情報処理学会論文誌　データベース　Vol. 4　No. 1　40–49 (Mar. 2011)

多周期的 Web クローリングにおける時間分解能向上手法

田村孝之[†1,†2]　喜連川優[†2]

Web 情報を継続して収集する際は，個々の Web 情報に適した周期で収集を繰り返す多周期的 Web クローリングが求められるが，Web 情報ごとに過去の状態を管理するデータベースへのランダムアクセスがボトルネックとなり，実現可能な最小収集周期が制限されてしまう．筆者らはすでに，データベースレコードを次回更新予定時刻に基づいて分割格納する編成方式を提案し，これによりシーケンシャルアクセスに近い性能が得られることを示した．本稿では，本方式における時刻分割の分解能を可変にすることで，最大収集周期を 1 年とした場合，従来 1 時間程度が限界であった最小収集周期を 1 分程度にまで短縮する手法について述べ，実機上での評価に基づき，その有効性を示す．

A Method for Improving Time Resolution in Multi-periodic Web Crawling

Takayuki Tamura[†1,†2] and Masaru Kitsuregawa[†2]

For capturing temporal changes of the Web continuously, the authors have developed a multi-periodic Web crawler which revisits Web pages with individual intervals. The multi-periodic Web crawler was found to suffer from the overhead of updating the per-page state database on each download. As a remedy for the problem, we have proposed a novel scheme for secondary storage organization, which is suitable for periodically updating database records with independent intervals. This paper proposes a method for further improving the storage organization to achieve even higher time resolutions, such as one minute.

†1 三菱電機株式会社情報技術総合研究所
Information Technology R&D Center, Mitsubishi Electric Corporation
†2 東京大学生産技術研究所
Institute of Industrial Science, The University of Tokyo

1. まえがき

筆者らは Web 情報の分析基盤として日本の Web 情報を網羅的に収集・蓄積した大規模 Web アーカイブの構築を進めている[1),2)]．Web 空間全体を一括収集するクローリングでは 1 カ月程度の時間分解能が限界となることから，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多周期的（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)]．

多周期的 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することで，HTTP における条件付き GET 要求の発行や，最適な Web ページアクセス周期の導出を可能にするものである．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全体で数百 GB 規模に達し，その管理コストは軽微ではない．むしろ，見かけ上ランダムな Read-Modify-Write アクセスにより，収集コンテンツの格納に要する時間よりも多くの待ち時間を発生させる原因となる．

大規模データベースに対する多周期的更新の高速化においては，従来のキャッシュやログ構造化などの手法は有効でないため，筆者らはアクセス予定時刻順編成によるレコード格納方式を提案し，その有効性を実証した[5)]．本方式では，各レコードの更新周期に関する知識を利用し，次の読み込み時期に応じてレコードをクラスタリングする．また，書き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．しかしながら，本方式では最大更新周期と最小更新周期の比に応じた数の書き込みバッファを主記憶に用意する必要があり，更新周期の範囲を大きくとれないという課題があった．

本稿では，アクセス予定時刻順編成におけるレコードのクラスタリングにおいて，時刻の分解能を更新周期に応じて可変とし，長周期側のレコードを少数の書き込みバッファに集約することで広範な更新周期を実現する手法について提案する．これにより，ほとんど変化のない静的 Web ページに対する年オーダ周期の収集と，マイクロブログなどの高頻度に変化する情報の分オーダによる収集とを統一的に扱うことが可能になる．

以下，2 章では多周期的 Web クローリングの課題と従来の解決手法の限界を明らかにする．3 章では提案手法について説明し，4 章で実データの特性を反映したデータセットを用いて実機上で性能評価を行う．5 章で関連研究について概観し，6 章で全体のまとめを行う．

2. 多周期的 Web クローリングの課題

2.1 状態情報管理にともなうオーバヘッド

多周期的 Web クローリングの動作は，**図 1** のようにモデル化することができる．すなわち，クローラプロセス（Crawler）がイベントキューから現在時刻 t に収集予定となっている Web ページの URL を取り出し，当該 URL にアクセスして取得したコンテンツをディスク上のリポジトリ（Contents Repository）に格納するとともに，当該 URL に固有の収集周期に基づいて次回収集時刻（t'）を決定し，URL とともに再度イベントキューに投入する，というものである．クローラプロセスはさらに，各 Web ページ固有の管理情報を格納したディスク上の状態データベース（Per-page State DB）の参照・更新を行い，収集の効率化を図る．すなわち，前回収集時のタイムスタンプやタグ情報（HTTP 応答ヘッダに含まれる Last-modified および ETag フィールドの値）を Web サーバに送信し，未更新情報の再収集を避ける HTTP 条件付き GET 要求[6] の発行や，更新有無の履歴に基づいて推定した平均更新周期を Web ページの収集周期に反映することなどを行う[4),7)]．収集に先立って URL に対応する状態情報（*state*）が読み出され，収集後は取得した情報に基づいて更新された新たな状態（*state'*）が書き戻される．

