多周期的更新アクセスに適した二次記憶管理技法
——連続的 Web クローリングへの適用——

田村　孝之[†,††a)]　　喜連川　優[††b)]

Secondary Storage Management for Multi-Periodic Updates

Takayuki TAMURA[†,††a)] and Masaru KITSUREGAWA[††b)]

あらまし　本論文では大規模なデータベースの各レコードを独立周期で更新するアクセスパターンに適した新たな二次記憶編成方式を提案する．このようなアクセスパターンは，筆者らが取り組んでいる大規模 Web アーカイブ構築のための連続的 Web クローリングに特徴的なものである．連続的 Web クローリングにおいては，Web ページごとに更新頻度を推定し，アクセス間隔を適応的に制御することで収集の効率化を図るが，更新頻度推定に必要な Web ページごとの状態情報をアクセスのつど更新する必要が生じ，その更新負荷が大規模化の妨げとなってしまう．提案手法はレコードのアクセス予定時刻の知識を利用してデータを配置するものであり，大規模 Web クローリングの実データを用いた評価によりその劇的な効果を実証する．

キーワード　連続的 Web クローリング，二次記憶編成，シーケンシャルアクセス，ランダムアクセス，B-木

1. まえがき

筆者らは Web の構造や時間変化の分析に基づく社会知の抽出に取り組んでおり，その基盤として日本の Web 情報を網羅的に収集・蓄積した大規模な Web アーカイブの構築を進めてきた [1], [2]．Web 空間全体を一括収集する従来のクローリングでは時間分解能に限界があることから，個々の Web ページを独立した周期で繰り返しアクセスし続ける連続的クローリングにより Web アーカイブの維持を行っている [3], [4]．

連続的 Web クローリングは過去のクローリングの実績を Web ページごとの状態情報としてデータベース化し，これに基づいて各 Web ページの更新頻度を推定することで最適なアクセススケジュールを導出するものである．また，新たなアクセスの結果を即座に状態データベースに反映し，更新頻度推定精度の継続的な向上を図る．

Web ページ状態情報は Web ページのコンテンツ本体と比較すると小さいものの，10 億ページを超える大規模クローリングにおいては状態データベース全体は数百 GByte 規模に達し，軽視することはできない．一方，Web ページコンテンツは単純な構造のリポジトリに追記されるのが通常であり [5]，更新負荷はむしろ低いと考えられる．したがって連続的 Web クローリングの効率化のためには，大規模な状態データベースに対して発生する更新の負荷を軽減することが不可欠となる．同様の問題は，各種のセンサが大量に配置されるに伴い，それらが発信する情報の網羅的かつ効率的な収集を試みる際にも発生し，今後その重要性が増すものと考えられる．

データベースの更新負荷軽減のために従来提案されて来た手法として，キャッシュやログ構造化などが挙げられる．しかし，クローリングにおいては相手 Web サーバに負荷を集中させないようにするため，アクセス先の局所性を避けるような制御をすることが一般的であり，大規模クローリングにおけるキャッシュの効果は期待できない．一方，ログ構造化技法は更新後の書込みデータを，その元の位置にはかかわらず連続してディスクに書き込むことで書込みのスループットを

† 三菱電機株式会社情報技術総合研究所，鎌倉市
Information Technology R&D Center, Mitsubishi Electric Corporation, 5–1–1 Ofuna, Kamakura-shi, 247–8501 Japan
†† 東京大学生産技術研究所，東京都
Institute of Industrial Science, The University of Tokyo, 4–6–1 Komaba, Meguro-ku, Tokyo, 153–8505 Japan
a) E-mail: ttamura@acm.org
b) E-mail: kitsure@tkl.iis.u-tokyo.ac.jp

上げるものであるが，書込み後のデータが断片化し，再度読み込む際に性能ペナルティとなることが知られている．連続的 Web クローリングは，更新が周期的かつレコードごとに異なる多周期的更新アクセスが発生することが特徴的であり，その特性に合った手法の開発が必要である．

そこで本論文では，上記のようなレコードごとの固有周期による多周期的更新アクセスにおける性能を向上する二次記憶編成方式を提案する．本方式では，更新周期に関する知識を利用し，次の読込み時期に応じてレコードをクラスタリングする．また，書込みにおいては大域的なバッファリングによりランダムシークの発生を抑え，性能低下を防止する．

以下，**2.** では連続的 Web クローリングとその多周期的更新アクセスについて述べるとともに，従来手法の問題点を明らかにする．**3.** では提案する二次記憶編成方式について説明し，**4.** で大規模 Web クローリングの実データを用いた性能評価を行う．最後に，**5.** で全体のまとめを行う．

2. 連続的 Web クローリングの処理モデル

2.1 Web ページ更新間隔推定と状態情報

筆者らは大規模 Web アーカイブを維持するために，2005 年から網羅的な連続的 Web クローリングを実施している．最近では Web ページの更新周期を RSS・Atom フィード，sitemaps ファイル [6] 等により明示する Web サイトも増えつつあるものの [7]，ほとんどの Web ページの更新周期は不明であり，クローラは Web ページ内容の継続的な観測結果からその値を推定する必要がある [8], [9]．筆者らは，Poisson 過程の最ゆう推定による Cho ら [10] の手法に基づき，次のように Web ページの更新間隔推定を行っている [3], [4]．すなわち，ある Web ページの初回アクセスからの経過時間を T，その間に行ったアクセスで更新が検出されなかった際のアクセス間隔の累積値を U，更新が検出されたアクセス間隔の最小値 $t_{c\min}$ 及び平均値 $t_{c\,\mathrm{avg}}$ の幾何平均 $\bar{t_c} = \sqrt{t_{c\min} \cdot t_{c\,\mathrm{avg}}}$ を用い，以下の式 (1) により当該 Web ページの更新間隔 λ^{-1} を推定する．

$$\hat{\lambda}^{-1} \simeq \frac{\bar{t_c}}{\log \dfrac{T}{U}} \quad (0 < U < T), \tag{1}$$

