LCSを用いたアクセスログ解析の並列処理による性能向上

戸田　誠二[†]　　横田　治夫[††,†]

† 東京工業大学 大学院 情報理工学研究科 計算工学専攻
†† 東京工業大学 学術国際情報センター
E-mail: †toda@de.cs.titech.ac.jp, ††yokota@cs.titech.ac.jp

あらまし　近年，Web による情報発信は重要な位置を占めている．このため Web サイトへのアクセス状況を的確に把握することが管理者に求められる．これに対して，我々は LCS を用いたアクセスログ解析を行うことで頻出アクセスシーケンスを発見する手法を提案している．また我々はこの手法に対し，ハッシュ関数を用いた LCS 対象の削減による性能改善手法も提案している．本稿では処理の並列化により，LCS を用いたアクセスログ解析におけるさらなる性能向上を図る．これにより長期間にわたり取得されたログ，大規模サイトで取得されたログに対しても利用が可能になると期待できる．実際に本手法を実装し，アクセスログ解析を行うことでその効果を測定し，考察する．

キーワード　Web，データマイニング，Longest Common Subsequences，アクセスシーケンス抽出，並列処理

The Performance Improvement by Parallel Processing for Access Log Mining with LCS

Seiji TODA[†] and Haruo YOKOTA[††,†]

† Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology
†† Global Scientific Information & Computing Center, Tokyo Institute of Technology
E-mail: †toda@de.cs.titech.ac.jp, ††yokota@cs.titech.ac.jp

Abstract　Nowadays, information distribution via websites is one of the most important issues. Therefore, website administrators are required to understand access trends of the websites properly. In our previous work, we have proposed a method for mining access logs with LCS (Longest Common Subsequences) to extract frequent access sequences. We have also proposed two approaches to improve the performance. In this paper, we achieve more improvement by parallel processing and incremental mining for analyzing larger access logs. We evaluate the efficiency of the approaches using access logs of a real website.

Key words　web, data mining, longest common subsequences, access sequence extraction, parallel processing

1. はじめに

現在，Web サイトは広報・広告・マーケティングの手段として極めて重要な位置を占めている．しかし現在は情報が氾濫しており，一見充実したコンテンツやサービスを展開している Web サイトであっても，効果が上がらない，収益が上がらないというものは数多くある．こういった場合の多くは，ユーザビリティに問題があり，ユーザが自分の望む情報を効率良く獲得できていないと考えられる．ユーザにとって使いやすく，分かりやすい，つまりユーザビリティの高い Web サイトを構築することによりこのような問題を解決することが必要である．

このような状況の中で，Web サイトへのアクセス状況を的確に把握し発信したい情報がユーザに確実に届いているかどうかを見極めることは Web サイト管理者の最も重要な仕事の一つである．その手法の一つとして Web サイトのアクセスログ解析がある．管理者はアクセスログの解析結果から取り出す情報を意思決定材料とすることで，ユーザのニーズにあったサイト構築を行うことが可能となる．

Web サイト管理者は，IP アドレス，URL，アクセス日時や，Cookie 情報等をアクセスログとして記録することができる．一般的に広く行われているアクセスログ解析のアプローチにはページ別アクセス頻度，時間別，期間別アクセス頻度，訪問者の使用している OS，ブラウザの種類の集計，Web サイトの参照元の集計などが存在する．この解析手法によってサイト管理者はページ別，時間帯別のアクセス状況，自サイトへの参照元などを知ることができる．これらの情報も管理者にとって重要

な要素の一つであるが，それだけでは訪問者がどのようにサイトを巡回する傾向にあるのか判断できない．このような単純な集計だけではサイトの構成上の問題点について理解することはできない．

我々は以前の研究でアクセスパターンを解析し，ユーザの高頻度な巡回パターンを抽出するため，LCS (Longest Common Subsequences) を用いたアクセスログ解析を行った [1]．これは目的のサイト内での全セッションの URL の推移をシーケンスとして抽出し，各シーケンスについて総当たりで LCS を求め，頻出アクセスパターンを発見するものである．また，集計された LCS をサイトの再構成に活用する手法についても考察を行った．

しかし，以前の研究では単純に各シーケンスに関して総当たりで LCS を計算していたため，その計算量がアクセスログのセッション数の二乗オーダで増加するという問題があった．一般的に，アクセスログ解析は特に商用の大規模サイトでの需要が高く，高いスケーラビリティが要求される．このため本稿では，アクセスログからセッションを抽出し，そこから LCS を抽出し頻出シーケンスを発見する過程において，計算時間を短縮することで性能の改善を図る．我々は [2], [3] においてアクセスシーケンスから LCS を計算する際，ハッシュによって計算対象を削減することと，完全に同一のシーケンスを統合することを提案しているが，本稿ではさらに処理の分割を図り，インクリメンタルな解析手法，並列処理を用いる性能改善手法をそれぞれ提案する．さらに上記手法を利用して，我々の研究室で公開している Web サイトのアクセスログを解析することで，性能改善の効果を検証する．

本稿の構成は以下の通りである．2. 節では関連研究について述べる．3. 節では LCS を用いたアクセスログ解析について説明する．4. 節では提案手法の説明を行う．5. 節で実験，考察を行い，6. 節でまとめる．