多周期的クローリングにおける状態データベースのアクセスタイミングは，イベントキューの内容に従うためクローラ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一定せず，実質上ランダムアクセスとなる．状態情報は数百バイト程度であり，平均 20 KB 以上ある Web ページコンテンツと比較すると 2 桁小さいが，コンテンツはシーケンシャルアクセスによりレポジトリに追記できるため，データ量だけからはどちらの I/O が支配的となるか判断できない．**図 2** は，実際の Web クローリングにおける，Web サーバとの通信（Communication），状態情報の読み書き（State info），およびコンテンツのレポジトリへの書き込み（Contents），の各フェーズの平均的な所要時間を示したものである．ただし，状態データベースは URL をキーとする B-tree として単一ディスク装置に格納するものとした．最も多くの時間を要しているのは Web サーバとの通信であるが，大規模 Web クローリングでは多数の Web サーバと並行して通信することにより通信遅延を隠蔽し，実効的な所要時間を大幅に低減させることが可能である．一方，状態情報の読み書き時間はコンテンツの書き込み時間より 1 桁大きく，ディスク I/O のボトルネックは状態情報に関するものであることが分かる．状態情報の I/O 多重度を上げるには並列に駆動するディスク装置の台数を増加させる必要があり，H/W コストの上昇が避けられない．大規模なクローリングにはある程度の H/W 量が要求されるものの[⋆1]，収集規模（コンテンツ書き込み速度）に見合うものであることが望ましい．そこで本稿では，ディスク装置台数を固定した場合の状態情報アクセスの性能向上に議論を絞ることにする．

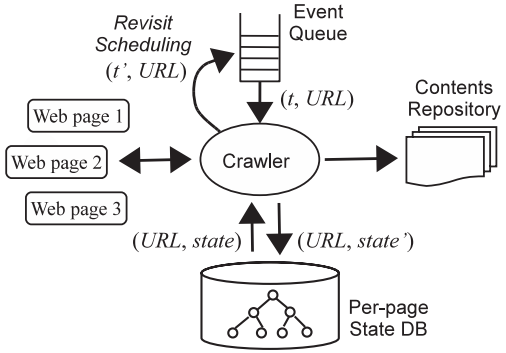


図 1　従来型の多周期的 Web クローラモデル
Fig. 1　A conventional model of multi-periodic Web crawler.

2.2 アクセス予定時刻順編成（SAT）によるランダムアクセス回避

多周期的データベースアクセスに適した二次記憶編成法として，筆者らは**図 3** に示すアクセス予定時刻順編成（SAT）を提案している[5)]．更新プロセス（Updater）が図 1 の Crawler に相当し，Web ページコンテンツの取得およびレポジトリへの格納は省略している．二次記憶上の構造は，イベントキューに状態データベースを統合（キーである URL を介して結

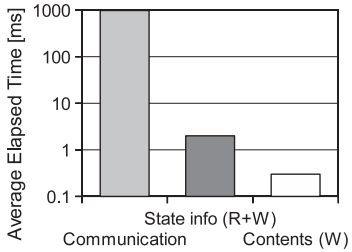


図 2　Web クローリングのフェーズ別平均所要時間
Fig. 2　Average elapsed times of each phase in Web crawling.

⋆1 筆者らは 64 台の PC サーバによる並列収集を実施している．

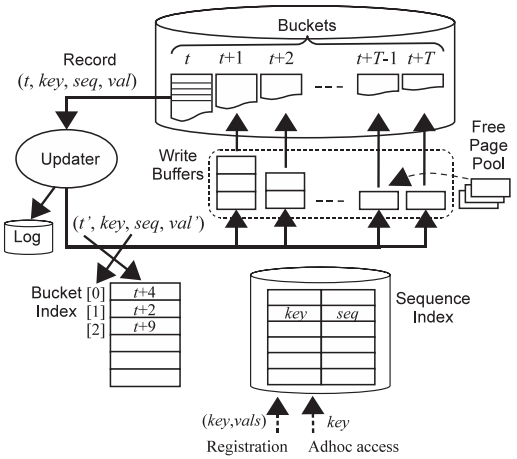


図 3　アクセス予定時刻順編成（SAT）の概観
Fig. 3　Overview of Scheduled Access Time organization (SAT).

合し，非正規化）したものととらえることができる．すなわち，状態レコードがアクセスされるタイミングは，当該レコードのキー（URL，図 3 では key）がイベントキューから取り出されるタイミングに等しいため，イベントキューのエントリに状態情報（図 3 では val）を格納し，キーによる両者のジョイン操作を不要としたものである．

イベントキューからの読み出し操作は先頭要素（スケジュール時刻 t が最小のもの）に対して行われるため，二次記憶上でエントリ（レコード）をスケジュール時刻順に配置しておくことで，ランダムアクセスを回避することができる．一方，イベントキューへの書き込み操作は任意の位置が対象となるため，挿入位置を決定する際に二次記憶へのランダムアクセスが発生しやすい．SAT では，時刻の分解能を比較的粗くしておき，レコードの挿入位置を少数にとどめ，書き込みの効率化を図っている．すなわち，スケジュール間隔の下限 u（時間分解能）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 を単位とする整数値で表現した場合に，レコードをスケジュール時刻に固有の書き込みバッファ（T 個）に振り分けて主記憶上にいったん保持する．書き込みバッファはそれぞれ必要に応じて空きページプール（Free Page Pool）から主記憶ページを割り当て，独立に拡張されるが，空きページプールが空になった時点で書き込みバッファと 1 対 1 に対応する二次記憶上の領域（バケット，ファイルとして実装）への書き込み（フラッシュ）が行われる．書き込みバッファを可変容量とすることで，スケジュール時刻のばらつきを吸収し，主記憶を効率良く利用することができる．また，各書き込みバッファを時刻順にフラッシュするとともに，小さな書き込みバッファは最初はフラッシュ対象外とするなどの最適化により，ランダムアクセスの影響を可能な限り排除している．