クローラは各 Web ページについて，T，U，及び $\bar{t_c}$ に対応する状態情報をデータベースとして維持する．表 1 にその属性の一部を示す．式 (1) の T は Last visited と Epoch の差から求め，$t_{c\,\mathrm{avg}}$ は $(T-U)/m$ により求める．また，Checksum は最終アクセス時における Web ページ内容のハッシュ値（MD5 による）であり，次回アクセス時に Web ページの更新有無を検出するために用いる．このほかにも，HTTP 要求ヘッダに含めるための情報や，直近のエラーの発生傾向，プライオリティ制御情報などが含まれており，URL 等の可変長文字列を除いても 1 レコード当り約 100 バイトとなっている．この値は平均的な Web ページのサイズである約 20 KiB(注1) と比較すると 2 けた小さいが，10 億ページに対しては 100 GiB 規模となり，巨大なデータベースを構成する．

表 1　連続的 Web クローリングにおける Web ページ状態情報の属性（抜粋）
Table 1　An excerpt from the per-page state record for continuous crawling.

属性名	データ型	用途
URL	文字列	HTTP 要求生成
Referer	文字列	HTTP 要求生成
Last modified	日付	HTTP 要求生成
Etag	文字列	HTTP 要求生成
Status	整数	HTTP 応答保存
History	整数	定常的エラーの検出
Checksum	文字列	更新有無の検出
Epoch (T_0)	日付	更新間隔推定
#Visits	整数	更新間隔推定
#Versions (m)	整数	更新間隔推定
Last visited ($T+T_0$)	日付	更新間隔推定
Stable period (U)	整数	更新間隔推定
Min. interval ($t_{c\min}$)	整数	更新間隔推定
Interval (λ^{-1})	整数	更新間隔推定結果
...	...	...

一方，Web ページ全体に対する単一ハッシュ値を用いる更新検出方法には，内容の些細な部分（アクセスカウンタ，日付，及び広告等）が変化しただけでも更新とみなしてしまうという問題がある．Olston ら [11] は，Web ページを複数の部分に分割し，それぞれに対して一致判定を行うことで更新検出の精度を高める試みについて述べている．この場合，状態情報には複数のハッシュ値属性を保持する必要が生じる．また，式 (1) ではアクセス間隔の統計値のみを用いているが，推定の精度を上げるために最近のアクセス間隔の値をいくつか状態情報に保持することも考えられる．このように，更新間隔推定の精緻化を考慮すると，状態情報のレコード長は更に大きくなると想定する必要がある．

(注1)：KiB $= 2^{10}$ Bytes，MiB $= 2^{20}$ Bytes，GiB $= 2^{30}$ Bytes.

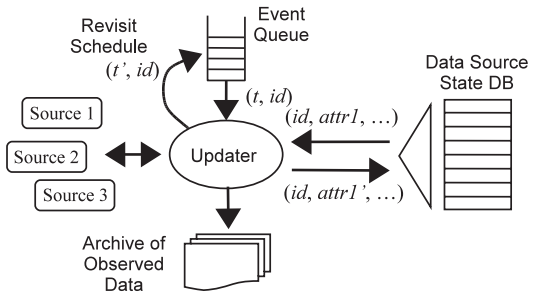

図 1　多周期的更新アクセスの処理モデル
Fig. 1　A processing model of multi-periodic updates.

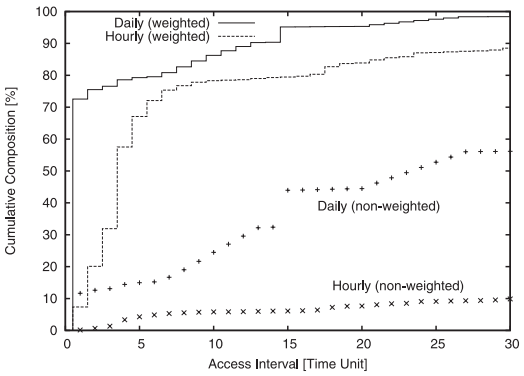

図 2　単位時間内にアクセスする Web ページ集合におけるアクセス間隔の累積組成比
Fig. 2　Cumulative access-interval composition of Web pages accessed in a time unit.

2.2　多周期的更新アクセス

連続的 Web クローリングにおける Web ページ状態情報更新の動作は，図 1 のようにモデル化することができる．すなわち，更新プロセス（Updater）が複数の外部情報源（Source 1, Source 2, ...）をそれぞれに固有の周期で観測するとともに，各情報源固有の管理情報を格納した情報源状態データベース（Data Source State DB）への参照・更新を行うというものである．イベントキューは情報源識別子 id と観測予定時刻 t を対応づけて管理し，情報源ごとの観測タイミングを制御する．情報源状態データベースの各レコードは情報源識別子 id により識別される属性集合（$attr1, ...$）からなり，観測に先立ってプロトコルパラメータ等を得るために読み出され，観測後に統計情報等を更新した結果（$attr1', ...$）が書き戻されるものとする．また，更新プロセスは当該情報源の次回観測時刻（t'）を決定し，識別子 id とともに再度イベントキューに投入する．なお，観測データ自体はアーカイブに追加され，別途異なるプロセスにより処理されるものとする．

本モデルにおいては，情報源観測，すなわち状態データベースのレコードアクセスのタイミングは更新プロセスにとって完全に既知であるが，各レコードのアクセス順序は見掛け上ランダムになる．本論文ではこのようなアクセスパターンを多周期的更新アクセスと呼び，大量の情報源からの情報収集を特徴づけるものとして扱う．観測データの格納は大きなデータ量を伴う可能性があるもののシーケンシャルなアクセスが主体となるため，情報収集全体の性能を決定づけるのは状態データベースに対する多周期的更新アクセスの効率であると考えられる．

