修 士 論 文

移動計算機環境に適したファイルシステムの設計と実装

指導教官　中島達夫 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

田中 道信

1997 年 2 月 14 日

要旨

本論文では，移動計算機環境に適したファイルシステムの設計と実装について述べられている．移動計算機環境では，バッテリの消耗や使用可能なデバイスなど従来の固定計算機環境に比べて劇的な環境の変化にさらされている．このような環境下ではサービスを行うシステム自身も環境の変化に応じて資源管理ポリシの変更やサービスモジュールの切替えが必要となる．また，バッテリ稼働である移動中ではバッテリの消費をできる限り少なくすることは大変重要である．

固定計算機環境ではファイルシステムはパフォーマンス (throughput and latency) と信頼性 (reliability) だけが主に重視されてきた．しかし，移動計算機環境ではバッテリ稼働の場合更に消費電力の節約 (power saving) を考慮する必要がある．そこで，本論文では特に供給電源の状況に応じて資源管理ポリシの変更やモジュールの切替えを行うファイルシステムの設計・実装を行った．

移動計算機環境では特にバッテリ消費の節約が重要となるのだが，ファイルシステムにおいて節電を行うにはディスクの回転時間を短くすることがもっとも効率が良いとされている．そこで，本ファイルシステムはアプリケーションが要求するディスクへのアクセスを管理することでディスクの回転時間を短くするのだが，そのためにはアプリケーションが行うファイルの読み書きを制御する以外に，アプリケーションが使用する仮想記憶の管理も行う必要がある．

本ファイルシステムは上記の要件を満たすために，オブジェクトファイリングを元にしたライブラリとして実装を行った．このライブラリは RT-Mach 上で動作するアプリケーション用に開発が行われ，アプリケーションはコンパイル時に本ライブラリをリンクすることで実行時の環境に応じた振舞を行うことが可能となる．

目 次

第 1 章

はじめに

1.1 移動計算機環境

移動計算 (Mobile Computing) を行うためには，PDA やノート型 PC など携帯型の計算機が利用されるが，携帯型計算機はその特質からハードウェア的に従来の添え置き型計算機に比べ CPU の処理性能，物理メモリやディスクの容量，ディスプレイの大きさなどさまざまな面で劣る．そのため，そこで実行されるアプリケーションの数や種類は自然と制限され，移動計算を行うユーザは状況に応じて利用するアプリケーションを使い分ける事になる．また，最近のアプリケーションは従来からある科学計算やシミュレーションだけではなく，HTTP を使った WWW のブラウザやデータベース，ワープロ，表計算，電子メール，ゲームなど各種多彩である．しかもこれらのアプリケーションはそれぞれオペレーティングシステムに対する要求が異なるため，資源の乏しい移動計算ではよりアプリケーションに特化したサービスが重要となってくる．

また，PCMCIA による PC カードを利用すると計算機の実行中に装着された PC カードを取り換える事が可能であり，資源の乏しい移動計算機ではユーザは使用するアプリケーションや状況に応じて扱うデバイスを切替えることが予想される．例えば，移動中では無線 LAN カードによるネットワークへのアクセスを行っていたところ，室内においてはイーサネットカードによるネットワークへのアクセスに切替えられる．

さらに，移動計算ではバッテリ電源による稼働であるため電力消費の節約は非常に重要であると考えられる．このように，移動計算機環境では従来の固定型の計算機に比べ環境の変化が激しく，移動計算機上でサービスを行うシステムは環境の変化に応じた振舞が要

求される．そのためには，状況に応じて資源管理ポリシの変更や，サービスモジュールの動的な切替えを行う必要がある．

1.2 目的

固定計算機環境ではファイルシステムはデータの読み込みや書き込みの性能・パフォーマンス (throughput and latency) やデータの安全性・信頼性 (reliability) に重点がおかれていた．しかし，移動計算機環境ではバッテリ稼働という状況が存在するため，パフォーマンスや信頼性に加えてバッテリ消費の節約 (power saving) も重要となってくる．移動計算中は，バッテリ稼働であるため消費する電力が少ないほど実行可能な時間を長くすることができる．ところが，室内などで固定して使用する場合には従来の固定計算機環境と同じく AC 電源による電力供給が可能となり，消費電力を考慮する必要は無い．そこで移動計算機環境に適したファイルシステムでは供給電源の状態に応じてパフォーマンス・信頼性・バッテリ消費の節約の 3 つの点についてトレードオフ (優先順位) を考え，資源管理ポリシの変更やサービスモジュールの切替えを行う必要がある．

移動中ではバッテリ消費の節約がもっとも重要となる．バッテリ消費の節約にはファイルシステムを構成する要素の中でもっとも電力消費が多いと考えられているディスクへのアクセスを減らすことが考えられる．特に，ディスクは回転している時と回転していない時ではその消費電力にかなりの差がある事からディスクの回転を長く止めていることがもっともバッテリ消費の節約に効果的であると考える [8]．しかし，ディスクの回転を頻繁に止めるとファイルデータに対するアクセス速度などの性能が悪くなったり，あるいは変更データの書き込みが長くなるとその分信頼性が保てないという問題が発生する．そのため，一概に移動中のバッテリ稼働の間は消費電力の節約のためにディスクの回転を止めたままにすることはできない．

また，PDA やノート型 PC といった移動計算機は個人的な用途に限られており，実行されるアプリケーションも従来の科学計算やシミュレーションに比べ特殊なものが多い．これらのアプリケーションでは，従来からある UFS(UNIX File System) のようなファイルシステムが提供する一般的なサービスでは効率が悪く，アプリケーションに特化したサービスが必要であるとされている．そこで，本論文で述べる移動計算機環境に適したファイルシステムでは個人的用途に適したファイルシステムとしてオブジェクトファイリングを用いることにした．オブジェクトファイリングは，OODBMS で基本技術として用

いられており，データベースや表計算では有用な技術であると考えられる．

本論文の目的は，移動計算機における環境の変化，バッテリの電力残量や利用可能なデバイスを考慮したサービスを提供するファイルシステムを設計・実装することである．具体的には，主にバッテリ消費の節約を目的として，状況に応じた資源管理ポリシの変更や，サービスモジュールの切替えを行う．また，本論文で述べるファイルシステムはオブジェクトファイリングを基板として RT-Mach 上で実行されるアプリケーション用のライブラリとして実装される．本ライブラリをリンクしたアプリケーションは実行時の環境に応じて適切な振舞を行うことが可能となる．

以下に続く本論文の構成は，まず第 2 章において移動計算機環境に適したファイルシステムを作るための問題点などを上げ，本ファイルシステムの設計について述べる．そして第 3 章では本ファイルシステムの実装について述べ，第 4 章では実装を行った本ファイルシステムの評価実験，及びその考察について述べる．そして第 5 章では関連研究について述べ，第 6 章で結論と今後の課題を述べ本論文のまとめとする．