図 3 のシーケンス索引（Sequence Index）およびバケット索引（Bucket Index）は，イベントキューが備えていない，キーに基づくレコードの読み出しおよび更新を実現するための二次索引である．任意の値をとりうるキーとレコードの格納位置を直接対応付けることは，元の状態データベースと同様のオーバヘッドをもたらすため，レコードを新規登録する際に割り当てる密なシーケンス番号（seq）を介し，Updater が更新するバケット索引はコンパクトな配列として主記憶上に保持できるようにしている（格納位置の精度もバイト単位ではなく，バケット単位にとどめている）．

2.3　SAT 方式の課題

SAT では，書き込みバッファあたりのレコード数を増加させることにより，二次記憶に対するシーケンシャル書き込みを実現する．したがって，バケット数 T を大きくとると，書き込みバッファが細分化して十分な効果が得られないことになる．

Web 上ではほとんど更新されない Web ページが多数存在し，Web ページ収集周期の上限は 1 年程度であることが望ましい．一方，Twitter に代表されるマイクロブログのように，途切れることなく情報が提供される Web サービスも存在する．Web サーバへのアクセス負荷に配慮すると[8]，1 分程度の収集周期が必要となる．スケジュール間隔の範囲を 1 分〜400 日とすると $T = 576{,}000$ となり，書き込みバッファが細分化されるのは明らかである．

スケジュール間隔の範囲を分割してそれぞれ専用のクローラを用いるという方法もあるが，Web ページの収集間隔は Web ページの更新有無に応じて変化させるため，過渡的な状態における管理作業が繁雑になり，望ましくない．次章では，単一の機構により広範なスケジュール間隔に対応するための手法について述べる．

3.　可変時間分解能によるアクセス予定時刻順編成（SAT/L）

SAT においてスケジュール間隔の範囲拡大とともにバケット数が線型に増大するのは，すべての時刻に対して同一の時間分解能を適用しているためである．しかしながら，直近の時刻に対しては高い時間分解能（小さな時間単位）を，遠い将来の時刻に対しては低い時間分解能（大きな時間単位）を適用する方が直感的である．そこで，**図 4** の SAT/L 方式では，現在時刻から遠い領域では隣接する複数の書き込みバッファをまとめ，書き込みバッ

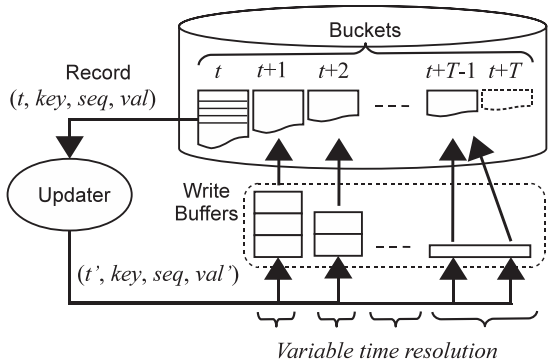


図 4　可変時間分解能によるアクセス予定時刻順編成（SAT/L 方式）
Fig. 4　Scheduled Access Time organization with variable time resolution (SAT/L).

表 1　SAT および SAT/L における使用バッファ数の比較
Table 1　Comparison of number of buffers used in SAT and SAT/L.

Min. interval	Max. interval	SAT	SAT/L
1 day	400 days	400	90
1 hour	400 days	9,600	173
1 minute	400 days	57,600	275

ファ数を削減する（Free Page Pool，Log，Sequence Index，および Bucket Index は図 3 と共通であり，図 4 では省略した）．

SAT/L では，時刻 t において時刻 t' にスケジュールされるレコードが書き込まれる書き込みバッファ b を以下のように決定する．

$$\tau = \left\lfloor \exp(\left\lfloor \log(t'-t) \right\rfloor) \cdot s \right\rfloor \tag{1}$$

$$b = \left\lfloor \frac{t'}{\tau} \right\rfloor \tau \tag{2}$$

τ はスケジュール間隔 $t'-t$ に応じて定まる時間分解能であり，$t'-t$ とともに階段状に増加する．s は τ と $t'-t$ の比を与えるパラメータであり，$s=1/12$ としている（1 年後のイベントを 1 カ月の精度で管理）．書き込みバッファ b はスケジュール時刻 t' を τ で丸めた値である．対数をとっているため，$t'-t$ が大きくなるほど τ の変化は緩やかになり，より多くの値が同一の τ に対応するようになる．その結果，同一の b を与える t' の範囲も広がり，使用されるバッファ数は $t'-t$ に対して対数的に増加することになる．**表 1** に SAT/L における使用バッファ数削減効果を示す．なお，400 日付近における時間分解能 τ の値は 23 日である．