2.3　B-木に基づく実装と問題点

情報源状態データベースの一般的な実装方法として，識別子 id をキーとする検索構造を B-木により実現することが考えられる．イベントキューには識別子 id がほぼランダムに並ぶため，1 件ずつ B-木をアクセスすると完全なランダムアクセスとなって I/O 性能が大幅に低下してしまう．そこで，観測イベントの時間分解能を 1 日あるいは 1 時間等に設定し，単位時間内に観測するすべての情報源の識別子をソートしてから B-木へのアクセスを行うことで，性能低下を緩和する必要がある．

図 2 は，実際の連続的 Web クローリング(注2)における Web ページ更新間隔推定結果に基づき，再収集の単位時間（1 日（Daily）または 1 時間（Hourly））内にアクセスする Web ページ集合について，アクセス間隔の組成を示したものである．τ を整数とし，アクセス間隔（= 更新間隔推定値）τ をもつ Web ページの数を $f(\tau)$ 個とすると，そのような Web ページをある単位時間内にアクセスする確率は τ^{-1} であり，その数は $\tau^{-1}f(\tau)$ となる．単位時間内にアクセスする Web ページ数の期待値 $\sum_\tau \tau^{-1}f(\tau)$ との比 $\tau^{-1}f(\tau)/\sum_\tau \tau^{-1}f(\tau)$ の累積値をプロットしたものが図 2 の weighted である．一方，non-weighted は $f(\tau)/\sum_\tau f(\tau)$ の累積値である．

図 2 は情報源状態データベースの各レコードに対するアクセス傾向に対応している．単位時間 1 日の場合はアクセスするレコードの 70%以上が次の日もアクセ

(注2)：2008 年 12 月から 2009 年 2 月にかけて実施．再収集の単位時間は 1 日としたが，HTTP 応答ヘッダに `Last-Modified` タイムスタンプが含まれる場合はその値を利用するため，1 日未満の更新間隔推定結果が得られることもある．

スされるのに対し，単位時間 1 時間の場合は毎時間アクセスするレコードは 10%未満であることを示している．前者はより単一周期のアクセスに近いといえるが，non-weighted が示すように，データベース全体と比較すると毎日アクセスされるレコードは 10%程度であり，識別子順のアクセスを行ったとしても，読み込んだ B-木ページの不要部分が多く，I/O 効率は低くなると考えられる．小さな単位時間に対しては更に識別子が疎となるため，ランダムアクセスとの差はほとんどなくなってしまう．一方，各レコードは単位時間内に一度しかアクセスされないため，バッファリングにより I/O を回避しようとすると，主記憶の利用効率が非常に低くなる．

このように，多周期的更新アクセスの課題としては，更新対象となる大量レコード取得の効率化が重要であり，B-木に代表される従来の検索構造は適さないと考えられる．同様に，LFS（Log-structured File System）[12] や Write-optimized B-tree [13] 等の書込みのシーケンシャル化による性能向上手法においても書込み後のデータの再読込みについては効率化されないため，多周期的更新アクセスの性能向上は望めない．

3. アクセス予定時刻順編成

3.1 基本構造とバッファ管理（SAT/Basic 方式）

前章の多周期的更新アクセスに適した二次記憶編成法として，図 3 に示すアクセス予定時刻順編成（基本方式：SAT/Basic）を提案する．ただし，図 3 では図 1 における情報源及び観測結果アーカイブを省略した．この構造は，イベントキューに状態データベースを統合し，非正規化したものととらえることができる．

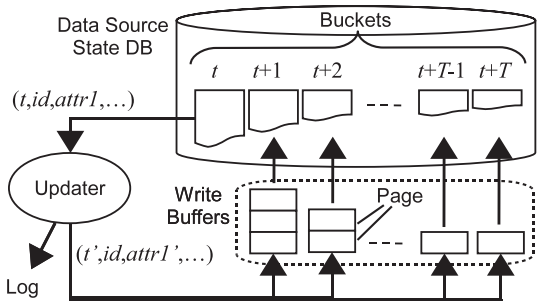

図 3 アクセス予定時刻順編成の基本構造（SAT/Basic）
Fig. 3 Scheduled Access Time Organization in its basic form (SAT/Basic).

本論文のモデルにおいては，更新プロセスが情報源の観測周期，すなわち各レコードのアクセス周期を把握しており，次にレコードがアクセスされる時刻は既知である．そこで，二次記憶を複数の領域（以下，バケットと呼ぶ）に分割し，それぞれに同一の次回アクセス予定時刻をもつレコードを格納するようにする．ただし，情報収集の要求として収集周期の下限 u と上限 $T \cdot u$（T は整数）が与えられているものとし，時刻を u を単位時間とする整数値で表現する．時刻 t においては，バケットは $t, t+1, t+2, ..., t+T$ のそれぞれに対応する $T+1$ 個が存在する．また，各バケットにはレコードを先頭から順に単純に格納し，そのサイズには制限を設けない．一方，主記憶上にはバケット $t+1, t+2, ..., t+T$ とそれぞれ 1 対 1 に対応する T 個の書込みバッファ（$t+1, t+2, ..., t+T$）を用意する．

以下，時刻 t における更新プロセスの動作概要を述べる．

（1） レコードポインタ r をバケット t の先頭に設定する．

（2） r が指すレコードを読み込み，属性集合（$attr1, ...$）からプロトコルパラメータ等を得て，識別子 id に対応する情報源に対しデータを要求する．

（3） 情報源からデータを取得し，アーカイブに追記する．

（4） 取得したデータに基づき，統計情報等を更新した属性集合の値（$attr1', ...$）を作成するとともに，次回アクセス予定時刻 $t'(t+1 \leq t' \leq t+T)$ を決定する．

（5） レコードの新たな値（$t', id, attr1', ...$）をクラッシュ回復用の更新ログ（Log）及び書込みバッファ t' に追記する．

（6） r を 1 レコード分進め，r がバケット t の末尾に達していなければ（2）に戻る．

（7） バケット t を削除し，その領域を解放する．

（8） 書込みバッファ$t+1$ のすべてのレコードをバケット $t+1$ の末尾に追記する．続いて，書込みバッファ$t+1$ を削除し，その領域を解放する．

（9） バケット $t+T+1$ 及び書込みバッファ$t+T+1$ を新たに作成する．

ステップ（2）においては，処理対象レコードを受動的に決定するため，識別子 id に基づくレコードの検索は不要であり，バケット内の複数レコードの先読みによる高速化が実現できる．

