システムコールレベルのアクセスログを用いたディスクアクセスパターンマイニング

Disk Access Pattern Mining with System Call Level Access Log

上田 高徳[1]　平手 勇宇[2]
山名 早人[3]

Takanori UEDA　Yu HIRATE
Hayato YAMANA

ハードディスクへのランダムアクセスはボトルネックの大きな原因である．近年，アクセスログからデータマイニングの手法を用いてアクセスパターンを抽出し，キャッシュ管理に利用することが考えられている．本論文では，DBT-2（TPC-C）ベンチマークを実行した際のシステムコールレベルでのアクセスログに対して，シーケンシャルパターンマイニングの手法である CloSpan を適用することでアクセスパターンを抽出し，プリフェッチとキャッシュリプレースに利用した．提案手法では，ディスクへのシーケンシャルアクセスパターンをログからフィルタし，ログサイズを小さくすることでシーケンシャルパターンマイニングに掛る処理時間を短くする．実験により，フィルタを行った場合は CloSpan の実行時間を 13%以下に短縮でき，フィルタを行って抽出したパターンを用いた方がキャッシュミス率を改善できることを確認した．

Random access to hard disks is fatal bottleneck in current computer architecture. Recently, disk access patterns, which are extracted from disk access log with data mining techniques, have been considered as useful information for cache management. In this paper, we extracted disk access patterns by applying CloSpan, the sequential pattern mining algorithm, to system call level access log. Then we developed a prefetch and cache replacement technique by using extracted disk access patterns. To reduce the size of the access log dataset and the execution time of the sequential pattern mining, our method focuses on random disk access logs by filtering sequential disk access logs. Our experimental results show that, by filtering sequential disk access logs, the run time of CloSpan can be reduced to 13% and the cache miss ratio can be also reduced.

[1] 学生会員　早稲田大学基幹理工学研究科修士課程
ueda@yama.info.waseda.ac.jp
[2] 正会員　早稲田大学メディアネットワークセンター
hirate@yama.info.waseda.ac.jp
[3] 正会員　早稲田大学理工学術院，国立情報学研究所
yamana@yama.info.waseda.ac.jp

1．はじめに

ストレージシステムのボトルネックは古くからの研究対象である．ディスク容量は毎年 50%の割合で向上してきたが，アクセス速度の向上は毎年 8%にとどまっている[9]．結果として，ディスクドライブの容量にみあったアクセス速度が得られず，ストレージのボトルネックはますます深刻になっている．特にランダムアクセス速度の遅さは致命的であり，ランダムアクセスでディスク全体へアクセスするのに必要な時間は，1992 年のディスクでは 1 時間だったものが，2002 年には 80 時間となっている[9]．近年ではマルチコアCPUの普及に伴い，ストレージシステムへのアクセス集中がさらに大きな問題となってきている．マルチコアCPU環境では，多くのアプリケーションが並列に動作するため，ストレージへの並列アクセスが大量に発生するからである．特に，今後登場することが確実なメニーコアCPU環境では，アクセス集中の問題はますます大きなものになると予想される．ストレージシステムのボトルネック問題は，今後も大きな研究課題である．

ボトルネックの伝統的な解決策のひとつに，必要なデータを予測し，あらかじめメモリ上に読み込んでおく「プリフェッチ」がある．例えばLinuxは，シーケンシャルアクセスに対する先読を行う[2]．シーケンシャルアクセスの場合，アクセスされた直後のデータが近い将来アクセスされると期待できるため，プリフェッチが容易に行える．

さて，前述のようなランダムアクセスの問題を考えた場合，今後はランダムアクセスに対するアクセス高速化技術の実用化が不可欠と考えられる．ランダムアクセスの場合も，シーケンシャルアクセスの場合と同様にプリフェッチ等の高速化が行える．ただし，シーケンシャルアクセスのようにヒューリィスティクなアクセス予測は困難なため，アクセスパターンを抽出する必要がある．本論文では特に，ランダムアクセスのアクセスパターン抽出方法に注目する．

これまでにも，ファイル同士の相関や，ファイル内のブロック同士の相関を抽出し，キャッシュ管理に利用する研究が行われている．その多くは，木構造やグラフを用いてアクセス回数をカウントし，ファイル間の相関を抽出するものである[5][8]．ただし，木構造やグラフによるアクセスパターンの抽出は，ファイル数やブロック数に対するスケーラビリティの面で問題がある[6][11]．そこで，データマイニングの手法を用いてブロック間の相関を抽出する研究が行われている[6][11]．例えば[11]では，ストレージレベルでのアクセスログにシーケンシャルパターンマイニングを適用することでディスク上のブロック間の相関を抽出し，プリフェッチやディスクレイアウトに利用している．ただし，ストレージレベルでは，OSが管理している物理メモリ上のディスクキャッシュにヒットした場合，アクセスログを取得することができない．