現在時刻が進むとともに書き込みバッファに対応する時間分解能 τ が減少し，それまでスキップされていた書き込みバッファも書き込み対象となる．それにともなってバケットも新たに作成される．たとえば，図 4 のバケット $t+T$ は時刻 t においては欠番であるが，時刻 $t+T-1$ にスケジュール間隔 1 でスケジュールされるレコードなどが書き込まれ，時刻 $t+T$ における処理対象となる．なお，現在時刻 t を 1 単位進める場合，SAT では書き込みバッファ $t+1$ を削除し，書き込みバッファ $t+T+1$ を新設するだけでよいが，SAT/L ではバッファの要否が不規則に変化するため，すべてのバッファをチェックする必要がある．

SAT/L は遠い将来の時刻に対しては低い時間分解能が適用可能という仮定に依存しているが，この仮定がつねに妥当であるとは限らない．本来のスケジュール時刻 t' に対し，$b \le t'$ であるため，スケジュール時刻の精度が問題となる場合は，現在時刻 b においてバケット b を処理する際に，$t' > b+e$（e は許容誤差）となるレコードを再度時刻 t' にスケジュールすればよい．再スケジュールにおいても書き込みバッファ t' 以外に書き込まれることはありうるが，$t'-b < t'-t$（t は当該レコードが最初にスケジュールされた時刻）であるから時間分解能 τ は次第に減少し（s と同程度の比で減少），1（誤差なし）に収束する．

次に，SAT/L におけるバケットサイズ（レコード数）について考察する．まず，現在時刻を t とし，SAT においてバケット t の処理が完了した直後の状態を考える．i を整数とし，スケジュール間隔 i（$1 \le i \le T$）を持つレコード数が $f(i)$ で与えられるとすると，当該レコードのスケジューリングが一様に発生する場合，当該レコードはバケット $t+1$ からバケット $t+i$ まで均一に分布することになる．したがって，バケット $t+j$（$1 \le j \le T$）に含まれるスケジュール間隔 i のレコード数 $g(i,t+j)$ は，

$$g(i,t+j) = \begin{cases} f(i)/i & \cdots \quad i \ge j \\ 0 & \cdots \quad \text{otherwise.} \end{cases} \tag{3}$$

で与えられ，バケット $t+j$ に含まれる総レコード数 $h(t+j)$ は，

$$h(t+j) = \sum_{i=j}^{T} f(i)/i \tag{4}$$

となる．

一方，SAT/L においては，レコードがバケットに局在しており，周期的にレコード数の大

小が生じる．時刻 $t+1$ でスケジュール間隔 i ごとの周期的な変化の位相が揃うと仮定すると，SAT/L においてバケット $t+j$ に含まれるスケジュール間隔 i のレコード数 $g'(i,t+j)$ は，

$$g'(i,t+j) = \begin{cases} f(i)/\lfloor i/\tau(i) \rfloor & \cdots \quad i \geq j \text{ かつ } (j-1) \bmod \tau(i) = 0 \\ 0 & \cdots \quad \text{otherwise.} \end{cases} \tag{5}$$

で表される．ただし，$\tau(i)$ はスケジュール間隔 i に対応する時間分解能である．特に，バケット $t+1$ のレコード数 $h'(t+1)$ は，

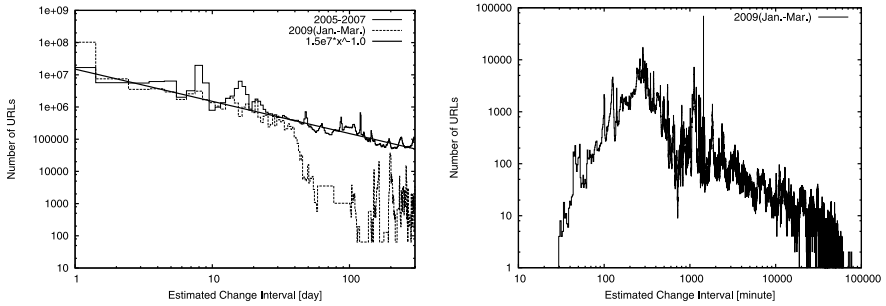
$$h'(t+1) = \sum_{i=1}^{T} f(i)/\lfloor i/\tau(i) \rfloor \tag{6}$$

となる．この値は $h(t+1)$ より大きくなっており，バケットの集約が重なった最悪ケースでのバケットサイズを表している．

4. 性能評価

4.1 評価用データセット

提案する SAT/L 方式を，実クローリングを模したデータセットを用いて実機上で評価した．**図 5** は，2005 年から 2007 年にかけての約 2 年間と，2009 年 1 月から 3 月にかけての約 3 カ月間に繰り返し収集した Web ページの更新有無の履歴から推定した Web ページ更新間隔の分布である（ただし，アクセス回数が 3 回未満の Web ページは除いた）．図 5 (a) は



(a) 時間分解能 = 1 日　　(b) 時間分解能 = 1 分

図 5　大規模 Web クローリングに基づく Web ページ更新間隔推定値の分布
Fig. 5　Distribution of estimated change intervals of Web pages based on runs of large scale Web crawling.