なお，実際の Web クローリングではステップ（3）～（5）を非同期的に実行し，複数 Web サーバへの並行アクセスにより通信スループットの向上を図っている．ここでは I/O 特性を明示するため Web サーバの応答遅延を無視し，逐次的な記述とした．

次に，ステップ（5）における書込みバッファの管理の詳細について説明する．書込みバッファは固定長のページ単位で拡張し，各ページの先頭アドレスを配列に格納して管理する．書込みバッファには先頭から順にレコードを格納して行き，空き領域がなくなったら新たなページを割り当ててレコードの残りの部分を格納する．書込みバッファに割り当てられるページ数は初期状態で 0 であり，上限は存在しない．そのため，書込みバッファごとに書き込まれるレコード数が均一でなくても主記憶を効率良く利用することができる．ただし，新たなページを割り当てる際に，各バッファに割り当てられたページ数の合計がしきい値を超えている場合は，書込みバッファのフラッシュを行う．

以下に，書込みバッファのフラッシュ動作の手順を示す．

（1） $p = 16, 4, 1$ の順に（2）～（4）を実行する．

（2） $i = 1, 2, ..., T$ の順に（3）を実行する．

（3） バッファ$t+i$ に割り当てられたページ数 $> p$ であればバッファ$t+i$ の全レコードをバケット $t+i$ の末尾に追記し，バッファ$t+i$ に割り当てられたすべてのページを解放する．

（4） 解放されたページが存在すれば終了．

（5） ページが割り当てられたバッファを一つ選択し，その全レコードを対応するバケットの末尾に追記し，当該ページを解放する．

ページ割当要求が発生しているバッファとは独立にすべてのバッファを順にフラッシュすることで，バケット間の切換を除き，ほとんどの書込みをシーケンシャルアクセスとすることが可能となる．ただし，フラッシュ対象バッファの最小ページ数 p を指定することで，小規模な書込みの抑制を図っている．また，ステップ（5）は空き領域の残っている単一ページからなるバッファをフラッシュせざるを得なくなった場合に実行されるが，バッファ数に対して割当可能なページ数が十分でない状況を示唆しており，小規模な書込みの多発を避けるため，フラッシュ対象を 1 ページに限定している．

アクセス予定時刻順編成においては，状態データベース自体がイベントキューを兼ねる．そのため，ディスク上のデータ構造は，メッセージングシステムにおけるキューデータベースと類似している [14]．しかし，メッセージングシステムではメッセージを遅延なくディスクに書き込むため，複数バッファによる書込み最適化の効果は得られない．メッセージングシステムは基本的に FIFO として動作し，時刻の絶対値に基づいてレコードを多数の書込み先に振り分ける状況を想定していない．

3.2 キーに基づくアクセスの考慮（SAT 方式）

SAT/Basic 方式においては，レコードを次回アクセス予定時刻に基づいて配置することで読込みを最適化するとともに，複数バッファを一括して順次フラッシュすることで書込みの最適化を行っている．一方，犠牲となっているのが更新プロセス以外からの要求によるアドホックなレコードアクセスである．SAT/Basic では，情報源識別子 id をキーとしてレコードを効率的にアクセスすることはできず，すべてのバケットを走査する必要がある．

連続的 Web クローリングにおいては，任意の Web ページの状態情報を取得するという要求が生じるのは主としてデバッグ用途であり，非効率性を許容するという選択肢も考え得る．一方，クローリング中に任意の Web ページの状態情報を更新するという要求は重要である．これには，手動でパラメータを変更するほか，フィードや sitemaps 等の別 URL に記述された更新頻度のヒント情報を自動的に反映することなどが含まれる．

そこで，SAT/Basic の特長を犠牲にすることなく，アドホックなレコードアクセスを可能にしたものが図 4 の SAT 方式である．データ構造として追加されたのは，シーケンス索引とバケット索引である．シーケンス索引は B-木等の検索構造を用い，各レコードの識別子 id に対応するシーケンス番号 $seq(= 0, 1, 2, ...)$ を格納する．シーケンス番号はレコードの状態データベースへの追加順を表す一意な整数値である．

レコードの状態データベースへの追加は，更新プロセスとは異なる追加プロセスが次のように実行する．まず，追加対象レコードの識別子 id によりシーケンス索引を検索し，id が存在する場合は当該レコードを棄却する．一方，識別子 id がシーケンス索引中に存在しなければ，新たなシーケンス番号 seq を採番し，(id, seq) の組をシーケンス索引に登録するとともに，初回アクセス時刻 $t+1$ と seq をレコードの属性に加えた値をバケット $t+1$（またはバッファ$t+1$）に追

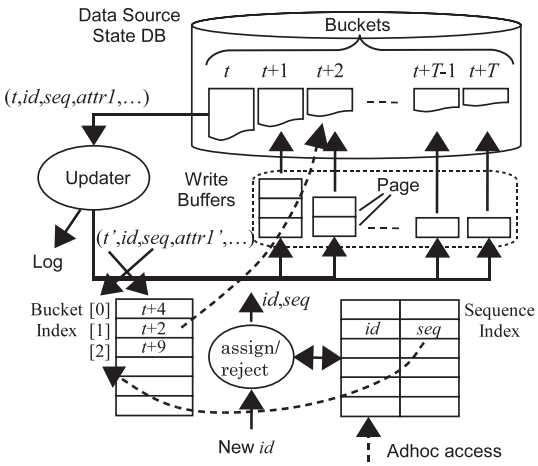

図 4　キー指定アドホックアクセスに対応したアクセス予定時刻順編成（SAT 方式）
Fig. 4　Scheduled Access Time Organization with key-based access support (SAT).