2. 関 連 研 究

近年，データマイニング手法等を利用したアクセスログ解析の研究は，盛んに行われてきている．

特に，Web サイトの再構成を目的とした研究としては Srikant らによるユーザのバックトラックポイントの減少を目的とした解析手法が挙げられる [4]．しかし，ユーザ操作はランダム性に富み，またネットワークの状態はユーザごとに異なるため，アクセスログのみからバックトラックポイントと目的のページを区別することは困難である．

また，アクセス解析を目的としたアクセス状況の可視化の研究も盛んである．その中でもユーザのクリック操作の可視化を目的としたものが多く，Web サイトのアクセス状況を示し，解析，改善を支援する様々なツールが提案されている [5], [6]．構造解析のためには可視化が重要であるが，それらのアプローチは個々のページのアクセス頻度や推移に主眼を置いており，ユーザのアクセスシーケンスの傾向を見いだすことはできない．

我々は以前の研究において，LCS (Longest Common Subsequences) を用いた手法を提案している [1]．これは目的のサイト内での全てのセッションについて URL の推移をシーケンスとして抽出し，各シーケンスについて総当たりで LCS を求め，頻出アクセスパターンを発見するものである．このように抽出された頻出アクセスパターンはより効率的なサイトの再構成に役立つ．また，頻出パターンを可視化したり，他の解析手法と併用したりすることによってさらなる効果が見込める．しかしながら，この手法は計算量が大きく，スケーラビリティに問題があった．

この LCS を求める問題は DNA や蛋白質の類似性の比較に用いられるため，バイオインフォマティクスの分野でも議論されている．[7] ではこの問題を解く多数のアルゴリズムについて述べられている．これらのアルゴリズムでは平均の時間計算量を小さくできることを示しているが，比較する 2 つのシーケンスの長さをそれぞれ M，N とした場合，最悪の時間計算量は単純な動的計画法で達成される $O(MN)$ 以上に改善していない．平均の計算量の改善を図る場合，Web ログの解析においては解析対象のデータの特徴に適した新たな LCS 抽出手法を用いる必要がある．

また Web ログの解析は DNA や蛋白質の類似性の比較と異なり，個別のシーケンス長は短いものの，多くの LCS 抽出問題を解く必要があるという特徴がある．このため，本研究では個別の LCS 抽出だけでなく，解析全体の効率化を達成する手法を用いる．

3. LCS を用いたアクセスログ解析

アクセスログ解析において，まず生のログに対し前処理を行い，ユーザのセッションを抽出する必要がある．次にマイニングアルゴリズムを適用することで様々なパターンを生成する．これをパターン分析することで有用なパターンを得ることができる [8].

具体的には，本手法ではまずアクセスログからセッションを抽出する．抽出されたセッションでアクセスされた URL の列を，URL を各要素に持つシーケンスと見なす．次に各セッションについて総当たりで任意の 2 つのアクセスシーケンスから LCS を抽出する．そして得られた LCS を集計し，高頻度なアクセスパターンを発見する．

以下，本手法について詳しく説明する．

3.1 前 処 理

アクセスログ解析を行う際，蓄積されている生のアクセスログを精錬してマイニングに必要なデータのみを取りだし，セッション毎に抽出する必要がある．以下，その手法について説明する．

まず [9] にある標準的な前処理により，Web ページ間の移動情報に関係のない HTML ファイル以外のファイルへのリクエストを取り除く．検索エンジンのロボットのアクセス情報等も除去する．これにより，Web ログファイルは $\frac{1}{4}$ から $\frac{1}{10}$ 程度に減少する．

また，訪問者が対象サイト内の 1 ページのみを見てセッションを終了した場合，目的とするサイト内での移動情報を得ることができない．このため Web サイトの再構成に利用するのが

難しいと判断し，このような情報も取り除くこととする．

さらにアクセスログからセッションを抽出する．本研究では，Cookie を用いて各セッションに一意なセッション ID を割り振る手法を用いる．これにより，プライベート IP で管理される複数ユーザが同一 IP からアクセスした場合等も区別することができる．

3.2 LCS の抽出と頻出アクセスパターンの発見

まず LCS とは何かについて簡単に説明し，次に LCS の抽出手法と頻度集計について述べる．

3.2.1 LCS

リスト x の部分列 x_a とリスト y の部分列 y_b の中で両方のリストに含まれるものを共通部分列という．共通部分列の中で最も長いものを最長共通部分列 (Longest Common Subsequences) と呼び，頭文字をとって LCS と略する．

二つのリストの中に同じ要素が同じ順序で出現したものが共通部分列なので，LCS が長いということは二つのリストの類似性が高いことを表す．

以下にリスト X，Y の LCS を抽出した例を示す．

$$X = (\underline{A}, F, \underline{B, D}, E)$$

$$Y = (\underline{A, B}, C, \underline{D}, E)$$

$$LCS(X, Y) = (\underline{A, B, D})$$

これを URL シーケンスに適用することで，URL シーケンス間の類似性を発見することができると考えられる．

3.2.2 LCS の抽出

Wu らは LCS を求める問題と等価である SED (Shortest Edit Distance) を求める問題に関して，効率のいい手法を提案している [10]．本研究ではこのアルゴリズムを用いて，アクセスログから得られた全セッションについて総当たりで任意の 2 セッション間の LCS を求める．