時間の分解能を収集の最短周期と同じ 1 日とした場合の結果であり，更新間隔ごとの Web ページ数が Zipf 分布[9]（$y = 1.5 \times 10^7 \cdot x^{-1}$ で示した両対数グラフ上の直線）に従っていることが分かる．2009 年のデータは期間が短いために，長周期側の情報が不足しているが，3 カ月以内の分布に関しては長期間実施した結果と同様の傾向を示している．

図 5 (b) は 2009 年のデータについて，時間分解能を 1 分として導出した更新間隔の分布である．クローラが同一ページに再アクセスする周期は 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できるため，1 日未満の推定値も得られている．ここでは Zipf 分布から乖離し，極大値を持つ分布となっている．この分布はクローリング自体の周期を短縮すると変動する可能性があるが，つねに Zipf 分布のように最小周期の頻度が最大になるとは考えにくい．

そこで，この観察に基づき，**図 6** (a) に示す 2 つのデータを用いることにした．1 つは純粋な Zipf 分布であり，スケジュール間隔 i（$1 \leq i \leq T$）を持つレコード数 $f_{\mathrm{Zipf}}(i)$ が次式に従うものとする．

$$f_{\mathrm{Zipf}}(i) = N/H_T \cdot i^{-1} \tag{7}$$

ただし，H_T は T 番目の調和数であり，次式で与えられる．

$$H_T = \sum_{k=1}^{T} k^{-1} \tag{8}$$

もう一方は周期 100 単位時間まで周期に比例する頻度を持ち，それ以降は反比例に転ずる分布である（以降では Peaked と呼ぶ）．そのレコード数の分布 $f_{\mathrm{Peaked}}(i)$ は，

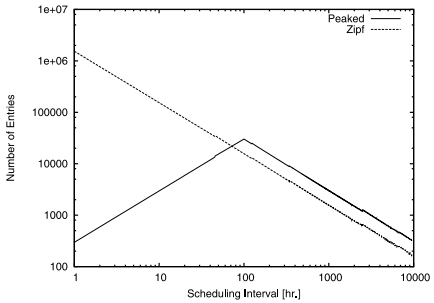
$$P_T = \sum_{k=1}^{100} \frac{k}{100^2} + \sum_{k=101}^{T} k^{-1} \tag{9}$$

で与えられる P_T を用い，次式で表すことができる．

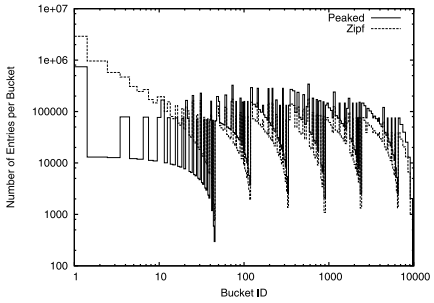
$$f_{\mathrm{Peaked}}(i) = \begin{cases} N/P_T \cdot i/100^2 & \cdots \quad 1 \leq i \leq 100 \\ N/P_T \cdot i^{-1} & \cdots \quad 100 < i \leq T \end{cases} \tag{10}$$

いずれの分布においてもレコード数の合計 N は 1.5×10^7 とした．

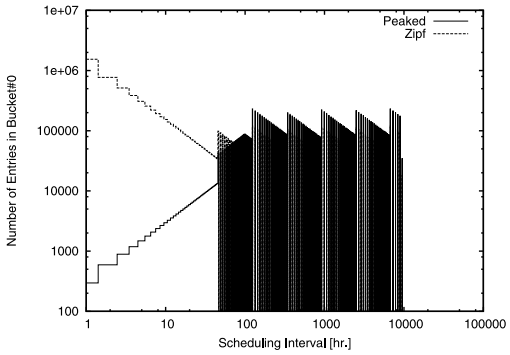
各レコードは 200 バイトの固定長とし，状態データベース全体が後述する評価用マシンの主記憶容量を大きく上回るようにした．図 6 (b) は SAT/L における，時刻 0 での各バケットのサイズを示したものである．可変時間分解能の適用により，レコードが一部のバケット



(a) スケジューリング周期の分布



(b) SAT/L の初期状態におけるバケットサイズ



(c) SAT/L の先頭バケット中のレコードのスケジューリング周期の分布

図 6　評価用データセットの特性（最大周期 $T = 9{,}600$，レコード数 $N = 1.5 \times 10^7$）
Fig. 6　Characteristics of evaluation data.

に局在していることが分かる．SAT/L の性能評価においては，この状態から処理を開始した．また，図 6 (c) は最初に処理対象となるバケットにおけるレコードのスケジューリング周期の分布である．Zipf 分布では直近に再スケジュールされるレコードが最も多いのに対し，Peaked では離れた時刻に再スケジュールされるものが多く，書き込みバッファの使用傾向やフラッシュの発生頻度などに違いが生ずると考えられる．

4.2　評 価 環 境

表 2 に性能評価を行った環境の諸元を示す．この環境は並列クローリングを実施した際の 1 ノードを構成する PC サーバであり，前記 1.5×10^7 URL は，10 億 URL を 64 ノードに分割した場合に相当する．JFS（Journaled File System）[10] はエクステントによる領域割当てを行うファイルシステムであり，Red Hat Enterprise Linux 3 の標準的なファイルシステムである EXT3（Third Extended File System）と比べ，大容量ファイルのシーケンシャルアクセスの性能が高い．SAT および SAT/L 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．書き込みバッファは 4 KiB の固定長ページ単位に割り当てるものとした．

表 2　性能評価環境の諸元
Table 2　Specification of performance evaluation environment.