記する．

一方，バケット索引はレコードのシーケンス番号 $(0, 1, 2, ...)$ を添字とするバケット番号の配列であり，シーケンス番号に対応するレコードが格納されているバケットの特定を可能にする．バケット番号はバケットに対応する時刻と基準時刻の差を用いるなどしてなるべく小さな整数型で表現し，バケット索引全体が主記憶上に保持できるようにする．例えば，単位時間が 1 日であれば，16 ビット整数で 170 年以上表現可能であり，256 MiB のバケット索引で 1 億件以上のレコードに対応できる．単位時間が 1 時間の場合は，16 ビット整数で表現可能なのは 7 年とやや短いため，基準時刻の切上げを行いつつ使用するか，24 ビットへの拡張を行う．

アドホックなレコード更新の要求は以下のように処理される．

（1） シーケンス索引を検索し，レコードの識別子 id に対応するシーケンス番号 seq を得る．

（2） バケット索引により，シーケンス番号 seq に対応するバケット番号 b を得る．

（3） バケット b（またはバッファ b）にレコード更新要求メッセージを追記する．

すなわち，アドホックなレコード更新は即時には実行されず，更新プロセスによる周期的更新に合わせて非同期的に処理される．

SAT における更新プロセスの動作は，SAT/Basic に以下のステップを追加・変更したものとなる．

（0.5） バケット t のレコード及び更新要求メッセージを識別子順にソートする．

（2） r が指すレコードを同一識別子をもつ更新要求メッセージとともに読み込み，更新要求メッセージが存在する場合はその内容に基づいてレコードの属性集合を更新する．更に，属性集合からプロトコルパラメータ等を得て，識別子に対応する情報源に対しデータを要求する．

（5.5） レコードのシーケンス番号に対応するバケット索引のエントリに，t' に対応するバケット番号を格納する．

ステップ（0.5）及び（5.5）は，更新要求メッセージの有無にかかわらず発生する固定オーバヘッドである．ステップ（0.5）における外部ソートは，バケットに対するシーケンシャルアクセスが主体となるため，大きな性能低下にはつながらないと考えられる(注3)．一方，バケット索引は主記憶上に保持するため，ステップ（5.5）のオーバヘッドは問題にならない．なお，更新プロセスはシーケンス索引には一切アクセスしない．

ステップ（2）における更新要求メッセージの処理は，レコードのシーケンシャルな読込みと連動して効率的に実行することができ，また，実際のアドホック更新要求の発生頻度に対応しているため，全体に対する影響は無視できる．このように，SAT 方式ではアドホック更新要求を非同期的に処理することで，即時処理に必要となるバケットの走査及び書換えを避け，更新プロセスのバケットアクセスに干渉することを防ぐことができる．

次章の性能評価における比較のため，SAT 方式における書込みバッファの管理を次のように単純化したものを SAT/Naive 方式とする．すなわち，各バッファに割当可能なページ数は最大 1 ページとし，ページの空き領域がなくなった際には当該ページの内容を対応するバケットに追記した後にページを再利用する（バッファごとに独立した固定長バッファリング）．また，割当可能なページ数の合計がバッファ数（T）よりも小さい場合，空のバッファへのページ割当が行えないことがあり得る．この場合，**3.1** のステップ（5）と同

(注3)：バケットのソートを避けるため，更新要求メッセージをレコード用のバケットと対をなす専用バケットに格納し，レコード用バケットの読込みに先立って更新要求メッセージを主記憶上のハッシュテーブルにロードするという実装も可能であるが，ここではあえて単純な実装を用いることにする．

様，ページが割り当てられたバッファを一つ選択し，その内容をバケットに追記して当該ページを解放する．

4. 性能評価

4.1 評価環境

アクセス予定時刻順編成の多周期的更新アクセスに対する効果を実証するため，連続的 Web クローリングにおける状態データベースの実データに基づく性能評価を行った．実データは **2.** の図 2 に用いたものと同じであり，64 ノードによる並列クローリングの 1 ノード分のうち，1 回以上正常にアクセスできた 17,496,056 URL に対応する状態レコードである．状態レコードからは URL の文字列長，次回アクセス予定時刻の初期値，及び更新間隔推定値（アクセス間隔）の属性を利用した．各 URL の文字列長は 11 バイトから 3715 バイトまで広く分布しており，平均は 63.7 バイトであった．状態レコードのレコード長は個々の URL 文字列長に固定長（50，100，200，または 400 のいずれか）を加えたものとした．なお，URL に対応するコンテンツサイズの平均値は 22.0 KiB であった．

状態データベースの更新性能のみを評価するため，更新プロセスの動作は極力簡略化した．すなわち，読み込んだ属性値をそのまま更新後の値とし，アクセス間隔の値は定数として扱った．また，コンテンツの取得とアーカイブへの格納も省略した．

表 2 に性能評価を行った環境の仕様を示す．SAT（及びその変種）の各バケットは JFS 上のファイルとして実装し，ファイル名を時刻に対応させた．また，比較対象は URL のフィンガープリント（16 バイト）をキーとする B-木とした．B-木の実装には Oracle Berkeley DB 4.6.21 を利用し，トランザクションサポートや並行アクセス制御のない最も軽量なモード（Berkeley DB Data Store）を用いた [15]．なお，B-木は状態データベースのみに用い，イベントキューは時刻ごとの更新対象 URL を列挙した複数のファイルとした．イベントキューの管理オーバヘッドの影響を排除するため，これらのファイルは動的には更新せず，前もって実行した際の内容をあらかじめ格納した．