我々は [2], [3] でさらに時間計算量を小さくするための手法を提案しているが，これについては 3.3 節で詳しく述べる．

3.2.3 頻出アクセスパターンの発見

前節ではアクセスログから得られたセッションのアクセスシーケンスに関して LCS の抽出を行う手法について述べた．次に得られた全ての LCS に関して集計を行い，その頻度に基づき LCS を並べ替える．これにより頻出アクセスパターンを知ることができる．

3.3 LCS 抽出の効率化

LCS を用いたアクセスログ解析手法では抽出されるセッションの総数を S とすると，セッション数の二乗に LCS 計算の平均計算量をかけた $O(S^2NP)$ の計算量が必要となる．このためセッション数 S が大きくなるとき，その二乗に比例して時間計算量が増加するため，スケーラビリティに問題がある．我々は [2], [3] において LCS 抽出の効率化の手法を提案している．以下，その手法について説明する．

スケーラビリティ改善のアプローチとして，LCS 生成の対象となるセッションの組み合わせと，LCS 算出時の要素数を削減する手法が考えられる．本手法ではセッションの組み合わせと要素数の両方を同時に削減する手法として，ハッシュによる手法を用いる．これまで関係データベースの結合演算等において，対象の組み合わせを削減するためにソートやハッシュを用いた手法が提案されている．しかしソートを用いる手法は，リスト内部の部分列を対象とする場合は効果が望めない．ハッシュを用いることで比較の必要がない組み合わせを省略することができる．我々はさらに比較するシーケンス要素の絞り込みにも適用することで LCS の比較回数と LCS の平均計算時間を共に削減する．このアプローチでは最悪の計算量は変わらないが，各シーケンスの平均差異が大きい場合に計算量を下げることができる．

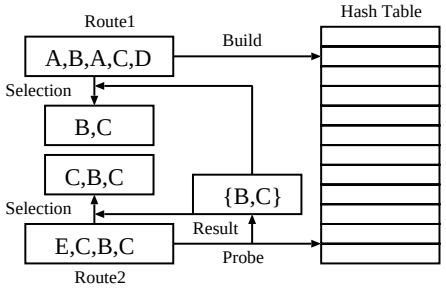

図 1　ハッシュによる LCS 対象の削減

これに加えて解析対象のアクセスシーケンスのうち，まったく同一のものを統合して計算することでさらに実行時間を短縮する．

計算対象の削減は対象とするシーケンスが多様であるほど有効であり，同一シーケンスの統合は完全に同一なシーケンスが多いほど有効であるため，併用することで様々な傾向をもつアクセスログに対して効果があると期待できる．

3.3.1 ハッシュによる LCS 対象の削減

本手法では図 1 に示すような方法で LCS 対象の削減を行う．以下にその方法を説明する．

（1） Route1 が要素に持つ URL をハッシュテーブルにマッピングする (Build).

（2） Route2 が要素に持つ URL とハッシュテーブルに含まれる URL を比較する (Probe).

（3） (2) の結果を用いて Route1，Route2 に共通して現れない要素をそれぞれ除去する (Selection).

Route1，Route2 の長さをそれぞれ M，N として時間計算量について見てみると，(1) では $O(M)$，(2) では $O(N)$，(3) では $O(M+N)$ の計算量が必要になる．つまり，LCS 対象の削減全体で $O(M+N)$ の時間計算量がかかることになる．総当たりで LCS を計算する今回の場合，(1) で作ったハッシュテーブルを再利用することで，計算量を削減することができる．また空間計算量に関しては，ハッシュテーブルを記憶するために $O(M)$ の計算量が必要である．

この計算対象の削減を行うことで，LCS の計算を行う際に，共通する部分のみを抽出した結果を計算することができる．当然のことながら，二つのシーケンスに共通して現れない要素を取り除いても LCS の計算結果に影響はない．また 3.2.2 節で説明したように，$O(NP)$ アルゴリズムでは二つの文字列の差異

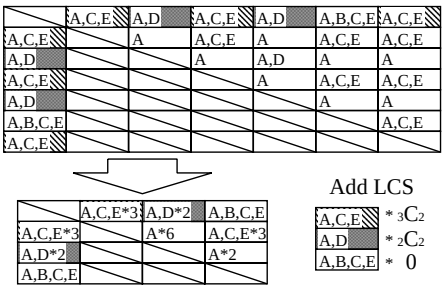

図 2　同一シーケンスの統合

が小さいほど計算量が小さくなるという特徴がある．このため，一方のシーケンスのみにしか現れない要素を取り除くことで，効率的に計算が行うことができると期待できる．本手法では非共通要素を除去するため，一般にアクセスパターンが多岐にわたるほど効果的であるとも言える．

3.3.2　同一シーケンスの統合

解析を行うアクセスシーケンスの中には完全に同一のものが複数含まれていることがあり，これらの LCS 計算をそれぞれ行うことは冗長である．このため，同一のシーケンスを 1 つにまとめて計算回数を小さく抑えるアプローチをとる．本稿で行う同一シーケンスの統合を図 2 に示す．以下，その方法について解説する．

（1） 解析対象の全てのアクセスシーケンスをスキャンし，同一のものを 1 つに統合する．

（2） 統合したシーケンスの数に応じて重み付けを行う．

（3） 従来手法と同様に全てのシーケンスについて総当たりで任意の 2 つのシーケンスから LCS を抽出する．この際，得られる LCS を各シーケンスの重みの積の数加える．