第 2 章

設計

この章では，移動計算機環境に適したファイルシステムを作るための設計について述べる．まずはじめに，設計要点について述べ，次に異なる環境に適応するための枠組として用いるオブジェクトグラフについて述べる．

2.1 設計要点

移動計算機環境においては，パフォーマンス・信頼性・バッテリ消費の節約の 3 つを重視しなければならない．しかし，信頼性を上げるために変更データを頻繁に書き出したりすればパフォーマンスの低下につながったり，またバッテリ消費の節約のため長い間ディスクの回転を止めていたりするとパフォーマンスや信頼性が下がってしまう．そのため，状況に応じてこの 3 つの点の優先順位を決め，それぞれ異なるポリシで管理する必要がある．

移動計算機環境では，特に移動中はバッテリの消費を少なくすることが重要となる．最近の携帯型計算機には APM(Advanced Power Management) と呼ばれる機能が提供されており，APM から取得される情報によって供給電源 (バッテリ) の現在の状態がわかる．そこでこの情報を元に供給電源の状態を 3 つに分類し，それぞれの状態において異なる管理ポリシを定義する．以下に各状態とその時の管理ポリシを示す．

- AC 電源から電力供給される **AC-online** という状態．この時には従来の固定型計算機と同様にパフォーマンスを重視したサービスを提供する．また，この時にはバッテリ消費を考慮にいれる必要がないので，消費電力の節約は行われない．

- バッテリ稼働でありバッテリの電力残量が充分ある **battery high** という状態．この時にはバッテリ消費の節約がもっとも重要視される．また電力消費が多いと考えられるディスクへのアクセスは極力少なくされ，ディスクの回転を止めている時間が長くなるようなサービスが行われる．
- バッテリ稼働でありバッテリの電力残量が残り少ない **battery low** という状態．この時には，ファイルサービスにおけるデータの安全性 (信頼性) が最も重視される．変更データはできる限り早くディスクなど 2 次記憶装置に書かれるようなサービスが行われる．

以上のような各状態においてそれぞれ別の管理ポリシを用いてファイルサービスを行うことで，移動計算機環境に適したファイルシステムを提供することができる．

2.2 オブジェクトグラフ

供給電源の状況に応じて資源管理ポリシの変更を行うために，本ファイルシステムでは動的なシステムの変更を可能とする枠組であるオブジェクトグラフを用いて実装を行う [10]．

ファイルシステムのようなある程度まとまった機能を提供するために 2 つ以上のオブジェクトを静的な依存関係によって結合し構成したオブジェクトのことを複合オブジェクト (composite objects) と呼び，その複合オブジェクトを構成する各オブジェクトを要素オブジェクト (component objects) と呼ぶ．要素オブジェクト間における参照などの静的な依存関係をリンクとみなし要素オブジェクトを節とみなすと，複合オブジェクト全体は要素オブジェクトの静的な依存関係によって決まるグラフと見ることができる．これをオブジェクトグラフ (object graph) と呼ぶ．

オブジェクトグラフではグラフに対する基本操作として，

1. オブジェクトグラフのコピー
2. オブジェクトグラフの破壊
3. オブジェクトの置換
4. オブジェクトの挿入
5. オブジェクトの削除

が定義されており，この基本操作をある元となるオブジェクトグラフに適用することで別の新しいオブジェクトグラフを作ることができる．

移動計算機環境に適したファイルシステムでは，この枠組を用いることで供給電源の状態に応じて 3 つの資源管理ポリシの切替えを行う．

第 3 章

実装

この章では移動計算機環境に適したファイルシステムの実装について述べる．まず本ファイルシステムの実装の元となるオブジェクトファイリングについて述べた後に今回本論文で実装に用いた Texas Persistent Storage について説明を行う．そして移動計算機環境に適応させるための要求仕様を述べた後，実装したシステムの説明を行う．最後に，2.1 説で述べた 3 つの管理ポリシについて今回の実装ではどのように行ったかを述べる．

3.1 オブジェクトファイリング

オブジェクトファイリングとは，通常のオブジェクト[1]に従来からあるファイルシステムが提供するファイルのような永続性を持たせたオブジェクトを提供する機構のことである．このようなオブジェクトを永続的オブジェクト (persistent object) と呼び，これに対する通常のオブジェクトを一時的オブジェクト (transient object) と呼ぶ．オブジェクトファイリングは，オブジェクト指向データベースシステム (OODBMS) を構築するための基本技術として用いられている．

一般的に永続的オブジェクトの実現にはディスクなどの 2 次記憶装置を用いてそこに実体が存在する事になるが，アプリケーションの実行時には物理メモリ上にも存在する．そのためディスク上にある永続的オブジェクトを利用するには，アプリケーションはディスクから物理メモリにそのオブジェクトを読み込む作業が必要になる．また，永続的オブジェクトの内容を変更した場合にはその結果をディスクに書き込む必要もある．オブジェ

[1] アプリケーションの終了時には消滅するもの

クトファイリングシステムは，このような永続的オブジェクトの読み込みや書き込みを容易に行える機構や自動的に行う機構を提供するものである．また，自動的に行うシステムには永続的オブジェクトと一時的オブジェクトの透過性を提供するものもある．

本論文で述べる移動計算機に適したファイルシステムは Texas 大[2] で開発された Texas Persistent Storage (以下 TPS と略)[17] を元に実装が行われた．TPS は永続的オブジェクトと一時的オブジェクトを透過的に扱うための C++ で書かれたライブラリであり，オブジェクトファイリングを実現する機構を提供している．これ以降の説明は，この TPS を中心に行う．

3.1.1 TPS の特徴

TPS を利用するアプリケーションでは一時的オブジェクトか永続的オブジェクトか意識して使用することはほとんど無い．アプリケーションプログラマに対するこの 2 つのオブジェクトの違いは一時的オブジェクトは`new` という通常の C++ の命令によって生成されるのに対し，永続的オブジェクトは `pnew` という TPS のマクロによって生成されるだけである．この`pnew` は TPS が提供するオブジェクトの生成関数 (`new`) によって定義されている．この 2 つのオブジェクトは両方とも仮想メモリ中に領域が割り当てられるのでアプリケーションプログラマは 2 つのオブジェクトを区別することなく扱うことが出来る．以下に TPS の特徴について列挙する．

- ページフォルトを使った効率的なアクセス (後述)．
- C++で書かれているため容易に他の環境で使うことが可能である．本論文執筆時点で Sun Sparcstation(SunOS4.1.3, Solaris 2.5)，DECstation(Ultrix 4.2)，486/Pentium PC/AT 互換機 (Linux 1.2.13) で動作が確認されている．さらに，386/486 PC/AT 互換機では OS/2 2.1 にも移植されている．また，C++で書かれているため，他の機種や OS にも容易に移植が可能である．
- モジュール性の高い構成であるため独立性がある．つまり，TPS では単に永続性をオブジェクトに付加しただけであるので，その他データの圧縮や暗号化などの機能を付け足すことが容易にできる．