本論文の手法では，システムコールレベルで取得したアクセスログに対してシーケンシャルパターンマイニングを適用し，アクセスパターンを抽出する．システムコールレベルであれば，ディスクキャッシュにヒットするアクセスであってもログに記録することができ，情報の損失がない．また，[11]では，ディスクへのシーケンシャルアクセスのパターンも抽出している．しかし，シーケンシャルアクセスはヒューリィスティクな予測が容易であり，計算負荷の大きいデータマイニング手法を用いて抽出するパターンとしては適

切でない．そこで本論文では，シーケンシャルアクセスのログをフィルタすることで，ログのサイズを小さくし，データマイニングに掛る処理時間を小さくした．また，フィルタを行わないログから抽出したアクセスパターンよりも，フィルタを行ったログから抽出したアクセスパターンを用いた方が，性能向上が可能なことをトレースドリブンなシミュレーションにより示す．

以下，２．で関連研究について述べる．３．においてログ取得方法の検討を行い，４．で提案手法を述べる．５．において評価を行い，６．でまとめを行う．

２．関連研究

プリフェッチの伝統的な方法としてヒント付きプリフェッチがある．これは，アプリケーション内で利用されているデータ構造などの知識を用いて，アプリケーション側からOSないしはストレージシステムへヒントを送るものである．ただし，ヒントを送るためにはアプリケーションの修正が必要であり，アプリケーション作成者の負担になる．そこで，コンパイラの補助により自動でI/Oのプリフェッチを行う研究もある[1]．これまで，メインメモリからのプリフェッチをコンパイラベースで行う研究[4][7]は盛んに行われており，ストレージに対するプリフェッチへの応用も期待できる．

一方，ヒントを用いずに，ファイルアクセスのパターンから自動的にストレージ内の相関性を抽出し，ストレージの高性能化に利用する研究が行われている[3][5][6][8][11]．ストレージシステム内の相関性は，ファイル相関（File Correlation）とブロック相関（Block Correlation）の２種類に分けられる．

ファイル相関とはファイル同士の関係性のことである．例えば，あるHTMLがアクセスされた場合，そのHTMLからリンクされている画像がアクセスされる可能性が高いことを考えると，HTMLファイルとリンクされている画像ファイルは強い相関があるといえる．ファイル相関が分かれば，キャッシュ管理に利用することが可能である．これまでファイル相関の抽出には木構造やグラフが用いられてきた[5][8]．例えば，重み付き確率グラフを用いてファイルの相関性を抽出し，プリフェッチを行うことで性能向上が可能である[5]．しかし，木構造やグラフによるアクセスパターンの抽出は，ファイル数に対するスケーラビリティの面で問題視されている[6][11]．

ブロック相関とは，特にディスク上のセクタ同士の相関を指す．しかし，ブロック数は膨大であるため，木構造やグラフを用いた方法では計算コストとメモリコストが大きい[6][11]．そこで，データマイニングの手法を用いてブロック相関を抽出する研究が行われている[11]．[11]はストレージレベルでのアクセスログにシーケンシャルパターンマイニングを適用し，ブロック間の関連を抽出することに成功している．ただし，ストレージレベルでは，OSが管理している物理メモリ上のディスクキャッシュにヒットした場合，アクセスログを取得することができない．また，[11]ではシーケンシャルアクセスのパターンも抽出している．シーケンシャルアクセスはアクセス予測が容易であり，計算負荷の大きいデータマイニング手法を用いて抽出するパターンとしては適切ではない．

本論文の目的は，OSレベルで取得できるログからアクセスパターンを抽出し，ディスクキャッシュ管理に利用することである．[11]とは異なり，システムコールレベルのログからアクセスパターンを抽出する．システムコールレベルであれば，ディスクキャッシュにヒットするアクセスであってもログに記録することができ，情報の損失がない．そのため，より正確なアクセスパターンを抽出できることが期待できる．また本論文では，シーケンシャルアクセスのログをフィルタすることで，ログのサイズを小さくし，シーケンシャルパターンマイニングに掛る処理時間を小さくした．

３．ログ取得方法の検討

ファイルアクセスのログを取得する際に検討するべきことは，どのアーキテクチャレベルのログを用いるかということである．本論文では評価に Linux を用いるため，以下では Linux を例に述べるが，他の OS にも同等のことがいえる．