（4） 統合したシーケンス同士の LCS として，n 個のシーケンスを統合した場合 ${}_nC_2$ 通りの組み合わせがあるので，この数のシーケンスを LCS として加える必要がある．

上記の手法により統合を行わない場合と同じ結果を効率的に出力することができる．ただし，この手法も 3.3.1 節で述べた手法と同様，最悪の計算量を減らすことはできない．

統合に必要な計算量は (1)，(2) においてシーケンス全体をスキャンする $O(S)$，(4) で統合したシーケンス自体を LCS に加える $O(S')$ の計算量であり，全体として $O(S)$ の計算量で効率よく計算が行える．ここで S，S' はそれぞれ統合前後のセッション数を表す ($S \geqq S'$)．本手法では同一のシーケンスの冗長な計算を除去しているため，一般にアクセス偏りが大きい場合に特に有効であると言える．

4.　処理の分割による性能改善

3.3 節において LCS の抽出を効率化する手法について述べたが，本稿ではさらなるスケーラビリティの改善を図り，2 つの手法を適用する．

3.3 節で述べた手法では計算効率は向上するものの，全てのアクセスログを逐次的に解析しているため，そのスケーラビリティに限界があった．本節ではこれを改善するため，インクリメンタルな解析手法と並列処理により処理を分割して行うことで，根本的なスケーラビリティの改善を行う．

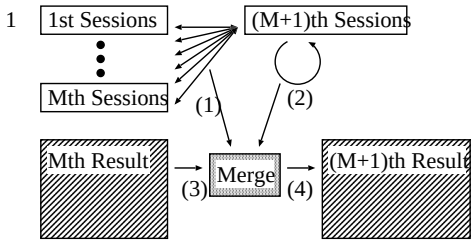

図 3　インクリメンタルな解析

インクリメンタルな解析手法とは，過去の解析結果を利用して新たなアクセスログの解析を行う手法である．Web サイトのアクセスログは定期的に収集されるという特徴があり，これを活かし段階的に解析を行うことで，1 回に行うアクセスログ解析の時間を大幅に削減することができる．ただし，この手法はある特定の時点からの解析を，定期的なアクセスログ収集のたびに行う場合にのみ有効である．

また，一般にアクセスログ解析を行う際に性能を向上させるアプローチとして，並列処理を行うことが考えられる．LCS を用いたアクセスログ解析手法に適した処理の並列化を行うことで，スケーラビリティの改善が見込める．

本節では上記 2 手法について説明する．

4.1　インクリメンタルな解析

Web サイトのアクセスログを解析するにあたり，その全てを計算すると，対象となるアクセスログのサイズの増加に伴い計算時間が大きく増加するという問題がある．これに対し本稿では，段階的に解析を行い，以前の解析結果を利用することでこれを回避するアプローチを取る．この手法を図 3 に示す．

このアプローチは Web サイトの開設・改装時のような，ある特定の時期からの解析を行う場合に有効であり，定期的にアクセスログが収集され，解析結果を逐次追加していくことを前提としている．これにより特定の時期からのアクセスログ解析におけるスケーラビリティの改善が図れる．

この手法について図 3 に沿って説明していく．M 回目までに収集されたアクセスログが抽出され (1st Sessions - Mth Sessions)，その解析結果が既に抽出されているものとする (Mth Result)．

（1） M 回目までに収集されているログ (1st Sessions - Mth Sessions) と，(M+1) 回目に収集されたログの間の LCS を抽出する．

（2） (M+1) 回目に収集されたログの各セッションについて総当たりで LCS を抽出する．

（3） M 回目までに収集されているログ (1st Sessions - Mth Sessions) の解析結果 (Mth Result) を読み込む．

（4） (1)，(2)，(3) の結果をマージして (M+1) 回目の結果を出力する ((M+1)th Result)．

定期的に収集されるアクセスログのサイズがほぼ一定と仮定し，本手法の時間計算量についてみる．M 回目までのセッション数を S，(M+1) 回目のセッション数を s，セッション長の平均を N，セッション間のアクセスシーケンスの平均差異を P，M 回目の解析結果のサイズを R_M とすると，(1) では $O(SNP)$，(

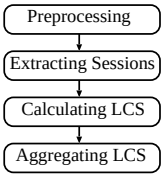

図 4　アクセスログ解析の流れ

2) では $O(S^2NP)$，(3) では $O(R_M)$，(4) では $O(R_{M+1})$ の計算量が必要となる．

従来手法ではセッション数の二乗オーダに比例して計算時間が増加していたが，本手法ではセッション数もしくは解析結果の線形オーダに計算時間の増加を抑えることができる．また本手法は 3.3 節に示す 2 手法を (1)，(2) において組み合わせて用いることでさらなる性能の改善が見込める．実際の性能改善の効果については 5.1 節で示す．

しかし，本手法は 1 回の解析あたりの実行時間を大きく削減することができるが，特定の時期からのアクセスログ解析を行う際にしか適用できないという制約があり，また過去の解析結果を保存する必要がある．