[2] http://www.cs.utexas.edu/users/oops

- アプリケーションプログラマは永続か一時的かを意識することなく簡単にプログラミング可能である．2 つのオブジェクトの違いは生成時に `pnew` か`new` のどちらかを使うかだけである．
- オブジェクトの書き込みでは一旦ログに書き出してからファイルに記録するのでクラッシュが起きてもデータが失われることは無い．

3.1.2 オブジェクトの load と pointer swizzling

永続的オブジェクトと一時的オブジェクトを透過的に扱うには 2 次記憶装置に対する永続的オブジェクトの読み込み (load) と書き込み (store) をシステム側で行う必要がある．TPS では永続的オブジェクトの読み込みを効率的に行うためにページフォルト (page fault) を利用している．アクセスが行われる予定の永続的オブジェクトに対してある特別なページを用意し，そのページ中にこの永続的オブジェクトの領域を割り当てる．このページは予約ページ (reserved page) と呼ばれ，予約ページは読み書き不能 (NO ACCESS) のアクセス制限が行われる．そのため，予約ページにある永続的オブジェクトに対し読み込みなどのアクセスを行うとページフォルトが発生する[3]．TPS の実装では SIGSEGV というシグナルに対してシグナル処理ルーチンが設定されており，この処理ルーチンによってページフォルトを発生させたページを 2 次記憶装置から読み込む作業を行っている．この時読み込まれたページに対し，次に説明する pointer swizzling を行い，その後このページは読み出し専用 (READ ONLY) のアクセス制御に切り替わる．

通常 TPS では永続的オブジェクトは 2 次記憶装置に保存されている時と物理メモリに読み込んだ後の仮想記憶上に存在している時とでは配置の順序が異なる．そのため，永続的オブジェクトを指し示すポインタの値は 2 次記憶装置に保存されている時と仮想記憶上にある時では別の値になっている．TPS では 2 次記憶装置への保存は通常の UNIX などで使われているファイルに行われていることから，保存状態ではポインタの値はファイルの先頭からのオフセットとして表現されている．それに対し，仮想記憶上にある時には仮想記憶のアドレスとして表現されている．このように，永続的オブジェクトはページフォルトによって 2 次記憶装置から読み込まれただけでは仮想記憶上にあるオブジェクトとして参照することは難しく，永続的オブジェクトを示すポインタの値を仮想記憶上のアドレスに書き換える必要がある．この書き換える作業を pointer swizzling と呼び，これは永

[3] 正確には保護違反 (protection fault) が発生する

続的オブジェクトを実現する基本的な技術のひとつである．

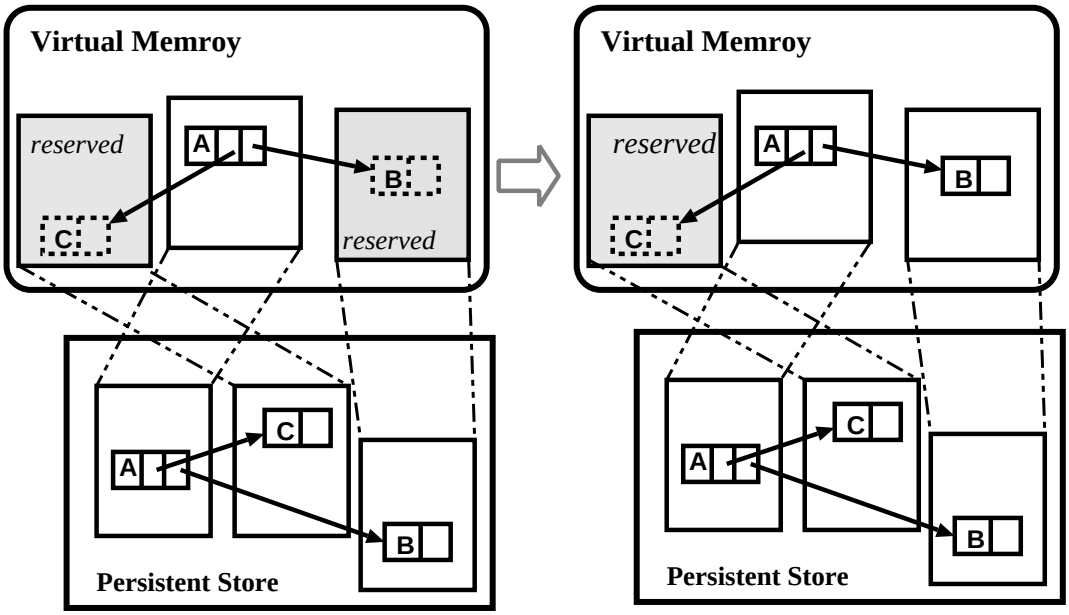

図 3.1: persistent object へのアクセスの様子

図 3.1は上記で説明した永続的オブジェクトの読み込みの様子を示している．図の左はオブジェクト A のあるページだけがメモリ上に存在していて，仮想記憶上のオブジェクト A は永続的オブジェクト B と C をそれぞれ指すポインタ (仮想記憶上のアドレス) を持っている．このポインタによって指されているオブジェクト B と C は実際にはまだメモリ上になく，これらのオブジェクトは予約ページ上にある．この時 B に対してアクセスを行うとページフォルトが発生し，右の図が示すようにディスクからオブジェクト B を含むページが読み込まれ，このページ内に含まれる永続的オブジェクトへのポインタに対し pointer swizzling が行われる．

この時，ポインタによって指されている永続的オブジェクトがメモリ中に存在しない場合，新たに予約ページが用意され，そのページ中のアドレスを指すようにポインタの値が書き換えられる．さらに，オブジェクトが提供するメソッドに対するポインタも仮想記憶上のアドレスに書き換えられる．

読み込みの終わった永続的オブジェクトに対してデータの書き換えを行うと保護違反が発生する．この時もページフォルト同様 SIGSEGV に対して設定されたシグナル処理ルーチンが呼び出されるのだが，今度は書き込み要求に対する処理を行う．それは，保護違反を起こしたページに対して dirty flag を設定し，読み書き可能 (READ WRITE) なア

クセス制御に切替える．この dirty flag は 2 次記憶装置へ保存するための書き込みの際に利用され，実際に変更のあったページだけ 2 次記憶装置への書き込みを行う事で処理の効率化を計っている．

2 次記憶装置への書き込みは，TPS が提供する `commit_transaction` がアプリケーションから呼ばれた時か，あるいはアプリケーションの終了時に自動的に行われる．