３．１．　トレースログを取得するアーキテクチャ層

図 1はLinuxにおけるファイルシステムのアーキテクチャの概要図である．アプリケーションからファイルの読み込みがリクエストされると，Linuxは物理メモリ上にキャッシュされているか確認し，キャッシュされていない場合はファイルシステムに対して読み込みリクエストを行う．そして，ファイルシステムはデータがディスク上のどのブロックに格納されているか確認する．その後，I/Oスケジューラを介してデバイスドライバに読み込みリクエストが発行される．そして，デバイスドライバはディスクコントローラに対してリクエストを発行する．以降はハードウェアの領域である．

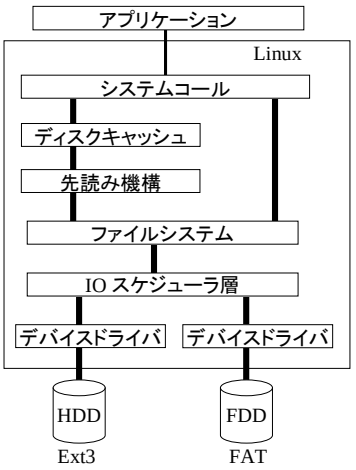

図 1: Linux のファイルシステムアーキテクチャ
Fig. 1: File System Architecture of Linux

ここで，アクセスログを取得するアーキテクチャ層として，以下の４通りを考える．

(1)　システムコールレベル
(2)　ファイルシステムレベル
(3)　デバイスドライバレベル
(4)　ハードウェアレベル

物理メモリ上のディスクキャッシュにヒットした場合，上記の(2)～(4)のレベルからはアクセスログが取得できないというデメリットがある．本論文が対象とするのは，(1)システムコールレベルのログである．次節より，(2)～(4)の各レベルにおけるログの性質について述べ，(1)については３．１．４で説明する．

3.1.1 ファイルシステムレベルでのログ

ディスクキャッシュにヒットしなかった場合や，更新されたデータを書き込む際には，ファイルシステムに対して要求が発行される．ファイルシステムの役割のひとつは，ファイル上のオフセットとストレージのセクタ番号の対応付けを行うことである．従って，ファイルシステムレベルでは，ファイルとセクタのマッピング情報も取得できる．一方で，ディスクキャッシュにヒットした場合はアクセスログが記録できない．また，ファイルの断片化によりファイル上では連続領域であっても，ディスク上では連続領域でない場合がある．このようなファイルの断片化はランダムアクセスの大きな原因である．本論文ではこの問題に対処するため，ファイルシステムと連携し，セクタ番号を用いてランダムアクセスの度合いを判定する．

3.1.2 デバイスドライバレベルでのログ

ファイルシステムがセクタ番号へのマッピングを行ったあと，デバイスドライバへのリクエストが行われる．デバイスドライバへのリクエストはセクタ番号と読み書きの種別で行われる．従って，デバイスドライバレベルからは，セクタ番号に対するアクセスログが取得できる．なお，ファイルシステムレベルと同様に，ディスクキャッシュにヒットしたアクセス情報は失われている．

3.1.3 ハードウェアレベルでのログ

ハードウェアレベルからは，アクセスログに加えて，データのディスク上での記録位置や，ディスクの回転角，ヘッドの位置といった情報が取得可能である．従って，ディスクの回転角やヘッドの位置を考慮した高速化も可能である．ただし，本論文が対象とするのは OS 内で取得可能なログを用いた，OS レベルでの高速化方法の検討である．従って，ハードウェアレベルでのログ取得は対象としない．

3.1.4 本論文の対象とするログ

本論文ではシステムコールレベルでのアクセスログを利用する．アプリケーションは一般に「read」といったシステムコールによりファイルへのアクセスを行う．そこで，本論文ではシステムコールレベルでログを採取する[4]．

ハードウェアレベルといったアーキテクチャの下位層では，物理メモリ上のディスクキャッシュにミスしたアクセスのみ記録できる，また，書き込みは物理メモリ上のキャッシュに溜められた後まとめて実行されるため，下位層から取得できる書き込みリクエストのログは，アプリケーションのアクセス順とは限らない．以上のことから，アーキテクチャの下位層のログからは誤った相関が得られる可能性がある．

システムコールレベルでは，ディスクキャッシュにヒットするアクセスも記録できるうえ，アプリケーションからのアクセス順序で記録できるため，情報の欠落がない．ただし，ディスクキャッシュにヒットする情報も含めてすべてのアクセスを記録すると，アクセスログが大量になるという問題がある．そこで我々は，シーケンシャルアクセスをフィルタすることで，ログサイズを小さくすることにした．ランダムアクセスのみのログを記録することで，ログ記録とパターン抽出のコストを小さくできる．