図 5 にレコードの固定長部分を変化させた際の状態データベース全体のサイズ（ディスク使用量）の変化を示す．SAT は単位時間 1 日（Daily）と 1 時間（Hourly）でバケット数 T が異なり，前者が 400，後者が 9600（$= 400 \times 24$）であるが，ディスク使用量にはほとんど差は見られない．B-木（B-tree）については異なるページサイズの結果を示した．4 KiB のページを用いたときのオーバヘッドが大きい．なお，B-木にレコードを挿入する際は，あらかじめレコードをキー順にソートして与えた．Berkeley DB では，キー順の追加に対しては，分割後のページの領域利用率が低下しないよう工夫されている．上記のいずれも評価環境の主記憶に全体をキャッシュするには大きすぎるサイズとなっている．

4.2 更新処理コストの時間推移

図 6 は単位時間を 1 時間としたときの各単位時間における 1 レコード当りの更新処理時間の初期状態からの推移を示したものである．単位時間当りの更新レコード数は平均 13.8 万件であり，そのサイズは 21.5 MiB となる（レコードの固定長部分を 100 バイトとした）．なお，測定に先立っては 3 GiB 程度のファ

表 2　性能評価環境の仕様
Table 2　Specification of performance evaluation environment.

CPU	Intel Core Duo T2500 2 GHz
Memory	2 GiB
HDD	ST3500630NS x2
Interface	SATA 3.0 Gb/s
Capacity	500 GB
Max. sustained transfer rate	72 MB/s
Cache	16 MB
Average latency	4.16 msec
Spindle speed	7,200 rpm
Random read seek time	$<$ 8.5 msec
Random write seek time	$<$ 10.0 msec
OS	Red Hat Enterprise Linux 3 (kernel 2.4.21)
File System	JFS

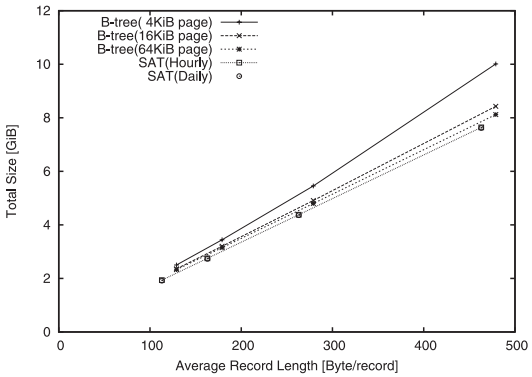

図 5　レコード長とデータベースサイズの関係
Fig. 5　Database size vs. record length.

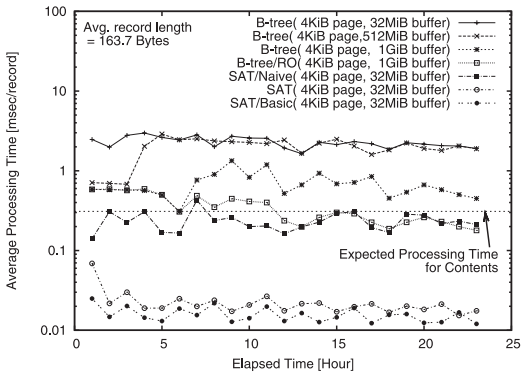

図 6　更新処理時間のトレース（単位時間 1 時間）
Fig. 6　Trace of update processing times (hourly).

イルを空読みし，OS バッファがコールド状態となるようにした．

SAT ファミリについてはバッファのページサイズ（割当単位）を 4 KiB とし，割当可能ページの総容量を 32 MiB とした．SAT ファミリにおける更新ログは状態データベースとは異なるディスクに出力した．また，SAT 及び SAT/Naive におけるバケットのソートでは，1 MiB の主記憶バッファを利用可能とした．ただし，実際のアドホック更新は発生しないものとした．更に 16 ビットのバケット番号を 1 億件程度登録して 256 MiB に成長させたバケット索引を与え，その更新も行った．バケット索引はメモリマップト I/O によりアクセスし，処理に先立って全体を走査し，主記憶にキャッシュされるようにした．第 1 時間目はこれらのオーバヘッドを含んだ値となっている．

B-tree/RO で示したものは指定された識別子をソートして順にレコードを取り出した場合（読込みのみ）の結果である．時間とともに処理時間が小さくなる傾向があるが，Berkeley DB が備えるバッファ（キャッシュ）の効果と考えられる．ここではバッファサイズを 1 GiB と大きくしたが，他の値でもほとんど差異は見られなかった．一方，更新を伴う B-tree では，バッファサイズの影響が大きい．512 MiB 以上バッファを与えた場合には，開始直後は読込みのみと同様の挙動を示し，ある時点で更新処理時間が大きく増加する．この時点でバッファの書戻しが発生し始めたものと考えられる．図 6 に水平線で示したのは，コンテンツの平均サイズである 22.0 KiB と評価環境のディスクの最大シーケンシャルアクセス性能である 72 MByte/s から求めた，コンテンツ書込み処理時間の期待値 0.31 ms である．B-木で実装した状態データベースは，明らかに支配的なボトルネックとなっている．

SAT はデータの更新を行っているにもかかわらず，読込みのみの B-tree/RO と比較しても 1 けた程度高速となっている．書込みを行う B-tree との差は 2 けたに達することもある．一方，書込みバッファの制御を単純化した SAT/Naive は SAT より 1 けた遅くなっている．これは，レコードの読込みをシーケンシャルに行うだけでは不十分であり，書込みバッファ全体を一括して制御し，書込みにおけるランダムアクセスも排除することが不可欠であることを示している．

また，アドホックなレコード更新を考慮しない SAT/Basic は SAT より高速であるが，その差は 10 μs であり，コンテンツ書込み処理に隠ぺいされるため実用上問題ないといえる．ここでは，実際のアドホック更新要求は発生しないものとしたが，仮にすべてのレコードに対するアドホック更新要求が一斉に発生したという極端な仮定を置いたとしても，各バケットのサイズが約 2 倍になり，更新プロセスの処理時間が 2 倍の約 40 μs になるだけであり，B-tree やコンテンツ書込みとの逆転は起こらない．実際には，アドホック更新を要求するプロセスのシーケンス索引に対する検索処理がボトルネックとなるが，アドホック更新の発生モデルや SAT における処理性能の限界については稿を改めて述べることとしたい．