3.2 移動計算機環境に適したファイルシステム

移動計算機環境に適したファイルサービスを提供するために，供給電源の状態に応じた管理ポリシの切替えが必要となる．そして特にバッテリ稼働時の管理ポリシでは消費電力の節約が重要となるのだが，ファイルサービスにおいてはディスクの回転時間を短くする事が最も効果的であると考えられている (参照 2.1節)．そこで，本節ではディスクの回転時間を短くする方法を考察し，そのためのファイルシステムに対する要求についてまとめてみた．また，管理ポリシを切替えるための機構として本論文の実装ではオブジェクトグラフを用いた．そこで次にオブジェクトグラフについての簡単な説明を行う．また，本論文のファイルシステムではバッテリの節電対策の一つとしてフラッシュメモリを利用することを考え，これについての考察も述べる．

3.2.1 節電を行うための要求

ファイルシステムに対してバッテリ稼働中における消費電力の節約には以下に示す事が必要であると考える．

1. ディスクアクセスの制御
2. 仮想記憶管理
3. アプリケーションごとのカスタマイズ

ディスクアクセスの制御は，ディスクの回転時間を短くするための最も直接的な方法である．ディスクに対するアクセスを減らすことでその分ディスクの回転を止めることができる．ディスクアクセスを減らす方法としては，読み込みの場合アプリケーションの特性による局所性を利用してよく使うデータを物理メモリ中に長く置く方法がある．これは従来のファイルシステムで言うところのキャッシュのヒット率を上げることと同じである．

また，書き込みの場合はアプリケーションによる変更データの書き込みをできる限り物理メモリ中に溜めておき (buffering)，ある程度変更データが溜った時にまとめて書き込みを行う．

しかし，アプリケーションが行う読み込みや書き込みの制御だけではディスクの回転時間短縮には不十分である．ディスクのアクセスを完全に制御するには仮想記憶の管理が必要である．通常のオペレーティングシステムではファイルサービスの他に仮想記憶サービスも提供されている．仮想記憶はアプリケーションが実行されている計算機の物理メモリ以上の領域を提供することが可能である．そのため，アプリケーションが利用している計算機の物理メモリ以上の領域を要求すると通常物理メモリ中にある任意のページがディスクなどに書き出される．このことから，ディスクへのアクセスを制御するためにはアプリケーションが使用する仮想記憶の管理も行う必要がある．ちなみにファイルシステムと仮想記憶システムがそれぞれ独立に機能しているようなオペレーティングシステムではこのような仮想記憶によるディスクアクセスをファイルシステムだけで制御することは難しい．

ディスクアクセスの制御と仮想記憶管理の 2 つを行う事で，変更データのディスクへの書き込みとページアウトを統一して考えることができる．つまり，ページアウトのポリシをアプリケーション特有な性質を用いることで物理メモリ中のデータの局所性を上げることができ，よりディスクへのアクセスを減らすことが可能となる．そのためには，アプリケーションごとにページアウトのポリシなどオブジェクトファイリングの資源管理に関するポリシを自由に変更 (カスタマイズ) 可能でなければならない．

以上 3 つの要求を満たすことによってよりバッテリ消費の節約を効果的に行うことができる．

3.2.2 仮想記憶管理

移動計算機環境に適したファイルシステムにおいてバッテリ消費の節約を行うためには仮想記憶の管理が不可欠であった．そこで，本ファイルシステムでは仮想記憶に対して以下に示すことがらを主に管理する．

まず，アプリケーションが使用する物理メモリ量を制御する．これは，本ファイルシステムを利用しない他のアプリケーションによるページアウトがなるべく発生しないようにするためである．また，将来的に物理メモリの予約システムが導入された時にも物理メモ

リを制御するという機能は有効となる．物理メモリ量を制御するために，本ファイルシステムではページフォルトによってデータが読み込まれる時に読み込んだページ数のカウントを行う．これにより，現在のアプリケーションが使用している物理メモリ量を把握することができる．そして，アプリケーションが使用する物理メモリ量の値に応じてページアウトを行うことで使用する物理メモリ量を制御する．

次に，dirty ページの管理である．これはオリジナルの TPS と同様で変更データの効率的な書き出しを目的としている．

そして最後にページアウトを行うページの管理である．これはディスクアクセスの制御と関連している．ディスクへのアクセスをなるべく減らしたい場合はディスクに書き込む必要の無い，変更の行われていない clean ページから優先してページアウトを行うが，データの安全性 (信頼性) を保つ場合には dirty ページから優先してページアウトを行う．しかし，実際にはアプリケーションの特質によるデータの局所性を生かしたポリシを適用する方が望ましく，どのページを追い出すかは充分に考慮する必要がある．

3.2.3 フラッシュメモリ

本ファイルシステムではバッテリ消費の節約を行う時，節電の効果をより出すためにフラッシュメモリをディスクの一部として用いる．フラッシュメモリはディスクよりも消費電力が少なく，また不揮発性である事から移動計算機の 2 次記憶装置として好ましいとされている [7]．以下に，フラッシュメモリの特徴を並べてみた．

- 消費電力がディスクの 5 分の 1 から 6 分の 1
- メモリと同等の読み込み速度
- 書き込みはディスクより遅い
- 書換え回数に制限がある
- バイト単価はディスクより高い
- 不揮発性である

以上の特徴から，フラッシュメモリは移動計算機の 2 次記憶装置として望ましいと考えられるが，バイト単価が非常に高いためディスク容量全てを補うことは難しい．そこで，本ファイルシステムではフラッシュメモリをディスクに対する書き込みのログとして利用

する．フラッシュメモリをディスクのキャッシュとして利用した Brian ら [13] によるとフラッシュメモリの容量は全ディスク容量と同等である必要はなく，アプリケーションが必要とするワーキングセットが入るだけあれば良いと述べている．

フラッシュメモリをディスクのログとして利用すると変更データのディスクに対する書き込みは全てフラッシュメモリに行われるため，その分ディスクへのアクセスが減らせる．フラッシュメモリはディスクに比べて消費電力が少ないのでディスクのアクセスが減った分節電ができると同時に，フラッシュメモリは不揮発性であるので変更データの信頼性を保つこともできる．

また，ログとして記録されたフラッシュメモリ中のデータは読み出しが発生した時にキャッシュとしても利用できる．しかも，フラッシュメモリは通常の揮発性のメモリと同等の読み込み速度を持っていることから，読み出し性能も上がると期待できる．

ただし，フラッシュメモリは書き換え回数に制限があることから頻繁なデータの書き換えは効率的でない．そのため，フラッシュメモリに書き出されるログは変更箇所の少ないデータ構造であることが望まれる．