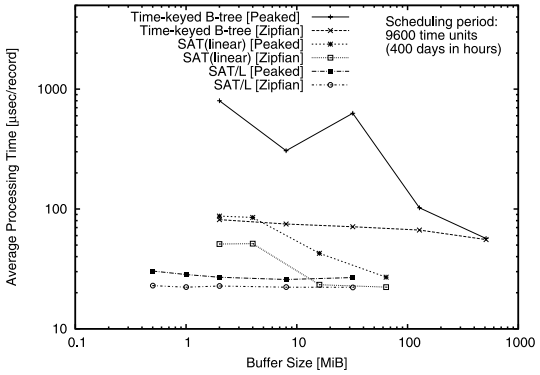
CPU	Intel Core Duo T2500 2 GHz
Memory	2 GiB
HDD	ST3500630NS × 2
Interface	SATA 3.0 Gb/s
Capacity	500 GB
Max. sustained transfer rate	72 MB/s
Cache	16 MB
Average latency	4.16 msec
Spindle speed	7,200 rpm
Random read seek time	< 8.5 msec
Random write seek time	< 10.0 msec
OS	Red Hat Enterprise Linux 3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

また，比較対象としてスケジュール時刻をキーとする B-tree（Time-keyed B-tree）を加えた．要素の取り出しは最小キーに対応するレコードの取得と削除により行い，書き込みは新規レコードの挿入として実装した．B-tree 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[11]．B-tree のページサイズは 64 KiB とした．従来型の URL をキーとする B-tree と SAT の性能比較に関しては文献 5) を参照いただきたい．

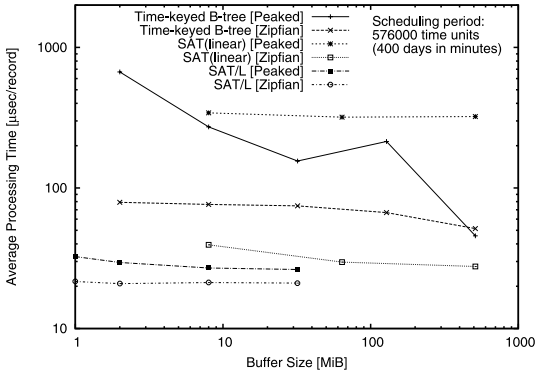
状態データベースの更新性能のみを評価するため，更新プロセスの動作は極力簡略化した．すなわち，読み込んだ属性値をそのまま更新後の値とし，アクセス間隔の値は定数として扱った．また，コンテンツの取得とアーカイブへの格納も省略した．

4.3　バッファサイズに対するレコードあたり処理時間の変化

図 7 (a) および (b) はバッファサイズ（総容量）を変化させた際の 1 レコードあたり処理時間を，2 つのスケジューリング間隔範囲について示したものである（縦軸は対数軸）．そ



(a) $T = 9{,}600$（1 時間〜400 日相当）



(b) $T = 576{,}000$（1 分〜400 日相当）

図 7　バッファサイズに対するレコード処理時間の変化
Fig. 7　Record processing time vs. buffer size.

れぞれ 30 単位時間分のレコードを処理し，経過時間と処理レコード数から求めた．B-tree においては，Berkeley DB が備えるプロセス空間内のデータベースページキャッシュのサイズとしてバッファサイズの値を指定した．

図 7 (a) によると，高速な順に SAT/L，SAT（SAT (linear) と表記），Time-keyed B-tree となっており，データセットの違いによる影響の小ささも同じ順序となっている．Zipf 分布に比べて Peaked の方がレコードの格納先が多様になり，書き込みコストの増大を招きやすいと考えられる．SAT ではバッファサイズが 16 MiB を下回ると処理時間が増加し始めるのに対し，SAT/L では 1 MiB 以下のバッファでも十分な性能を発揮している．Time-keyed B-tree については，Zipf 分布ではある程度の性能を維持できているものの，Peaked データセットに対しては 512 MiB 程度の大容量バッファを与えない限り十分な性能が得られない．これは，Berkeley DB のキャッシュが書き込みデータの再利用による I/O 回避の効果を狙って設計されていることによると考えられる．SAT や SAT/L における書き込みバッファは I/O 回避のためではなく，I/O の粒度を上げ，スループットを向上させるために用いられており，Peaked のように直近にアクセスされないデータの書き込みが頻発する際に有効に機能するといえる．

図 7 (b) ではさらに時間分解能が上がるため，SAT/L 以外の方式で性能低下が著しくなっている．SAT (linear) は，$T = 9{,}600$ ではバッファ容量が大きければ性能を維持できたのに対し，$T = 576{,}000$ では Peaked に対してほぼ最低の性能となっている．多数の小容量バッファが主記憶を奪い合い，小規模なフラッシュを頻繁に引き起こしていることが原因と考えられる．以上の結果から，可変時間分解能を用いる SAT/L 方式は，固定時間分解能による SAT や，バケット分割の概念がない Time-keyed B-tree と比べて，バッファサイズやデータ分布の影響を受けず，安定して高い性能を得られることが分かった．