なお，システムコールレベルでは，ファイル名，ないしは i-node 番号とオフセットでの記録になり，ディスク上での記録位置が分からないという問題がある．この問題を解決するために，本論文ではファイルシステムとも連携することで，ディスク上のセクタ番号を取得し，フィルタに利用する．

4. 提案手法

本節では提案手法について述べる．提案手法ではまず，システムコールレベルで記録したトレースログに対して，シーケンシャルアクセスのログを取り除くフィルタを行う．そして，フィルタ後のログに対してシーケンシャルパターンマイニングのアルゴリズムである CloSpan[10]を適用し，頻出シーケンスを抽出する．そして，頻出シーケンスから相関ルールを生成し，プリフェッチに利用する．また，キャッシュリプレース方式として相関リオーダを提案する．

4.1. トレースログに対するフィルタ

本論文ではシステムコールレベルでのアクセスログを利用する．ディスクキャッシュにヒットするアクセスも記録できるため，情報の欠落がないと考えられる．その反面，アクセスログが大量になる．そこで我々は，ランダムアクセスパターンが高速化のために重要という観点から，シーケンシャルアクセスのログをフィルタすることにした．

システムコールレベルでは，i-node 番号とファイル内のオフセットでデータの位置を識別する．しかし，ファイルの断片化により，ファイル上では連続領域であってもディスク上では連続領域でない場合がある．また連続領域であったとしても，ファイル間にまたがるアクセスの場合，ディスクヘッドのシーク距離がどの程度になるか，i-node 番号とファイルオフセットのみでは判断できない．

そこで本論文では，i-node 番号とファイルオフセットから，データが記録されているセクタ番号をファイルシステムと連携することで求め，フィルタに用いる．言うまでもなく，シーケンシャルアクセスとは連続したセクタへのアクセスであり，ランダムアクセスとは連続していないセクタへのアクセスのことである．通常，セクタ番号が離れているほどディスク上での距離は離れ，シーク時間が大きくなる．

この特徴に注目して，リスト 1に示したように，最後のファイルアクセスと現在のファイルアクセスの先頭セクタ番号の差 /8 が閾値 d 以上のアクセスのみをログに記録することでフィルタを行う[5]．d を大きくするほど，シーク時間が短いアクセスが多くフィルタされることになる．すなわち，d をフィルタの強さとして考えることができる．ここで，$d=0$ の場合はフィルタ無しに相当する．$d=1$ の場合は，同じオフセットへの連続したアクセスがフィルタされる．

```
AccessLogger ( i-node 番号, オフセット, セクタ番号) {
    if ( 今回と前回のアクセスの先頭セクタ番号の差 /8 ≥ d ) {
        アクセスログに i-node 番号とオフセットを記録する.
    }
    else {
        アクセスログに記録しない.
    }
}
```

リスト 1. アクセスログのフィルタ方法
List 1. Filter Method for Access Log

[4] 便宜上，mmap などでメモリにマッピングされたファイルへのアクセスもシステムコールレベルに含める．

[5] 通常，Linux カーネル内でファイルは 4KB 単位でアクセスされるが，セクタ番号は 512B のため 8 で割る．

$d=2$では，シーケンシャルアクセスがフィルタされる．そして$d \geq 3$では，dを大きくするほどシーク時間が短いランダムアクセスがフィルタされる．このフィルタを通して記録されたアクセスログに対してシーケンシャルパターンマイニングを行う．

4．2．　シーケンシャルパターンアルゴリズムの適用

本節では，記録したアクセスログに対してシーケンシャルパターンマイニングを適用する方法について述べる．同様の内容は[11]で詳しく議論されているが，本論文ではランダムアクセスパターンのみを抽出するため，４.２.３で説明する相関アクセスパターンを生成する際に，シーケンシャルアクセスと時間的局所性を表すパターンを除去する点が[11]と異なる．

4.2.1　相関ルールと頻出シーケンス

相関ルール抽出はデータマイニング手法のひとつである．例えば，あるイベントAとBが共起する確率が高いならば，AとBは強い相関があるといえる．相関ルール抽出は，このような相関$A \rightarrow B$を抽出することである．

相関ルールはストレージシステム内の相関に当てはめることが可能である．あるデータAがアクセスされた直後にBがアクセスされる可能性が高いなら，$A \rightarrow B$という相関ルールがアクセスログから抽出できる．また，逆に$A \rightarrow B$という相関ルールが存在するならば，Aがアクセスされた直後にBがアクセスされると予想できる．従って，相関ルールを用いることで，プリフェッチ等が可能となる．