4.3　パラメータの影響

図 7 及び図 8 はバッファサイズ（総容量）を変化させた際の 1 レコード当りの更新処理時間を，それぞれ 1 日及び 1 時間の時間単位について示したものである．単位時間 1 日については 5 単位時間分，単位時間 1 時間については 36 単位時間分の更新処理（それぞれ 8,713,026 件，5,478,975 件）を行った際の平均値を用いた．ページ単位でバケットへの書込みを行う SAT/Naive では，ページサイズが小さい 4 KiB の場合の処理時間の増加が顕著である．ページサイズを 64 KiB に増加させると書込みの断片化が緩和され，性能が大幅に向上しているが，SAT との明確な差は残っている．SAT ではバッファ総容量が大きい場合にはページサイズによる差異がなく，バッファ総容量が限られる領域では逆に 4 KiB ページの方が処理時間が小さくなっている．これは，SAT では書込みサイズとページサイズが独立であり，主記憶をより効率的に利用できる小ページが有利に働くためである．SAT とアドホック更新を考慮しない SAT/Basic の差は図 6

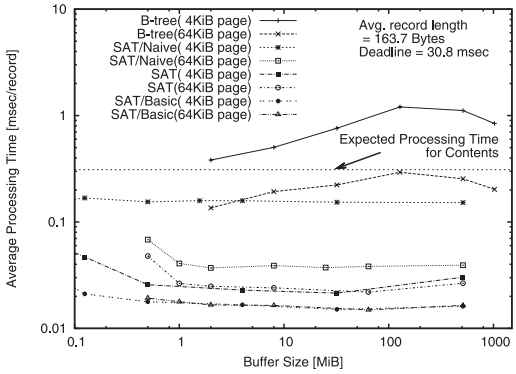

図 7　バッファサイズに対する更新処理時間の変化（単位時間 = 1 日）
Fig. 7　Update processing time vs. buffer size (daily).

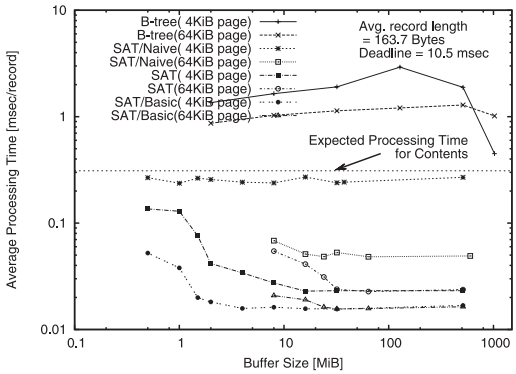

図 8　バッファサイズに対する更新処理時間の変化（単位時間 = 1 時間）
Fig. 8　Update processing time vs. buffer size (hourly).

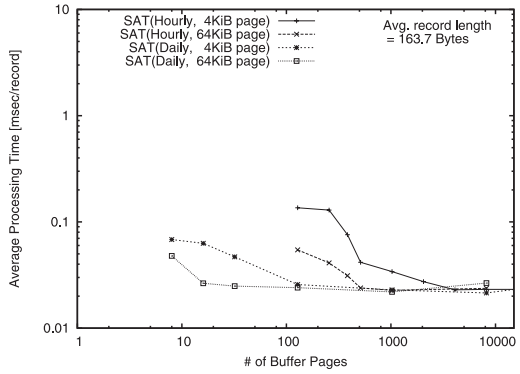

図 9　バッファページ数に対する更新処理時間の変化
Fig. 9　Update processing time vs. buffer size (in terms of #pages).

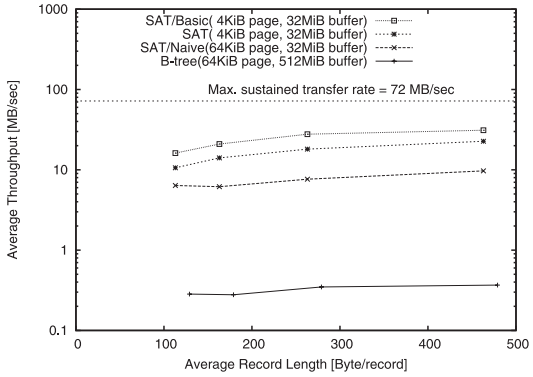

図 10　レコード長に対するスループットの変化（単位時間 = 1 時間）
Fig. 10　Update throughput vs. record length (hourly).

と同様，ほぼ 10 μs である．また，グラフ中に記した Deadline の値は，時間単位内に対象レコードを更新し終えるのに必要な 1 レコード当り更新処理時間の上限である．B-tree においては，1 時間単位の処理を行っている図 8 で，あまり余裕がないことが分かる．

特に，時間単位 1 時間に対応する図 8 でバッファサイズが限られている場合に，SAT の処理時間の増加が見られる．これは，横軸をページ数とした図 9 から明らかなようにページの枯渇が原因と考えられる．時間単位 1 日においてはページ数 20 以下，時間単位 1 時間においてはページ数 500 以下で性能低下が顕著となっている．これらはそれぞれにおけるバケット数，400 及び 9600 の約 1/20 に相当する．バケット数を目安にページを確保すれば十分と考えられるが，単位時間を更に小さくして観測精度を上げようとするとこれは問題となる．例えば，1 分単位で最大 400 日間のスケジューリングを行う際にはバケット数は 576,000 となり，4 KiB ページに対しては 2 GiB を超えてしまう．このような場合，一様な時間間隔でバケット分割するのではなく，例えば 1 日以上の周期に対しては 1 日単位とするなど，混合基数による分割を行えばバッファサイズの爆発を防ぐことが可能と考えられる．本論文では SAT の従来手法に対する優位性を示すことを主眼としているため，時間単位，最大アクセス間隔，及びアクセス間隔の分布等を変化させた際の SAT の詳細評価については稿を改めて述べることとしたい．