3.3 問題点

オブジェクトファイリングを実現する基本技術として pointer swizzling がある．TPS では pointer swizzling は読み込んだメモリ上でポインタの値を書き換える．そのため，オブジェクトファイリングとして管理しているページが読み出し専用であっても，オペレーティングシステムが提供する仮想記憶では dirty ページとして認識されてしまう．その結果読み出し専用であっても仮想記憶ではディスクに書き出す必要があるページとして考えられてしまう．オブジェクトファイリングでは，読み出し専用のページはページアウトの時にはディスクに書き出して欲しくないことから，これは効率が悪い．そこで，本研究では RT-Mach にページの dirty bit をクリアするシステムコール`vm_discard`(図 3.2参照)を付け足し，読み込み専用のページはこの`vm_discard` を使って dirty bit をクリアすることで仮想記憶でも clean なページとなるようにした．ちなみに，`vm_discard` にある引数の flag は `TRUE` の時には指定したアドレスの領域を解放してしまい，`FALSE` の時には指定したアドレスの領域を clean にする．

次に，TPS では予約ページは実際にある物理メモリが割り当てられその領域に対するアクセス制限を行うことで実現していた．しかし，これは実際にページフォルトが発生し

```
vm_discard(target_task, address, size, flag)
    mach_port_t   target_task;
    vm_address_t  address;
    vm_size_t     size;
    boolean       flag;
```

図 3.2: vm_discard の定義

データが読み込まれるまで使われることの無い領域であり，メモリ資源の乏しい移動計算機ではあまり好ましくない実装である．本オブジェクトファイリングでは，仮想記憶管理を RT-Mach が提供する外部ページャを用いて実装を行うため，実際に物理メモリを割り当てること無く予約ページを実現することができる．よって，TPS のように使用されない無駄な物理ページが存在するという問題は無い．

3.4 構成

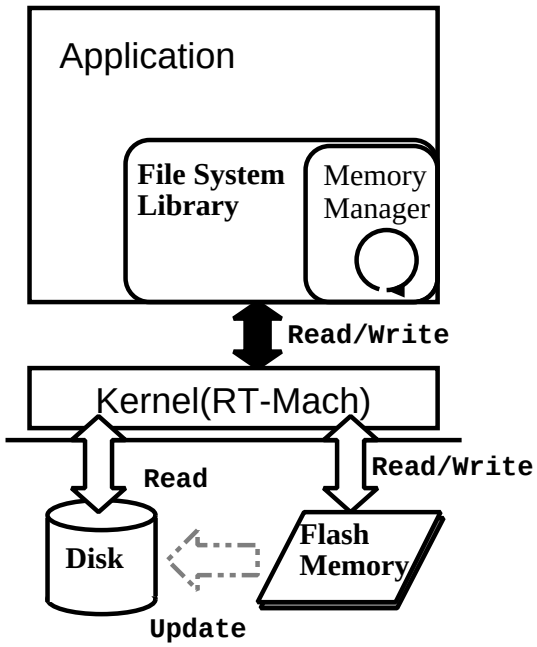

図 3.3: 全体構成

図 3.3に本ファイルシステムの構成を示す．本ファイルシステムは，RT-Mach 上で動作するアプリケーション用のライブラリとして提供される．本ライブラリは，通常のオブ

ジェクトファイリングが行う永続的オブジェクトに対する読み込みや書き込みのサービスを提供するとともに，ディスクやフラッシュメモリなど 2 次記憶装置への書き込みを制御する．

図中の Memory Manager はアプリケーションが使用する仮想記憶の管理を行っている．本オブジェクトファイリングでは，この Memory Manager は RT-Mach が提供する外部ページャ(External Memory Pager) を用いて実装されている．

外部ページャでは主にページフォルトの処理と，保護違反の処理を行う．以下に，主な外部ページャのインタフェース (関数) における処理の説明を行う．

vm_map 永続的オブジェクトが使用する仮想記憶の領域 (ヒープ) を確保する．

memory_object_data_request ページフォルトに伴う処理を行う．アプリケーションが使用している物理メモリのページ数をカウントし，2 次記憶装置からページフォルトを起こしたページを読み込み，pointer swizzling を行った後に，読みだし専用のアクセス制限をかけ`vm_discard` を使ってページを clean にする．

memory_object_data_unlock 保護違反に伴う処理を行う．保護違反を起こしたページの dirty flag を設定し，読み書き可能なアクセス制限をかける．

次に，オブジェクトグラフによる本ファイルシステムの構成を図 3.4に示す．図中の各 `P1`，`P2`，`P3` はそれぞれ AC-online 状態，battery high 状態，battery low 状態のページアウトのポリシオブジェクトである．また，`Persistent Store` とは 2 次記憶装置上の永続的オブジェクトを管理するオブジェクトであり，このオブジェクトによって永続的オブジェクトの実際の読み込みや書き込みが管理されている．また，battery 状態の時にある `Log Cache` はフラッシュメモリに書き出されるログの管理を行うオブジェクトである．

永続的オブジェクトの読み込みは外部ページャによって行われたが，ページアウトは図 3.4中の `Memory Manager` によって行われる．その時，どのページを追い出すかは各ページアウトのポリシオブジェクトによって決定される．ちなみに，実際のメモリ領域の解放は `vm_discard` を使って行われる．

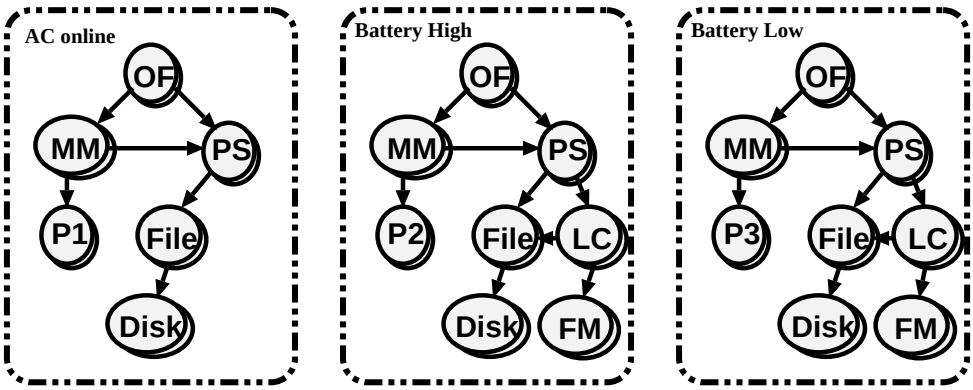

図 3.4: オブジェクトグラフによる構成

3.5 ポリシ

ここでは，3.4節で説明のあった図 3.4の中の各ページアウトポリシについて詳しく述べる．

Pageout Policy 1 AC 電源による電力供給を受けている時のページアウトポリシである．消費電力の節約を考慮する必要は無いのでディスクの回転を止める必要は無い．アプリケーションは可能な限り多くの物理メモリを使うことができるので，ページアウトは物理メモリに余裕がある限り発生しない．また，電力供給が止まる可能性は無いため多くの変更データはメモリ中に溜めておくことで，パフォーマンスの向上を目指す．