ただし，ストレージのアクセスパターンはアクセス順序に依存する．従って，シーケンスの順序を考慮することが可能な，シーケンシャルパターンマイニングのアルゴリズムを適用する必要がある．シーケンシャルパターンマイニングは，与えられたシーケンシャルデータベース（Dとする）から，与えられた最小サポート値以上の回数出現するシーケンスを抽出する手法である．例えば，

$$D = \{< ABC >, < ADBC >, < ADEFGB >, < ABC >\}$$

に対して，最小サポート値が 3 と設定された場合，

$$\{< A >: 4, < B >: 4, < C >: 3,$$
$$< AB >: 4, < AC >: 3, < BC >: 3, < ABC >: 3\}$$

が抽出される．ここで整数はサポート値である．本論文では，[11]でもアルゴリズムの基礎として利用されている，CloSpan[10]を用いた．CloSpanはClosed Frequent Sequenceを抽出するアルゴリズムである．Closed Frequent Sequenceとは，Frequent Sequence集合の中で，当該Closed Frequent Sequenceのスーパーシーケンスのサポート値と異なるFrequent Sequenceを指す．

例えば上記のDを対象として，最小サポート値を 3 と設定した場合，$< AC >: 3$が抽出されているが，$< ABC >: 3$というパターンに，$< AC >: 3$が含まれていると考えることができる．よって$< AC >: 3$は抽出する必要がない．データセットDを対象とした場合，CloSpan のアウトプットである Closed Frequent Sequence 集合は，

$$\{< AB >: 4, < ABC >: 3\}$$

である．

4.2.2　シーケンシャルパターンマイニングのアクセスログへの適用

本論文が扱うアクセスログが単一の時系列データであるのに対し，CloSpanはシーケンスデータベースに対するアルゴリズムである．そこで，アクセスログを適当な間隔でカットし，シーケンスデータベースを生成する必要がある．しかし，[11]でも議論されているが，ログをカットすることにより，カットされたトランザクション間にまたがるアクセスパターンが失われるという問題がある．ただし，カットの間隔が広いなら情報の損失は少ない[11]．従って本論文では[11]と同様に，カットウィンドウをオーバーラップさせない方法を採用した．

4.2.3　相関アクセスパターンの生成

抽出された頻出シーケンスパターンのそれぞれから，異なるアイテムを組み合わせてアクセスパターンを生成する．本論文ではアクセスパターンとして，ルールの左辺と右辺のアイテムが１つである$A \rightarrow C$といったパターンのみを生成する．例えば，

$$S =< A, B, B, C, A >$$

という頻出シーケンスからは

$$\{A \rightarrow C, B \rightarrow A, C \rightarrow A\}$$

というパターンを抽出する．ここで，本論文の目的はランダムアクセスパターンを抽出することなので，$A \rightarrow B$といったシーケンシャルアクセスのパターンや，$B \rightarrow B$といった時間的局所性を表すパターンは生成しない．以下では，xを左辺にもつアクセスパターンの右辺yの集合を

$$Correlate\ (x) = \{y \mid x \rightarrow y\}$$

であらわす．

4．3．　相関プリフェッチ

抽出されたアクセスパターンはプリフェッチに利用することが考えられる[11]．本論文における相関プリフェッチの実験においては，Aがアクセスされた場合に$Correlate(A)$をプリフェッチする．

ただし，プリフェッチはドライブの負荷を高める上，予測に失敗するとキャッシュ領域が無駄になるといったオーバーヘッドの問題がある．特に，ランダムアクセスに対するプリフェッチはよりコストが高いと考えられる．そこで，本論文では，キャッシュリプレースアルゴリズムとして，次節で説明する相関リオーダを提案する．

4．4．　相関リオーダ

現在，キャッシュリプレース方式としては LRU が用いられるのが一般である．本研究では，LRU の改良方式として相関リオーダを提案する．相関リオーダは，抽出したアクセスパターンに基づいて，近い将来アクセスされると予想されるキャッシュを LRU リストの末尾に移動させることである．すなわち，Aがアクセスされた場合に，

$$S = \{i \mid i \in Corrlate(A) \bigcap i \in LRU\text{リスト}\}$$

なるアイテム集合Sを LRU リストの末尾に移動させる．この動作は LRU リストの繋ぎ換えであるため，ほぼ定数オーダーで行え，ストレージへの負荷が増すことはないと考えられる．

