

```
lestat(610)% gunzip < arla-0.26.tar.gz | tar xfp -
lestat(611)% cd arla-0.26
lestat(612)% sh configure \
                --x-includes=/usr/X11R6/include \
                --x-libraries=/usr/X11R6/lib \
                --with-krb4=/usr/athena \
                --with-krb4-include=/usr/athena/include \
                --with-krb4-lib=/usr/athena/lib
lestat(613)% make
lestat(614)% su
lestat(615)# make install
lestat(616)# cat conf/services >> /etc/services
lestat(617)# echo coe.keio.ac.jp > /usr/arla/etc/ThisCell
```

本測定に利用するCellであるCOE.KEIO.AC.JPの情報がデフォルトのCellServDBにはないので追加する必要がある．/usr/arla/etc/CellServDBファイルに追加した内容を以下に示す．

```
>coe.keio.ac.jp       # Keio University, Center Of Excellence
133.27.4.247          # coeafs1.sfc.keio.ac.jp
>mag.keio.ac.jp       #Graduate School of Media and Governance
133.27.193.12         #afs2.mag.keio.ac.jp
133.27.193.13         #afs3.mag.keio.ac.jp
>csrg.sfc.keio.ac.jp  # MKng Project
133.27.186.10         # kodama.mkg.sfc.keio.ac.jp
```

AFSクライアントプログラムであるarladの起動方法を以下に示す．

```
lestat(620)% su
lestat(621)# /usr/arla/etc/startarla
```

これで/afs ディレクトリをアクセスすることで AFS を利用可能となった．以上で AFS クライアント環境の構築は終了である．

参考文献

[1] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network Filesystem", Proc. of the 1985 Usenix Summer Conference, June 1985, pp. 119-130.

[2] Sun Microsystems, Inc., "NFS: Network File System Protocol Specification", RFC 1094 (1989).

[3] Sun Microsystems, Inc., "RPC: Remote Procedure Call Protocol Specification Version 2", RFC 1831 (1995).

[4] Sun Microsystems, Inc., "XDR: External Data Representation Standard", RFC 1832 (1995).

[5] Brian Pawlowski, Chet Juszczak, Peter Staubach, Carl Smith, Diane Lebel and David Hitz, "NFS Version 3: Design and Implementation", in Proc. Summer 1994 USENIX Conf., pp. 137-151.

[6] Sun Microsystems, Inc., "NFS Version 3 Protocol Specification", RFC 1813 (1995).

[7] Copyright (c) Intel Corporation 1997, "Using the RDTSC Instruction for Performance Monitoring",
http://www.intel.co.jp/drg/pentiumII/appnotes/RDTSCPM1.HTM

[8] Morris, J.H., Satyanarayanan, M., Conner, M.H., Howard, J.H., Rosenthal, D.S.H., Smith, F.D. "ANDREW: A Distributed Personal Computing Environment", Mar. 1986, Communications of the ACM, Vol. 29, No. 3.

[9] Howard, J.H, "An Overview of the Andrew File System", Proceedings of the USENIX Winter Technical Conference Feb. 1988.

[10] Satyanarayanan, M., Howard, J.H., Nichols, D.A., Spector, A.Z., West, M.J, "The ITC Distributed File System: Principles and Design", Proceedings of the Tenth ACM Symposium on Operating System Principles Dec. 1985, Vol. 19, No. 5.

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

[12] Schill, Alexander, “DCE - The OSF Distributed Computing Environment Client/Server Model and Beyond”, 283. International DCE Workshop. Karlsruhe, Germany, October 7-8, 1993.

[13] Michael L. Kazar, Bruce W. Leverett, Owen T. Anderson, Vasilis Apostolides, Beth A. Bottos, Sailesh Chutani, Craig F. Everhart, W. Anthony Mason, Shu-Tsui Tu, and Edward R. Zayas, “DEcorum File System Architectural Overview”, Proc. 1990 Summer USENIX Conf., pp. 151-163 (1990).

[14] Sailesh Chutani, Owen T. Anderson, Michael L. Kazar, Bruce W. Leverett, W. Anthony Mason, and Robert N. Sidebotham, “The Episode File System”, Proc. 1992 Winter USENIX Conf., pp. 43-60 (1992).

[15] Satyanarayanan, “A Study of File Sizes and Functional Lifetimes”, Proc. Eighth Symp. on Operating Systems Principles, ACM, pp.96-108, 1981.

[16] Marshall K. McKusick, William N. Joy, Samuel J. Leffler, and Robert S. Fabry, “A Fast File System for UNIX”, ACM TOCS, Vol. 2(3), pp. 181-197 (1985).

[17] Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John K. Ousterhout, “Measurements of a Distributed File System”, in Proc. 13th SOSP (1991).

[18] Mendel Rosenblum and John K. Ousterhout, “The Design and Implementation of a Log-Structured File System”, in Proc. 13th SOSP (1991).

[19] M. Seltzer, Bostic, K., McKusick., M., Staelin, C. “A Log-Structured File System for UNIX”, in Proceedings of the Winter Usenix Conference (1993).

[20] Johannes Helander, “Unix under Mach The LITES Server”, Master’s thesis at Helsinki University,(1994).

[21] M. Satyanarayanan, “A Survey of Distributed File Systems”, Research Report CMU-CS-89-116 (1989).

[22] M.K.McKusick, M.J.Karels, J.S.Quarterman, “The Design and Implementation of the 4.4BSD UNIX Operating System”

[23] Andrew S. Tanenbaum, “Distributed Operating Systems”

[24] George Coulouris, Jean Dollimore, Tim Kindberg, “Distributed Systems”

[25] Uresh Vahalia, “UNIX Internals”

[26] The Arla Project: http://www.stacken.kth.se/projekt/arla/

[27] Kungliga Tekniska H skolan: http://www.pdc.kth.se/kth-krb/

[28] PCI Special Interest Group: http://www.pcisig.com/

[29] the NetBSD Project: http://www.netbsd.org/

[30] The FreeBSD Project: http://www.freebsd.org/

[31] the GCC home page: http://egcs.cygnus.com/

[32] Carnegie Mellon University: http://www.cs.cmu.edu/

[33] Sun Microsystems: http://www.sun.com/

[34] IBM: http://www.ibm.com/

[35] Intel: http://www.intel.com/

[36] Transarc: http://www.transarc.com/