Pageout Policy 2 バッテリ電源による電力供給であり，かつバッテリの電力残量に余裕がある．バッテリの消費電力を極力少なくするため，ディスクへのアクセスは最小限にとどめられる．そのためには，変更データは物理メモリ中に溜めておく．また，アプリケーションが使用可能な物理メモリは Policy 1 に比べ少なめに設定される．ページアウトが発生した場合には，ディスクへのアクセスが起きにくい clean ページを優先してメモリから追い出す．しかし，どうしても 2 次記憶装置に書き出す必要がある場合には，フラッシュメモリ上にログとして書き込まれる．

Pageout Policy 3 バッテリ電源による電力供給であり，かつバッテリの電力残量が少ない．電力供給がいつ止まるかわからないため，変更データの安全性が一番に考えられる．データの変更があった場合にはできるだけ早く 2 次記憶装置に書き込まれるのだが，この場合にも Policy 2 と同様フラッシュメモリにログとして書き込まれる．また，利用可能な物理メモリも Policy 2 と同様に設定される．

第 4 章

評価

ここでは，本論文で実装した移動計算機環境に適したファイルシステムを使った実験について述べ，本ファイルシステムの評価を行う．

4.1 ベンチマーク測定

実験を行ったシステム構成は，ハードウェアとして TOSHIBA の PORTEGE 610CT(Pentium 90MHz, memory 24MB, disk 686.65MB) を使い，フラッシュメモリとして SunDisk の FLASHDISK (20MB) を使用した．

実験用プログラムとしては，OO7[5] を用いて実験を行った．データベースのファイルサイズは 13MB である．また，各ポリシにおけるアプリケーションが使用可能なメモリ量は，AC-online が 6MB とし，battery は両方ともに 3MB とした．

実行結果を表 4.1と 4.2に示す．表は両方とも秒単位で表した全実行時間とバッテリの消費量を表したものである．表 4.1はデータの読み込みだけを行うベンチマークの測定結果である．変更データが発生しないため書き込みを行う必要がないぶん利用できるメモリ量に比例して実行時間が短くなっていることがわかる．表 4.2は読み込みに加えてデータの変更も行う．書き込みの回数が最も多いと考えられる battery low が実行時間が一番長くなっている．また，ページアウトのポリシがほとんど同じとみなせる AC-online と battery high の実行時間の差が約 30 秒あるが，これは利用可能な物理メモリ量が AC 電源とバッテリとではバッテリの方が少ない分余計にページインやページアウトが多いと考えられる．この結果は，読み込みだけでのベンチマークでも書き込みもあるベンチマーク

でも同じような差が出ていることからほぼ間違いないだろう.

Policy	time(seconds)	battery (%)
AC-online	32.8820	0.30
Battery high	74.2660	0.70
Battery low	75.6770	0.70

表 4.1: only traverse

Policy	time(seconds)	battery (%)
AC-online	116.596	1.00
Battery high	144.066	1.20
Battery low	381.803	3.00

表 4.2: traverse and update

しかし，バッテリの消費量を見比べると実行時間に比例して消費量が大きくなっているようにみえる．これは，今回測定に用いた機種では細かな電源管理を行うことができず，結局どの状態でもディスクは同じように回転していたと考えられる．そこで，今度は上記の実験においてディスクへのアクセスをログとして記録し，それを元にシミュレーションによってバッテリの節電効果を計算してみた．結果は，図 4.1と 4.2である．表は，縦軸が全くディスクの回転を止めなかった場合のバッテリ消費を 100 とした時の割合を示しており，横軸がディスクアクセスが止まってから何秒後にディスクの回転を止めるかの閾値を示している．この閾値が 0 の場合，ディスクへのアクセスが無い間は完全に止まっていると考えられるので理想的な値と見ることができる．また，バッテリ稼働においてはフラッシュメモリにアクセスしている時間はディスクの回転が止まっているとみなして計算を行った.

図 4.1を見ると節電効果は AC 電源の時の方が高いが，これは使用できる物理メモリ量が多いためバッテリ稼働に比べてページインの数が少ないためと考えられる．図 4.2では battery low の方がはじめは battery high より節電効果が上になっているが，これは

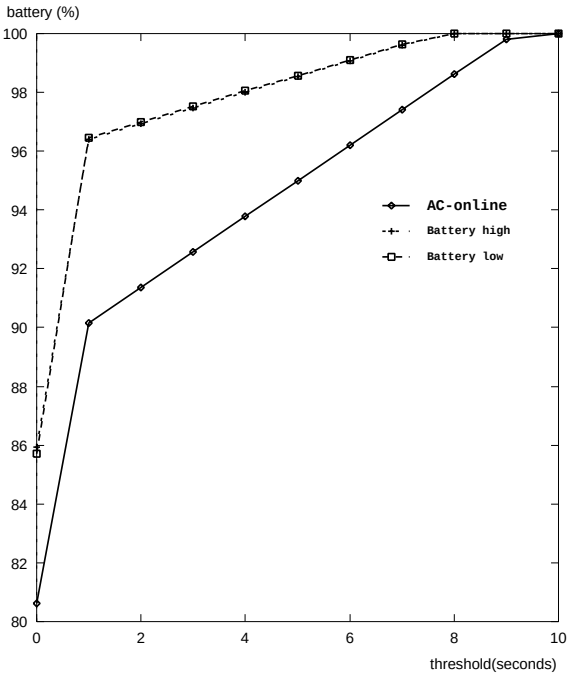

図 4.1: バッテリの節電効果 (only traverse)

battery low の方が全体的に実行時間が長くなっているためと考えられる．また，AC 電源の時と比べて，バッテリ稼働はかなり節電効果があるようにみえるが，これはフラッシュメモリへの書き込みに多くの時間がかけられているため，全体的にディスクへのアクセス時間が少ないと見られる．以上の結果を見て，状態に応じて資源管理ポリシを切替えることはかなり有効であることがわかるとともに，各ポリシの中でもパラメータの値の取り方によって効果が変わることがわかった．

4.2 議論

今回のベンチマークの結果ではバッテリ消費の節約はあまり効果が現れなかった．一つの原因として，今回測定に用いた機種ではディスクの回転を止めることによる節電効果よりも CPU による消費電力の方が大きく，CPU が動いているいじょうディスクの回転を止める効果があまり見られないと考える．また，今回ベンチマークに用いた OO7 というプログラムは汎用の OODBMS のパフォーマンスを計るために開発されたものであり，データベースのサーバーマシンとなるような非常に厳しい負荷 (load) が課せられるよう