5．評価

本論文ではアクセスログを用いてシミュレーションによる評価を行った．アクセスログを記録するワークロードにはTPC-C[13]のフリー実装であるDBT-2[12]を用いた．アクセスログの前半を用いてパターンを抽出し，アクセスログの後半でシミュレーションを行った．

5．1．　評価に使用するデータ

本論文では，評価のためのアクセスログを取得するワークロードとしてTPC-C[13]のフリーの実装であるDBT-2[12]を

用いた．ベンチマークで用いるデータベースソフトとして，PostgreSQL 8.2.0 を選択した．

記録したワークロードの概要を表 1に示す．TPC-Cは卸売り会社のワークロードをシミュレーションする．負荷はウェアハウス数とデータベースへの接続数により決まる．ここでは，ウェアハウス数を 50，接続数を 8 として，2 時間のワークロードを実行した．その結果，約 2,052 万回のアクセスがあった．なお実験では，フィルタ効果の評価を行うために，シーケンシャルアクセスも含み全ログを記録した．

5．2． フィルタの効果

記録したアクセスログの前半部分，すなわち約 1 時間分のログを用いてアクセスパターンを抽出する．まず，前半部分のアクセスログから，4．1．で説明した方法に基づいてフィルタを行った．フィルタの強さ d によるアクセスログ数の変化を図 2に示す．ここで $d=0$ はフィルタなしの場合に相当する．フィルタありのとき，ログのサイズは $d=2$ の場合に約 481 万アクセスとなったのが最大で，$d=67108864$（256GBのシーク距離に相当）の場合には約 14 万アクセスとなった．

パターン抽出のために，フィルタ後のログから，50 アクセスを 1 つのシーケンスとして区切り，アクセスパターン抽出用のテーブルを作成した．比較のために，フィルタ前の 1,026 万アクセスのログから，同じく 50 アクセスを 1 つのシーケンスとして区切り，約 21 万シーケンスのテーブルを作成した．生成されたテーブルに対して CloSpan を最小サポート値 45 で適用した．このサポート値は，フィルタ無しの $d=0$ のときに，CloSpan の処理時間がアクセスログの取得時間と同じ約 1 時間に近くなる値を選択した．

CloSpanの適用後，4．2．3 で説明した方法に基づいてパターンを生成したところ，抽出されたパターン数は，図 2に示すようになった．フィルタなしの場合，抽出されたパターン数は 19,127 パターンとなった．フィルタありの場合，$d=16$ の時に 8,697 パターンとなったのが最大であった．表 2の環境におけるCloSpanの実行時間と使用メモリは，フィルタなしの場合が 3,444 秒及び 43.91MBであり，フィルタありの $d=2$ の時に 434 秒及び 13.26MBであった．フィルタによるシーケンス数の減少により，フィルタを行った方がより少ない時間とメモリで処理が行えることが分かる．一方で，フィルタありの場合に抽出されたパターン数は，フィルタなしの場合よりも少なくなっている．次節において，このパターン数の減少が性能にどのような影響を与えるか評価する．

5．3． シミュレーションによる評価

本論文では，記録したログの後半に対し，抽出したパター

表 1. 記録したワークロード
Table 1. Traced Work Load for Experimentation

データベースソフト	PostgreSQL
バージョン	8.2.0
ベンチマークソフト	DBT-2 0.40
ウェアハウス数	50
接続数	8
テスト時間	7,200 秒
アクセス数	20,818,239

表 2. CloSpan 実行環境
Table 2. Environment for CloSpan Execution

CPU	Intel Xeon 5110 ×2
Memory	DDR2 533 16GB
OS	Linux 2.6.9-5.EL (x86_64 smp) (Red Hat Enterprise Linux WS Release 4)

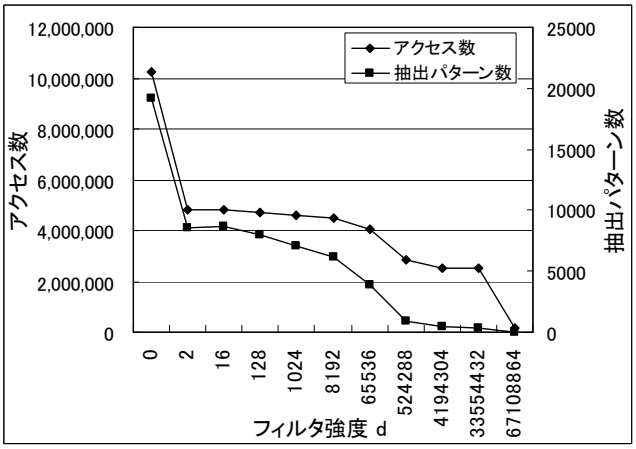

図 2. フィルタ強度によるアクセス数と抽出パターン数の変化
Fig. 2. The Change of the Number of Logs and Extracted Patterns