4.4　書き込みバッファ管理方式の効果

図 8 は SAT/L において各書き込みバッファを独立した固定サイズのバッファとして扱う SAT/L naive 方式のレコードあたり処理時間を示したものである．スケジューリング間隔範囲は $T = 576{,}000$ とした．バッファサイズの変化に対しては安定した挙動を示しているものの，Peaked データセットに対し，SAT/L naive の性能が大きく低下している．Zipf 分布に対しては大多数のレコードが同一バッファに集中するため，当該バッファからのフラッシュ動作で全体の性能が支配されるが，Peaked では多数の書き込みバッファにレコードが振り分けられるため，より高度なバッファ管理が必要になることを示している．

4.5　スケジューリング誤差解消動作の影響

図 9 は処理開始からの各時刻におけるバケットあたりの処理時間の推移を示したものである．スケジューリング間隔範囲は $T = 576{,}000$ とし，バッファサイズを 32 MiB とした．SAT/L (strict timing) と示したものは，バケット中のレコードの本来のスケジューリング時刻が現在時刻と異なるものを，スケジューリング時刻を変更しないまま再スケジューリングしたものである．再スケジューリングではレコードのスケジュール間隔より小さな間隔でスケジューリングが行われるため，直近の処理対象レコードの増加が懸念される．しかし，図 9 によれば，両者の差はわずかなものであり，スケジューリング時刻の精度を犠牲にし

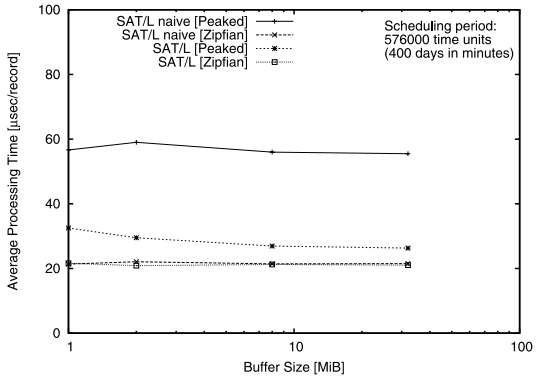


図 8　異なる書き込みバッファ管理方式に対するレコード処理時間の変化
Fig. 8　Record processing time with different buffer management methods.

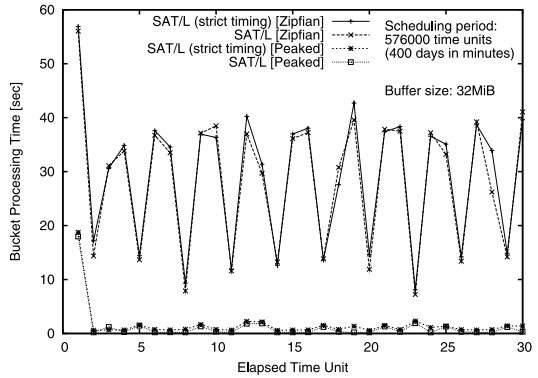


図 9　各時刻におけるバケット処理時間のトレース
Fig. 9　Trace of bucket processing times at each time slice.

なくても SAT/L の性能は維持されることが分かる．

5. 関 連 研 究

本稿で述べた SAT/L 方式は，二次記憶上で大規模なイベントキュー（優先順位付きキュー）を効率的に実現することを目指したものである．

離散事象のシミュレーションにおいてはイベントキューへの挿入操作における探索コストを削減するため，様々なデータ構造が考案されてきた．その多くは Calendar Queue[12] の改良に基づくものである[13]．Calendar Queue は $O(1)$ による挿入を実現するために各時刻に対応するリストを用意するが，スケジューリング間隔の増大に対しては，SAT の書き込みバッファのようにリストを際限なく増加させるのではなく，ラウンドロビン的に異なる時刻のイベントを同一リスト内に混在させるというアプローチをとる．したがって，イベントキューの読み出し時には，リスト内の不要な時刻のイベントをスキップする必要がある．その結果，Calendar Queue を二次記憶に単純に実装すると，時刻の混在により無駄な I/O 帯域を消費してしまう．また，SAT/L のように近接する時刻の混在ではなく，大きく異なる時刻の混在であるため，許容できる誤差範囲にあるとは限らない．

二次記憶への実装を意識した優先順位付きキューとして，Brengel らは R-heaps を提案している[14]．R-heaps は SAT/L と同様のアイデアに基づいており，キューの最小優先度（現在時刻に相当）と挿入優先度の差の上位ビットに基づいて要素をバケット分割する．しかし，SAT/L と異なり，キューの最小優先度が次第に上昇してバケットの優先度との差が小さくなると，当該バケットの内容を 1 ビット下位の差に対応するバケットに分割し直す．したがって，キューの先頭に到達したバケットにおいては優先度の混在はない．R-heaps ではこのように積極的にバケットの再分割を行うため，余分な読み書きが発生してしまう．SAT/L ではバケット中の時刻の混在を許し，先頭に到達した時点で誤差が許容範囲かどうか判定し，必要最小限の再スケジューリングにとどめることができる．また，R-heaps では複数バケットに対して直接書き込みを行うことを想定しており，高度なバッファ管理による性能向上については考慮されていない．