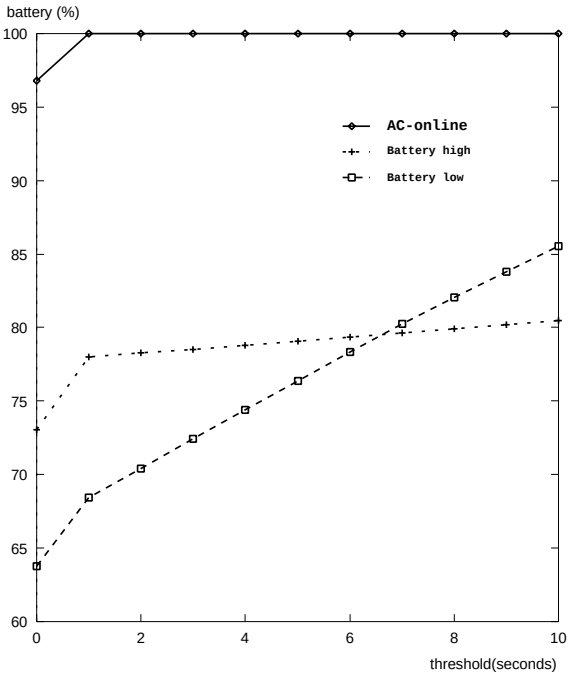

図 4.2: バッテリの節電効果 (traverse and update)

な設定になっていると考えられ，移動計算機で個人が使用するような設定とはかけ離れていると考える．そのため，ディスクへのアクセスが非常に頻繁に行われているため，実際にはディスクを止められる時間があまり無かった．また，今回測定に用いたベンチマークプログラムは 1 回の実行時間が APM が取得できるバッテリ残量の精度に比べて非常に短かったと考えられる．

また，今回測定に用いた機種ではソフトウェアからディスクの回転を止めることは可能であっても現在ディスクが回転しているかを知る方法が無い．そのため，より積極的なディスクのスピンダウンを行うことが非常に困難であることから，今回の実際の測定ではディスクのスピンダウンは行わなかった．

以上のことから，今回の教訓を生かしていくためにはハードウェアからのサポートが必要であると考える．まず 1 つ目はディスクが回転中か止まっているのかを知ることができる機構が必要である．そして 2 つ目はより積極的な電源管理をソフトウェアで制御できることである．例えば，ディスクやあるいはネットワークの I/O 待ちで CPU がアイドル状態の時には一時的に CPU への電力供給を止めたり，あるいはクロック数を下げるという

ことが考えられる.

第 5 章

関連研究

5.1 ディスクのスピンダウン・アップ

Fred Douglis ら [8] によると理想的なディスクのスピンダウンとアップを行うポリシ (off-line policy) では反応速度 (response time) を損なうこと無く通常の移動計算機に装備されているディスクが行う定期的なスピンダウンの場合と比べて 35 – 50 % の電力消費を抑えることができる．また，要求に応じてスピンダウンを行う間隔を変えるポリシ (THRESHOLD_DEMAND) でも理想的なポリシと比べて 7 – 52 % の節電効果があると報告されている．本研究のバッテリの残量に応じてシステムのメモリ管理ポリシを切替えるという発想は，この研究結果に基づいている．

5.2 フラッシュメモリ

Brian Marsh ら [13] のグループは本研究と同様にフラッシュメモリを移動計算機のディスクキャッシュとして利用している．その結果，フラッシュメモリのサイズがユーザのワーキングセットに対して充分ある場合，電力消費を 94%に抑えることが可能であり，さらに反応時間 (response time) を 32%に抑えることができるとある．また，彼らの行ったシミュレーションによると 11MB のフラッシュキャッシュで 61 年の耐久性があることがわかった．この結果から，フラッシュメモリをディスクのキャッシュとして用いることは効率的であることがわかる．彼らの研究と本研究の違いは，本研究はフラッシュメモリをキャッシュとして利用するのは移動計算機の消費電力を抑えることの手段にすぎず，本研

究はさらに状況に応じてフラッシュメモリをキャッシュとして利用したり，しなかったりという動的なシステム構成の変更を含んでいる．

5.3 圧縮キャッシュ

Fred Douglis [6] はプロセッサの処理能力 (速度) とディスクやネットワークなどの I/O の速度に大きな差 (gap) がある場合やメモリ容量が少ない場合にはメモリ上のデータを圧縮することで処理効率を上げることができると述べている．彼の研究は特に移動計算機に特化した話ではないが，携帯型計算機ではメモリやディスクの容量が少ないことから，データの圧縮は有効であると考えられる．今回本研究では圧縮の機能に触れる機会が無かったが，TPS の永続性と今回新たに付け加えられたメモリ管理機能は圧縮とは独立に存在することから，圧縮機能を本システムに拡張することは容易である．

5.4 適応的なシステム

Washington 大の SPIN[2] では spindle と呼ばれるカーネルモジュールを実行時にカーネルの中に組み込むことで拡張性と処理速度の高速化を同時に満たしている．これは本研究のアプローチとは全く異る方法である．次に，MIT の Exokernel[9] ではカーネルはハードウェアリソースの安全な多重化だけを行い後は全てユーザの制御可能としている．アプリケーション側でハードウェアリソースを制御しようという発想は，本研究と似ているところである．

また，ファイルシステムの研究においてアプリケーションごとのキャッシュ管理方式を導入した Pei ら [4] ではその有効性を示している．本研究もアプリケーションごとのカスタマイズを可能としているが，複数のアプリケーション間で管理方式の相違による性能効率に関しては今回は考慮していない．

第 6 章

おわりに

6.1 結論

本論文では，移動計算機に適したファイルシステムの設計・実装を行った．移動計算機環境では，システムは劇的な環境の変化に絶えずさらされており，状況に応じた資源管理方式やサービスを行うことが重要である．特に，バッテリ稼働時における消費電力の節約は大変重要である．移動計算機環境に適したファイルシステムにおいては，バッテリ消費の節約のためには 1. ディスクアクセスの制御 2. 仮想記憶の管理 3. アプリケーション特有な資源管理ポリシへの変更が必要となった．

実装を行ったシステムを実際に実行して評価を行ったところ，パフォーマンスや信頼性についてはまずまずの成果が結果として出た．しかし，バッテリ消費の節約に関しては，ハードウェアのサポートが不十分であったりして困難な面が多くあまり期待した結果は出なかったが，今後積極的にソフトウェアで電源管理を制御できるようになった時には本研究の有効性が示せるだろう．

6.2 今後の課題

バッテリ消費の節約を効果的に行うために，アプリケーションの特性に合わせたポリシの調整が必要である．また，今回は主にディスクアクセスの制御を行うことでバッテリ消費の節約を計ったのだが，CPU のバッテリ消費が最も大きいことから CPU に対する節電を行う必要もあると考える．

今回のベンチマークでは手軽に利用できるということで汎用的なベンチマークプログラムである OO7 を使用したが，次回にはより移動計算機で利用されるアプリケーションをターゲットとして選ぶ方が良いだろう．

また今回は，フラッシュメモリをディスクのログとしての利用しか考慮しなかったが，もっと積極的にキャッシュとして利用することも考えられる．その場合には，キャッシュ特有の問題として一貫性 (consistency) などを充分に考える必要がある．また，フラッシュメモリ内にあるログの管理は，今回はプロトタイプということで特に考慮していない面もあるが，データの局所性を出すための aging やあるいは GC などは必要であると考える．