図 10 はレコード長を変化させた際の挙動をスループットとして表したものである（各レコードを読込み及び書込みの 2 回転送していることを考慮し，平均レコード長 × 2/平均更新処理時間により求めた）．SAT では 10〜20 MByte/s が得られている．この値

はディスクデバイスの最大転送速度と比較すると 1/4 程度であるが，B-tree では 0.3 MByte/s にとどまっており，極めて低い性能しか得られていない．

以上から，多周期的更新アクセスに対する B-木の性能は，バッファ容量の増加，レコードの事前ソート等のチューニングを行っても非常に低く，深刻なボトルネックになるといえる．これに対して，アクセス予定時刻順編成では大量の更新処理をシーケンシャルアクセスに近い性能で実施でき，B-木と比較して 10〜100 倍程度の劇的な効果が得られた．

5. むすび

本論文では，大量レコードに対する多周期的な更新を大幅に効率化するための二次記憶編成法について提案し，その有効性を示した．このようなアクセスパターンは，特に大規模な連続的 Web クローリングにおいて問題となる．従来型の検索構造では，単一レコードのアドホックなアクセスを重視しているために，多周期的更新アクセスにおいてはアクセス対象レコードの sparse 性により，各レコードをランダムアクセスの性能で処理することを余儀なくされる．

本論文で提案したアクセス予定時刻順編成では，レコードの次回アクセス予定時刻に関する知識を用いて周期的アクセス時に最適となるように配置し，キーに基づく検索を二次的な機能とすることで上記の問題を解決した．実際に十億規模の Web ページを対象とする連続 Web クローリングにより生成されたデータベースを用いた評価により，提案手法がシーケンシャルアクセスに近い性能を引き出すことに成功し，B-木の 10〜100 倍という大きな優位性が得られることを示した．

本方式は，Web クローリングをはじめ，多周期的な観測にかかわる応用全般に対して有効であると考えられる．今後は，本方式が犠牲にしているアドホックアクセスの性能や，異なる時間分解能やアクセス周期の分布における効果への影響等について評価を行っていきたい．

文　献

[1] 喜連川優, 豊田正史, 田村孝之, 鍛治伸裕, 今村　誠, 高山泰博, 藤原聡子, “Socio Sense：過去 9 年に及ぶ Web アーカイブから社会の動きを読む,” 情報処理, vol.49, no.11, pp.1290–1296, 2008.

[2] M. Kitsuregawa, T. Tamura, M. Toyoda, and N. Kaji, “Socio-Sense: A system for analysing the societal behavior from long term web archive,” APWeb, pp.1–8, 2008.

[3] 田村孝之，喜連川優，“大規模 Web アーカイブのための更新クローラの設計と実装,” 日本データベース学会 Letters, vol.6, no.1, pp.173–176, 2007.

[4] 田村孝之，喜連川優，“大規模 Web アーカイブ更新クローラにおけるスケジューリング手法の評価,” 信学論（D）, vol.J91-D, no.3, pp.551–559, March 2008.

[5] S. Brin and L. Page, “The anatomy of a large-scale hypertextual Web search engine,” Proc. WWW7, pp.107–117, 1998.

[6] sitemaps.org, “Sitemaps XML format.” http://www.sitemaps.org/protocol.html

[7] U. Schonfeld and N. Shivakumar, “Sitemaps: Above and beyond the crawl of duty,” Proc. WWW ’09, pp.991–1000, 2009.

[8] B.E. Brewington and G. Cybenko, “How dynamic is the Web?,” Proc. WWW9, pp.257–276, 2000.

[9] J. Cho and H. Garcia-Molina, “The evolution of the web and implications for an incremental crawler,” Proc. VLDB 2000, pp.200–209, 2000.

[10] J. Cho and H. Garcia-Molina, “Estimating frequency of change,” ACM Trans. Internet Technology, vol.3, no.3, pp.256–290, Aug. 2003.

[11] C. Olston and S. Pandey, “Recrawl scheduling based on information longevity,” Proc. WWW ’08, pp.437–446, 2008.

[12] M. Rosenblum and J.K. Ousterhout, “The design and implementation of a log-structured file system,” ACM Trans. Comput. Syst., vol.10, no.1, pp.26–52, 1992.

[13] G. Graefe, “Write-optimized b-trees,” Proc. VLDB ’04, pp.672–683, 2004.

[14] Q. Wei, X. Lu, L. Ren, and X. Zhou, “A novel disk queue to reduce disk I/O of messaging system,” SIGOPS Oper. Syst. Rev., vol.37, no.3, pp.55–60, 2003.

[15] Oracle Corp., “Oracle Berkeley DB.” http://www.oracle.com/technology/products/berkeley-db/db/index.html

（平成 21 年 9 月 11 日受付，22 年 1 月 5 日再受付）

田村　孝之　（正員）

1991 東大・工・電子卒．1996 同大大学院工学系研究科博士課程単位取得退学，NEDO 最先端分野技術研究員等を経て，1998 三菱電機（株）入社．博士（工学）．現在，同社情報技術総合研究所専任並びに東大生産技術研究所共同研究員．並列データベース処理，Web クローリング・Web マイニングに関する研究に従事．情報処理学会，日本データベース学会，ACM，IEEE Computer Society 各会員．

喜連川　優　（正員：フェロー）

1978 東大・工・電子卒．1983 同大大学院工学系研究科情報工学博士課程了．工博．同年同大生産技術研究所講師．現在，同教授．同所戦略情報融合国際研究センター長．データベース工学，並列処理，Web マイニングに関する研究に従事．2009 ACM SIGMOD Edgar F. Codd Innovations Award 受賞．情報処理学会副会長，日本データベース学会理事，SNIA-Japan 顧問．本会データ工学研究専門委員会委員長（1997〜1998），ACM SIGMOD Japan Chapter Chair（1999〜2002）歴任．VLDB Trustee（1997〜2002），IEEE ICDE，PAKDD，WAIM 等ステアリング委員，IEEE ICDE Program Co-chair（1999），General Co-chair（2005）．