一方，二次記憶アクセスにおける書き込み性能を向上させる手法として，LFS（Log-structured File System）[15] や Write-optimized B-tree[16),17] などのログ構造化の手法が知られている．しかし，イベントキューの操作においては，先頭要素の効率的な読み込みも重要であり，ログ構造化手法は書き込み後のデータの再読み込みにおいてはむしろ不利になると考えられる．

6. ま と め

本稿では，多周期的 Web クローリングにおける時間分解能を向上させ，収集周期の範囲を大幅に拡大するための手法について提案し，その有効性を示した．筆者らが従来提案していたアクセス予定時刻順編成では，ディスク書き込み性能の向上のため，主記憶上でアクセ

ス予定時刻ごとにレコードをバッファリングすることが重要であり，アクセス予定時刻がとりうる値の範囲を大きくするとバッファリングの効果が薄れてしまうという課題があった．

本稿で提案した可変時間分解能方式では，アクセス周期と時刻許容誤差の間に見られる一般的な傾向を利用し，長周期アクセスされるレコードをより少数の書き込みバッファでまとめて扱うことにより，短周期アクセス側の時間分解能の向上を実現した．また，厳密な時刻管理が求められる場合に対しても，再スケジューリングにより大きな性能低下を招くことなく対応できることが分かった．本方式により，多様な情報源からの継続的な情報収集全般に対し，大容量メモリを要求することなく，高効率にスケジューリングを行うことが可能になる．

一方，通常のデータベース管理システムは外部からの更新要求に基づいてデータの更新を行っており，システム自体がデータ更新タイミングを決定できる本稿の手法を直接適用することはできない．しかし，レコードごとの更新要求の到着がポアソンモデルに従っていると見なすことができる場合，更新要求を蓄積したバッファに対して本稿の手法を適用することにより，更新性能を大きく向上させることができるものと期待できる．今後は，収集した Web 情報に基づくリンク元 URL データベースや全文検索用転置索引のオンライン更新に対し，本手法の適用を進めたい．

参 考 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) Kitsuregawa, M., Tamura, T., Toyoda, M. and Kaji, N.: 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，電子情報通信学会論文誌 D，Vol.J91-D, No.3, pp.551–559 (2008).
5) 田村孝之，喜連川優：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，電子情報通信学会論文誌 D，Vol.J93-D, No.6, pp.805–815 (2010).
6) Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T.: RFC 2616: Hypertext Transfer Protocol – HTTP/1.1 (1999).
7) Cho, J. and Garcia-Molina, H.: Estimating frequency of change, *ACM Trans. Internet Technology*, Vol.3, No.3, pp.256–290 (2003).
8) Twitter, Inc.: Rate Limiting (2010). http://dev.twitter.com/pages/rate-limiting
9) Adamic, L.A. and Huberman, B.A.: Zipf's law and the Internet, *Glottometrics*, Vol.3, pp.143–150 (2002).
10) JFS core team: Journaled file system technology for Linux. http://jfs.sourceforge.net/
11) Oracle Corp.: Oracle Berkeley DB. http://www.oracle.com/technology/products/berkeley-db/db/index.html
12) Brown, R.: Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem, *Comm. ACM*, Vol.31, No.10, pp.1220–1227 (1988).
13) Hui, T.C.-K. and Thng, I.L.-J.: FELT: A Far Future Event List Structure Optimized for Calendar Queues, *SIMULATION*, Vol.78, No.6, pp.343–361 (2002).
14) Brengel, K., Crauser, A., Ferragina, P. and Meyer, U.: An experimental study of priority queues in external memory, *J. Exp. Algorithmics*, Vol.5, No.17 (2000).
15) Rosenblum, M. and Ousterhout, J.K.: The design and implementation of a log-structured file system, *ACM Trans. Comput. Syst.*, Vol.10, No.1, pp.26–52 (1992).
16) Graefe, G.: Write-optimized B-trees, *Proc. VLDB'04*, pp.672–683 (2004).
17) Graefe, G.: B-tree indexes for high update rates, *SIGMOD Rec.*, Vol.35, No.1, pp.39–44 (2006).

(平成 22 年 9 月 20 日受付)
(平成 22 年 12 月 27 日採録)

(担当編集委員　吉田 尚史)

田村　孝之（正会員）

1991 年東京大学工学部電子工学科卒業．1996 年同大学大学院工学系研究科情報工学専攻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 年三菱電機株式会社に入社．博士（工学）．現在，同社情報技術総合研究所専任ならびに東京大学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．電子情報通信学会，日本データベース学会，ACM，IEEE Computer Society 各会員．

喜連川　優（正会員）

1978 年東京大学工学部電子工学科卒業．1983 年同大学大学院工学系研究科情報工学専攻博士課程修了．工学博士．同年同大学生産技術研究所講師．現在，同教授．2003 年より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 年 ACM SIGMOD Edgar F. Codd Innovations Award 受賞．現在，本会副会長，日本データベース学会理事，電子情報通信学会フェロー．電子情報通信学会データ工学研究専門委員会委員長（1997〜1998 年），ACM SIGMOD Japan Chapter Chair（1999〜2002 年）歴任．VLDB Trustee（1997〜2002 年），IEEE ICDE，PAKDD，WAIM 等ステアリング委員，IEEE ICDE Program Co-chair（1999 年），General Co-chair（2005 年）．