4.2　処理の並列化

本節ではより一般的な状況において解析効率を向上するため，アクセスログ解析処理の並列化を行う．これにより LCS を用いたアクセスログ解析のスケーラビリティをさらに向上することが可能となる．また，前節に述べたインクリメンタルな解析手法においても，過去の解析結果を読み込む部分以外は同様の処理を行うため，並列化が可能である．

LCS を用いたアクセスログ解析では図 4 のような流れで処理が行われる．ここに示されている各処理の並列可能性を考慮し，解析処理全体の並列化を行う必要がある．

収集されたアクセスログをデータベースに格納し，収集されたものから前処理 (Preprocessing) が行われると仮定すると，前処理の並列化の必要はないと考察される．このため本稿ではそれ以降の処理 (Extracting Sessions，Calculating LCS，Aggregating LCS) を可能な限り並列化し，解析時間の短縮を図る．

特に LCS の抽出 (Calculating LCS) にかかる時間はセッション数の二乗オーダで増加するため，並列化による効果が見込まれる．バイオインフォマティクスの分野では LCS の計算の並列化が行われているが [7]，本研究で用いるアクセスログ解析手法では多くの LCS 計算を行う必要があるため，個々の計算を並列化するのではなく，それぞれの問題を各 PE に割り当てるアプローチを取る．1 つの PE がコーディネータとして各 PE の出力する結果をマージすることとする．PE 数が多い場合にはコーディネータを階層化するアプローチも考えられる．以下に LCS 計算割り当て手法を述べる．

単純な割り当て手法

各 PE に LCS 計算問題を割り当てる最も単純な方法の PE 数 3 の場合の適用例を図 5 に示す．この方法では総当たりの LCS 計算を問題ごとに順次各 PE に割り当てている．これにより各 PE に対し，タスクを均等に割り当てることができる．図 5 では最も左の列から右方向へ，各列を上から下にトレースして出現する順に各 PE に割り当てている．このようにラウンドロビンにより割り当てを行うことで，タスクを各 PE に均等に分割することができる．

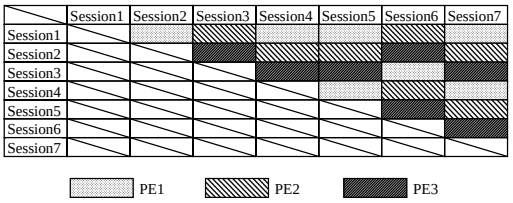

図 5　LCS 計算の各 PE への割り当て (1)

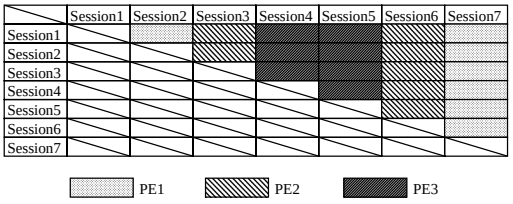

図 6　LCS 計算の各 PE への割り当て (2)

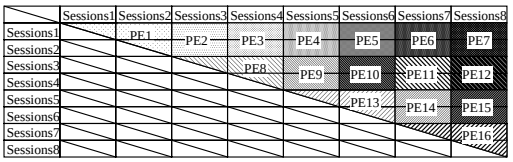

図 7　LCS 計算の各 PE への割り当て (3)

列ごとの割り当て手法

総当たり計算の列ごとに割り当てを行う方法を図 6 に示す．図では PE 数が 3 の場合を示している．この際，両側から 1 列ずつを同じ PE に割り当てることで各 PE に同数の問題を割り当てることができる．このように各列を同 PE に割り当てることで，3.3.1 節の手法を用いる際のハッシュテーブルを，冗長にビルドすることを回避できる．またこれにより比較的類似した問題が同 PE に割り当てられるため，出力し，コーディネータに送信する結果のサイズが小さく抑えられる．このように冗長にハッシュテーブルをビルドすることを回避し，送信する結果のサイズを小さく抑えることが期待できるため，上記の単純な割り当て手法に対し処理の効率化が図れる．

セッション抽出の分割を考慮した割り当て手法

セッションの抽出 (Extracting Sessions) の並列化を考えた場合，単純にタスクを分割すると，その結果を各 PE にブロードキャストする通信コストが大きくなってしまう．これを回避するため，各 PE で割り当てられた LCS の計算に必要なセッションのみを抽出するアプローチを取る．しかし図 5，図 6 に示す割り当て方法では各 PE が全てのセッションを把握する必要がある．このことを考慮した新たな LCS 計算の割り当て方法の PE 数 16 における適用例を図 7 に示す．

LCS を計算するために必要なアクセスログのサイズを小さくするためには，列ごとではなく格子状に分割する方が適している．図 7 の表における上三角形の問題全体を格子状に分割する

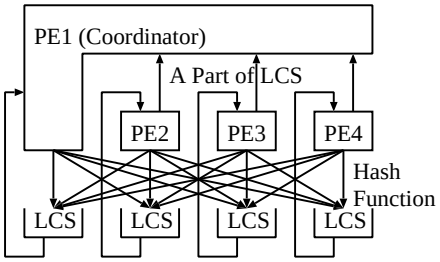

図 8　ハッシュを用いた分割集計

ためには，三角形のメッシュ状に構成することが考えられる．しかし PE 数が N^2 の場合，この方式では最大で $\frac{2}{N}$ のセッション抽出処理を行う必要がある．本手法ではこれを変形した割り当て手法を用いることで，最大で $\frac{3}{2N}$ のセッション抽出処理のみを必要とすることを実現する．