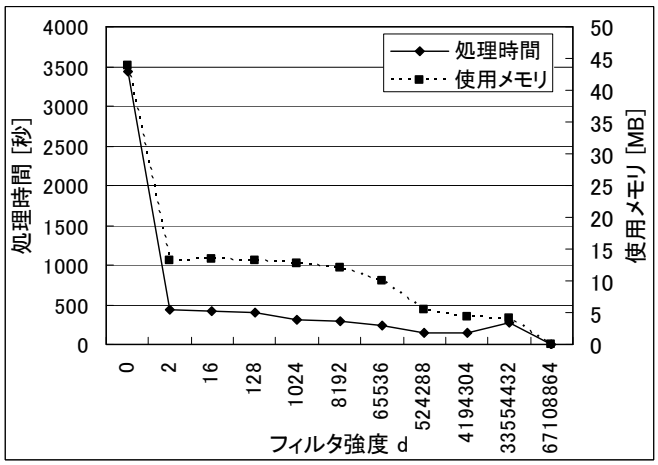

図 3. フィルタ強度による処理時間と使用メモリ量の変化
Fig. 3. The Change of Run-time and Memory Usage by Filter Strength

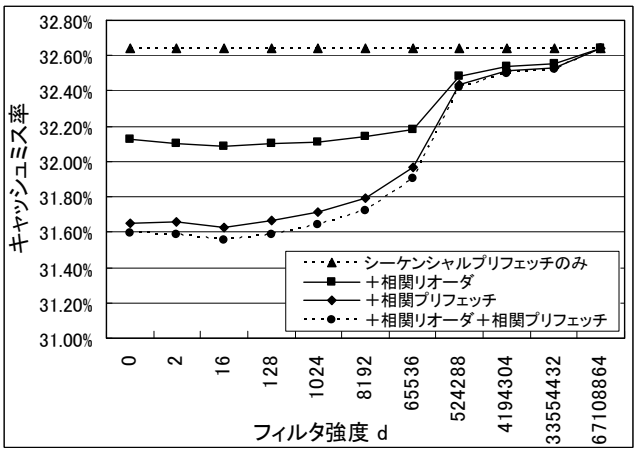

図 4. フィルタ強度によるキャッシュ性能の変化
Fig. 4. The Change of Cache Performance by Filter Strength

ンを用いてトレースドリブンなシミュレーションによる評価を行い，キャッシュミス率を比較した．シミュレーションのキャッシュ容量は 256MB とした．

結果を図 4に示す．実験結果によると，相関リオーダよりも相関プリフェッチの効果の方が高いことが分かる．相関リオーダはLRUリスト内のみの並び替えであるから当然の結果といえる．ただし，相関リオーダのみでも，フィルタありの $d=16$ の場合に，最大 0.55%の改善が見られた．特に，相関リオーダで並び変えているデータはランダムアクセスされるデータであるから，このキャッシュミス率の改善によるレスポンスタイム減少の効果は大きいものと考えられる．また，フィルタ適用の有無に関わらず全ての場合で，相関リオーダと相関プリフェッチを同時に利用した場合が最も性能がよくなった．

注目するべきことは，フィルタを利用してもキャッシュミス率の改善がみられることである．特に $d \leq 128$ では，フィルタを利用しなかった場合よりもキャッシュ性能の改善がみられる．フィルタなしの場合，相関リオーダと相関プリフェッチの両方を用いることで 1.05%の改善がみられた．一方，$d=16$ のフィルタを施して相関リオーダと相関プリフェッチの両方を行ったところ，最大で 1.09%の改善がみられた．

以上の結果から，ログをフィルタすることで，シーケンシャルパターンマイニングによるアクセスパターン抽出の処理時間を小さくでき，さらにフィルタなしの場合よりも少ないパターンで効率良く性能の改善を図れることが分かる．

6. おわりに

本論文では，システムコールレベルでのトレースログに対し，シーケンシャルパターンマイニングを用いてアクセスパターンを抽出した．シーケンシャルアクセスをログから取り除き，ログのサイズを小さくすることで，シーケンシャルパターンマイニングアルゴリズムである CloSpan の実行時間を短縮することができた．また，抽出したアクセスパターンをファイルアクセスのプリフェッチとキャッシュリプレースに適用した場合の性能についてシミュレーションで評価した．その結果，TPC-C によるトレースログでは，最大で 1.09%のキャッシュヒット率の改善がみられた．また，フィルタを用いた方が，フィルタなしの場合よりも少ないパターンで効率良く性能の改善を図れることが分かった．