さらに，今回アプリケーション特有な資源管理ポリシを考える時間があまり無かったが，より効果的に行うためにはページアウトにおける追い出すページの選択，ページアウトを行う使用メモリ量の閾値，一度に行うページアウトの数，ページアウトの時期などさまざまなことを考慮にいれる必要がある．

また，今回 2 次記憶装置への書き込みは全て 1 ページづつ同期的に行っていたので，これを改善する必要もある．さらに，プロトタイプであったためディスクのレイアウトなどは UFS のをそのまま利用していたが，オブジェクトファイリングに適したレイアウトを考えるべきであろう．

オブジェクトグラフを用いた資源管理モジュールの切替えでは，今回の実装では切替えるオブジェクトは実際には関数 (メソッド) しか含まれないもので行った．将来的には，状態 (status) を持ったオブジェクト，つまり変数も含むオブジェクトでも切替えが行えるような仕組みを考える必要がある．

謝辞

本研究を進めるにあたり多くの方の御協力を承り感謝致しております．以下にその感謝の念を記します．まず，本研究中もっとも主要な助言ならびに研究の進め方から一般的なプログラミング，さらには論文の作成まで面倒を見て下さった中島達夫助教授に心から御礼を申し上げます．次に，研究一般のことから研究成果発表の技法について助言をして下さった中島研究室博士後期過程の保木本晃弘さんにもお世話になりました．また，RT-Mach でのデバイスドライバの書き方から通常のプログラミングまで助言を頂いた赤木敏和さんにも同様にお世話になりました．他に研究で忙しい頭をリフレッシュするために，研究とは全く関係ないお話として社会現象や軍事関係・アニメーションなど多彩なことがらについてかなり専門的な話をして下さった則武淳さんと大平浩貴君．大平君は他にも jnethack についての専門的な会話も楽しませて頂きました．また，研究室内の交流を活発にされるため会合や合宿旅行など取り仕切ってくれた中野真紀子さん．そして，いつも物静かだけどいざという時にはちゃんと仕事をしてくれる寺田徹君．この他にも，篠田研の人々や，入学初期の堀口・阿部研仮配属の時に一緒だった人々にもお世話になりました．また，研究でなまった体を鍛えてくれた JAIST 公認バスケットサークルの倶楽部籠やの人達．同じく，適度な運動を与えてくれたインラインスケートの人達．そして，最後にここ北陸の地で生活していくのに充分な経済的支援をして下さった父と母に感謝致します．以上述べた方々に心から感謝の意をここに表します．

参考文献

[1] Kavita Bala, M. Frans Kaashoek, and William E. Weihl. Software prefetching and caching for translation lookaside buffers. In *Operating Systems Design and Implemenation (OSDI)*, pp. 243 – 253. USENIX, November 1994.

[2] Brian Bershad, Stefan Savage, Przemyslaw Pardyak, Emin Gun Sirer, David, Becker, Marc Fiuczynski, Craig Chambers, and Susan Eggers. Extensibility, safety and performance in the spin operating system. In *the 15th ACM Symposium on Operating System Principles*, Vol. 29, pp. 267 – 284. ACM, December 1995.

[3] Pei Cao, Edward W. Felten, and Kai Li. Application-controlled file caching policies. In *USENIX SUMMER 1994 TECHNICAL CONFERENCE*, pp. 171 – 182. USENIX, June 1994.

[4] Pei Cao, Edward W. Felten, and Kai Li. Implementation and performance of application-controlled file caching. In *Operating Systems Design and Implementation (OSDI)*, pp. 165 – 177. USENIX, November 1994.

[5] Michael J. Carey, David J. Dewitt, Chander Kant, and Jeffrey F. Naughton. A status report on the oo7 oodbms benchmarking effort. In *OOPSLA '94*, Vol. 29, pp. 414 – 426. ACM, October 1994.

[6] Fred Douglis. The compression cache: Using on-line compression to extend physical memory. In *Winter USENIX*, pp. 519 – 529. USENIX, January 1993.

[7] Fred Douglis, Frans Kaashoek, Braian March, Ramón Cáceres, Kai Li, and Joshua A. Tauber. Storage alternatives for mobile computers. In *Operating Systems Design and Implementation (OSDI)*, pp. 25 – 37. USENIX, November 1994.

[8] Fred Douglis, P. Krishnan, and Brian March. Thwarting the power-hungry disk. In *Winter USENIX*, pp. 293 – 306. USENIX, January 1994.

[9] Dawson R. Engler, M. Frans Kasshoek, and James O' Toole Jr. Exokernel: An operating system architecture for application-level resource management. In *the 15th ACM Symposium on Operating System Principles*, Vol. 29, pp. 251 – 266. ACM, December 1996.

[10] Akihiro Hokimoto, Kuniaki Kurihara, and Tatsuo Nakajima. An approach for constructing mobile applications using service proxies. In *The 16th International Conference on Distributed Computing Systems*, pp. 726 – 733. IEEE Computer Society, May 1996.

[11] Atsuo Kawaguchi, Shingo Nishioka, and Hiroshi Motoda. A flash-memory based file system. In *Proc. USENIX Technical Conference*, pp. 155 – 164. USENIX, January 1995.

[12] Yousef A. Khalidi and Michael N. Nelson. Extensible file systems in spring. Vol. 27, pp. 1 – 14, December 1993.

[13] Brian Marsh, Fred Douglis, and P. Krishnan. Flash memory file caching for mobile computers. Technical Report MITL-TR-59-93, Matsushita Information Technology Laboratory, Princeton, NJ 08640, June 1993.

[14] Vivek Narasayya, Tze Sing Eugene, Dylan McNamee, Ashutosh Tiwary, and Hank Levy. Reducing the virtual memory overhead of swizzling. In *Fifth International Workshop on Object Orientation in Operating Systems*, Seattle WA, October 27 – 28 1996.

[15] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. In *13th ACM Symposium on Operating Systems Principles*, pp. 1 – 15, 1991.

[16] M. Satyanarayanan. Fundamental challenges in mobile computing. In *15th ACM Symposium on Principles of Distributed Computing*. ACM, May 1996.

[17] Vivek Singhal, Sheetal Kakkad, , and Paul Wilson. Texas: An efficient, portable persistent store. In *Persistent Object System: Proceedings of the Fifth International Workshop on Persistent Object System*, pp. 11 – 33, September 1992.

[18] Christopher Small and Margo Seltzer. A comparison of os extension technologies. In *USENIX Technical Conference*. USENIX, January 1996.

[19] Paul R. Wilson and Sheetal V. Kakkad. Pointer swizzling at page fault time: Efficiently and compatibly supporting huge address spaces on standard hardware. In *International Workshop on Object Orientation in Operating System*, pp. 364 – 377, Paris, France, September 1992.