この割り当て方法では N^2 個の PE で並列処理することを仮定しており，まず取得されたアクセスログを $2\sqrt{N^2}(=2N)$ 等分する．その上で各タスクを図 7 のように分割する．具体的にはまず 2 グループずつのセッションから総当たりで LCS を求める問題を N 個の PE に割り当てる．次に 2 つのグループと 1 つのグループから LCS を求める問題を残りの各 PE に順次割り当てる．この分割手法を用いることで，各 PE が最大で 3 グループのセッションを把握することで LCS の計算が行うことができ，セッションの抽出に要する時間を $\frac{3}{2N}$ に抑えることができる．さらに，これにより各 PE が持つテーブルサイズを抑えることもでき，空間計算量についてもスケーラビリティを向上することができる．また LCS 計算に関しては各 PE に均等に分配することができ，比較的類似した計算を同 PE が割り当てられ，解析結果の通信，マージコストを抑えることもできる．セッションの抽出を分割することで，単純な割り当て手法，列ごとの割り当て手法に対しさらに性能を向上させることができると考えられる．

ハッシュを用いた分割集計

次に LCS の集計 (Aggregating LCS) の並列化について考える．各 PE で抽出された LCS をコーディネータに送信し，マージするためのコストはアクセスログのサイズに応じて増加する．しかし実際に必要とされる結果は，頻度順でソートした時に上位に現れるいくつかのアクセスパターンのみであると考えられる．図 8 に示すように各 PE で得られた LCS をそのハッシュ値により分割集計することで，高頻度なアクセスパターンのみを結果として集計することができる．これにより解析結果のサイズを小さく抑えることができ，解析結果の送信，マージコストを小さくすることができる．

本手法では LCS の計算が全て完了した後に結果をまとめて送信するのではなく，LCS が抽出されるたびに通信が行われる．これにより通信量やコーディネータの負荷が時間的に分散されるため，通信帯域が狭い場合や，コーディネータがボトルネックになっている場合に特に効果が期待できる．ただし，LCS 計算中の各 PE が同時に結果の送信を行うことになり，全体の CPU 使用量は増加する．通信帯域が広い場合にはアクセスログ解析全体の性能が低下してしまうことも考えられる．

表 1　対象 Web サイト諸元

URL	http://yokota-www.cs.titech.ac.jp
主要ページ数	約 50
サーバプログラム	Apache 1.3.27
平均リクエスト数	12063 requests/week
言語	日本語/英語

表 2　実 験 環 境

CPU	Intel Xeon 2.40GHz
メモリ	PC2100 SDRAM 3328MB
HDD	IBM Ultrastar 18GB 15000rpm
OS	Linux 2.4.20
Java 環境	Sun JDK 1.4.1_01

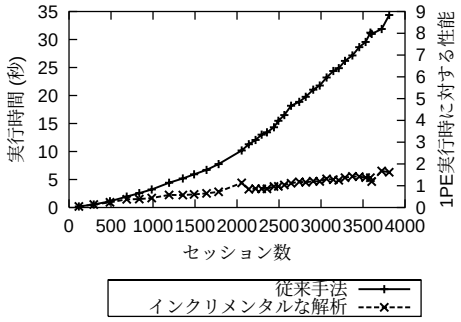

図 9　インクリメンタルな解析の実行時間

5.　実験・考察

本稿で提案する手法の有効性を確認するため，我々の研究室で公開している Web サイトのアクセスログの解析に本手法を適用した．対象とした Web サイトの特徴に関しては表 1 の通りである．

2003 年 5 月 12 日から 2004 年 1 月 19 日までの Web サーバへのリクエストに対するアクセスログを使って解析を行った．前処理を行うことで，49.9MB の生のログから 7.87MB の精錬されたログが得られた．また，前処理を行った後のセッション数は 3812，1 セッション当たりの平均 URL シーケンス長は 4.90 であった．

以降，インクリメンタルな解析，処理の並列化による性能の改善についてそれぞれ実験結果を示し，考察する．

5.1　インクリメンタルな解析による解析時間

Java で実装した実験システムでアクセスログ解析の性能測定を行った．実験環境に関しては表 2 に示す通りである．セッション数の増加に伴う従来手法とインクリメンタルな解析それぞれの実行時間の変化を図 9 に示す．ただし前処理はアクセスログをデータベースに格納する時に行うことが可能であるため，この時間に含まないこととする．ここで比較した従来手法は 3.3 節で示したハッシュによる LCS 対象の削減，同一シーケンスの統合の 2 手法を適用したものである．

従来手法は平均の計算時間を小さくするだけのものであり，セッション数の二乗オーダで実行時間が増加しているのに対し，提案手法ではほぼ線形オーダの増加にとどまることが分かる．

表 3　並列実験環境

ノード数	8 台
CPU	Intel Pentium III 933 MHz
メモリ	PC133 SDRAM 512MB
HDD	Seagate Barracuda IV 20.4GB 7200rpm
OS	Linux 2.2.17
ネットワーク	1000BASE-SX
Java 環境	Sun JDK 1.4.1