本論文はキャッシュミス率のみの評価になっており，レスポンスタイムが考慮されていない．より現実に即した評価を行うためにも，実際に Linux に実装し，実機による評価を行う予定である．

[謝辞]

この研究は，早稲田大学グローバルＣＯＥプログラム「アンビエントＳｏＣ教育研究の国際拠点」（文部科学省研究拠点形成費補助金）により支援された．

[文献]

[1] A.D.Brown and T.C.Mowry, “Compiler-Based I/O Prefetching for Out-of-Core Applications,” *ACM Trans. on Computer Systems*, vol.19, no.2, pp.111-170, May 2001.

[2] D.P. Bovet, M. Cesati, *Understanding the Linux Kernel, Third Edition*, O'Reilly, Nov. 2005.

[3] G.A.S. Whittle, J.F. Pâris, A. Amer, D.D.E. Long, and R. Burns, “Using Multiple Predictors to Improve the Accuracy of File Access Predictions,” In *Proc. of the 20th IEEE/11th NASA Goddard Conf. on Mass Storage Systems and Technologies* (MSS'03), pp.230-240, San Diego, US-CA, Apr. 2003.

[4] H. Kasahara and A. Yoshida, “A Data-Localization Compilation Scheme Using Partial Static Task Assignment for Fortran Coarse Grain Parallel Processing,” *Journal of Parallel Computing*, vol.24, issue 3-4, pp.579-596, May 1998.

[5] J.Griffioen and R.Appleton, “Reducing File System Latency using a Predictive Approach,” In *Proc. of the USENIX Summer 1994 Tech. Conf.*, Boston, US-MA, Jun. 1994.

[6] L.Yu, G.Chen, and J.Dong, “Mining Infrequently-Accessed File Correlations in Distributed File System,” In *Proc. of the Joint Conf. of the 9th Asia-Pacific Web Conf. and the 8th Int'l Conf. on Web-Age Information Management* (APWeb/WAIM 2007), pp.630-641, Huang Shan, China, Jun. 2007.

[7] T.C. Mowry, M.S. Lam and A. Gupta, “Design and Evaluation of a Compiler Algorithm for Prefetching,” In *Proc. of the 5th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems* (ASPLOS-V), Boston, US-MA, Oct. 1992.

[8] V.Vellanki, A.L.Chervenak, “A Cost-Benefit Scheme for High Performance Predictive Prefetching,” In *Proc. of the 1999 ACM/IEEE Conf. on Supercomputing* (SC99), Portland, US-OR, Nov. 1999.

[9] W.W. Hsu, A.J. Smith, H.C. Young, “The Automatic Improvement of Locality in Storage Systems,” *ACM Trans. on Computer Systems*, vol.23, no.4, pp.424–473, Nov. 2005.

[10] X.Yan, J.Han, and R.Afshar, “CloSpan: Mining closed sequential patterns in large datasets,” In *Proc. of the 2003 SIAM Int'l Conf. Data Mining* (SDM'03), San Francisco, US-CA, May 2003.

[11] Z.LI, Z.Chen, and Y.Zhou, “Mining Block Correlations to Improve Storage Performance,” *ACM Trans. on Storage*, vol.1, no.2, pp.213-245, May 2005.

[12] OSDL - OSDL Database Test 2, http://old.linux-foundation.org/lab_activities/kernel_testing/osdl_database_test_suite/osdl_dbt-2/.

[13] Transaction Processing Performance Council, http://www.tpc.org/.

上田 高徳 Takanori UEDA

2007 早稲田大学理工学部ＣＳ学科卒業．現在，早稲田大学大学院基幹理工学研究科修士課程在学中．Web 解析，情報検索，オペレーティングシステムなどの研究に幅広く従事．ACM，IEEE，IEICE，IPSJ，DBSJ 各学生会員．

平手 勇宇 Yu HIRATE

2005 早大・理工学研究科修士課程修了．2008 早大・理工学研究科博士課程修了．博士（工学）．2006 年より同大・メディアネットワークセンター助手．ACM，IEEE，IPSJ，DBSJ 各会員．

山名 早人 Hayato YAMANA

1993 早大・理工学研究科博士課程了．博士（工学）．1993-2000 電総研．2000 早大・理工助教授．2005 同大理工学術院教授，現在に至る．ACM，IEEE，IEICE，IPSJ，DBSJ 各会員．