このことから，特定の状況において段階的な解析を行うことで大幅にスケーラビリティが改善できたと言える．

またセッション数が 2000 付近の時に解析時間が大きくなっているが，これは Web サイトの構造が改変されアクセス傾向が変わった時点に一致し，これが原因であると思われる．このようにユーザのアクセス動向によって解析時間が変化するものの，従来手法に比べると大きく性能が改善されている．

インクリメンタルな解析手法適用時の並列化可能性を調査するため，過去の解析結果の読み込み時間についても測定を行った．3812 セッション解析時の全体の解析時間は 6.293 秒であり，その内過去の解析結果の読み込み時間は 2.051 秒であった．つまり解析時間の約 67%が並列化可能であるということである．

5.2　並列処理による解析時間

表 3 の環境の下，処理の並列化によるアクセスログ解析の実行時間を Java で実装した実験システムを用いて測定した．本実験では各 PE の結果をマージするコーディネータを，割り当てられた計算を行う PE の 1 つが兼ねることとした．また，前処理を行ったアクセスログを各 PE にあらかじめ配置し，単純な割り当て手法，列ごとの割り当て手法ではそれぞれの PE ですべてのセッション抽出を行い，セッション抽出の分割を考慮した割り当て手法では各 PE で必要なセッションの抽出のみを行うこととした．

単純な割り当て手法の並列処理による実行時間を図 10 に示す．解析対象のアクセスログのセッション数が小さい時には並列化により性能が低下しているが，セッション数の増加に伴い並列化により実行時間が短縮されているのが分かる．また，このグラフの範囲では PE が 5 以上ではほぼ実行時間が短縮されていないが，さらにセッション数が増加した場合，実行時間に差が生じてくることが予想される．

列ごとの割り当て手法の並列処理による実行時間を図 11 に示す．図 10 と比較して特に PE 数が大きい時に実行時間が短縮されている．さらに PE 数が大きくなった時に実行時間が短縮されると考えられる．PE 数が 8 の場合，平均で約 8.8%性能が向上していることを確認した．

単純な割り当て手法の並列処理による台数効果を図 12 に示す．アクセスログ解析に要した時間の逆数を性能とし，PE 数の増加に伴う性能の向上を，1PE 実行時の性能を 1 として正規化し表している．

セッション数が 984 の時はほぼ変化はなく，2056 の時は PE 数が 3 付近から，2984 の時は PE 数が 5 付近から，3812 の時は PE 数が 6 付近から性能の向上が緩やかになっているのが分

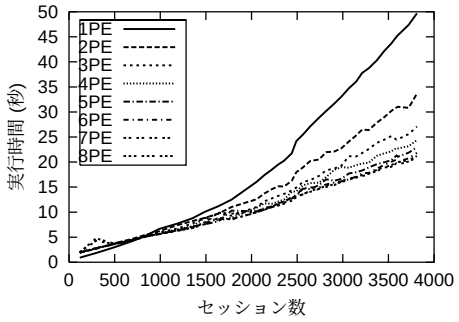

図 10　並列処理による実行時間 (単純な割り当て)

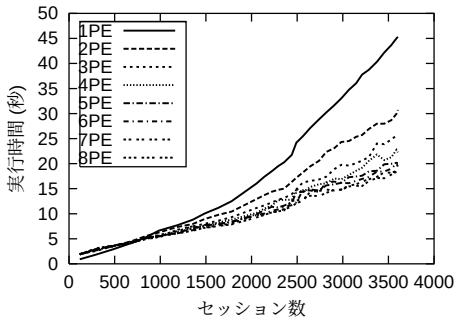

図 11　並列処理による実行時間 (列ごとの割り当て)

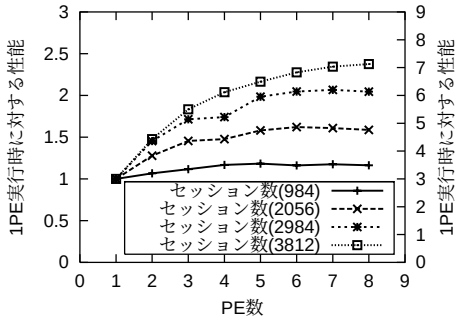

図 12　並列処理による台数効果 (単純な割り当て)

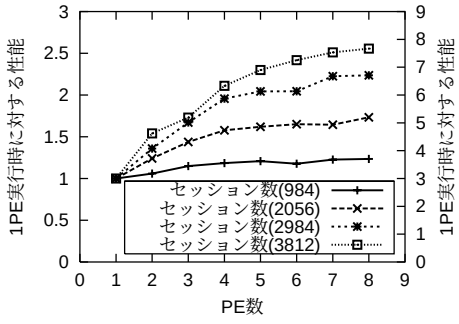

図 13　並列処理による台数効果 (列ごとの割り当て)

かる．セッション数の増加に伴い，並列化した LCS 計算部分の計算時間が全体に占める割合が高まり，台数効果は向上している．しかし，非並列実行部分や並列化のためのオーバヘッドの影響で本実験のセッション数の規模では十分な効果が得られていない．

次に列ごとの割り当て手法の並列処理による台数効果を図 13 に示す．この手法では単純な割り当て手法に比べて並列化のオーバヘッドが小さく，図 12 と比較すると特に PE 数が増加し

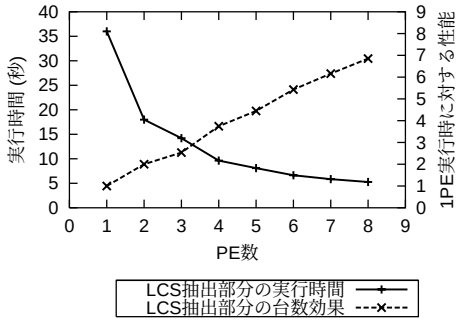

図 14　LCS 抽出部分 (並列化部分) の実行時間，台数効果

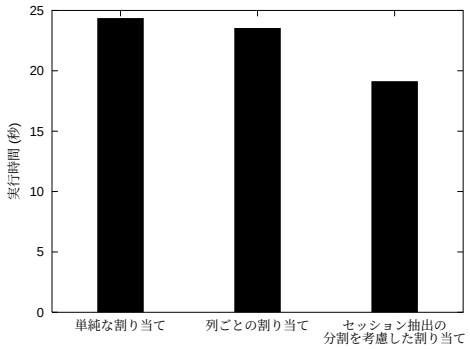

図 16　LCS 計算割り当て手法ごとの実行時間

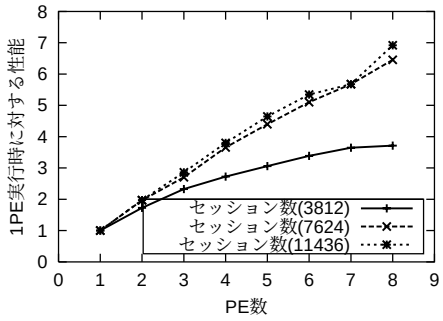

図 15　大きいアクセスログの解析における台数効果

た場合に性能が向上している．PE 数が 8 の場合には最大で約 2.6 倍の性能を示している．

次に解析時間全体における並列化部分の割合，台数効果を測定し，考察する．ただし，本実験における並列化部分は LCS の抽出 (Calculating LCS) 部分である．セッション数 3812 の場合，所要時間は並列化部分が約 36 秒，非並列化部分が約 9 秒という割合であった．また並列化のオーバヘッドは PE 数 2 から 8 の範囲では 3–4 秒程度であった．アムダールの法則によると，処理の割合 p の部分が n 分割されると

$$\frac{1}{(1-p)+\frac{p}{n}}$$

の性能向上が可能である．上記の並列化部分と非並列化部分の割合を代入し，並列化のオーバヘッドの影響を加えると

$$\frac{1}{\frac{20}{100}+\frac{1}{10}+\frac{80}{100}\cdot\frac{1}{8}} = 2.5$$

となり上記の実測値とほぼ合致している．並列化部分の実行時間，台数効果を測定した結果を図 14 に示す．ただし，LCS 計算割り当て手法は列ごとの割り当て手法を用いた．グラフから，並列化部分に関しては 8PE で約 7 倍の性能という高い台数効果が得られていることが分かる．

また 4.2 節でも述べたように，非並列化部分の実行時間はセッション数の線形オーダで増加するのに対し，並列化部分の実行時間はセッション数の二乗オーダで増加する．このため，より大きなサイズのアクセスログの解析においては並列化部分の割合が増加し，さらなる台数効果が見込める．

実際にアクセスログが大きくなった場合の台数効果を調べるため，収集したアクセスログをコピーし，2 倍，3 倍のアクセスログを解析する時間を測定した．コピーしたアクセスログのセッション ID にサフィックスを付与することで，新たに同量のユーザセッションが行われたことを表現しており，これらをあわせて解析することでより大きなサイズのアクセスログを解析するのに必要な時間を見積もることができる．この際，総当たりの LCS 計算が大きくなった場合の実行時間を測定するため，3.3 で述べた同一シーケンスの統合は行わないこととした．この結果を図 15 に示す．このグラフからアクセスログのサイズを 2 倍，3 倍とした場合には線形に近い台数効果が得られていることが分かる．特に 11436 セッションのアクセスログを解析した場合では，8PE で約 7 倍の台数効果が得られている．この場合の並列化部分，非並列化部分，並列化オーバヘッドの所要時間を測定したところ，それぞれ約 950 秒，約 13 秒，約 9 秒であった．それぞれの全体における割合を計算し，前述のアムダールの法則の式に代入すると

$$\frac{1}{\frac{1.4}{100}+\frac{1}{100}+\frac{98.6}{100}\cdot\frac{1}{8}} = 6.79$$

となりほぼ実測値と一致する結果であった．

これらのグラフから，さらにアクセスログのサイズが大きくなった場合，より高い台数効果が得られることが分かり，高いスケーラビリティが達成されていると考えられる．

次にセッション抽出を考慮した割り当て手法を実装し，その効果を測定した．セッション抽出の分割を考慮した割り当て手法では，平方数の PE で処理を分割する必要がある．今回の実験環境では 8PE までしか測定することができないため，4PE の場合のみ測定を行った．3812 セッションのアクセスログを 4PE で分割処理した場合の，LCS 計算割り当て手法による解析時間の違いを図 16 に示す．

4.2 章でも示したように，セッション抽出の分割を考慮した割り当て手法ではセッション抽出の時間を抑えることができるため，解析時間を小さくできていることがグラフから分かる．単純な割り当て手法に対しては 27%，列ごとの割り当て手法に対しては 23%の性能改善を達成している．セッション抽出の分割では，N^2 個の PE で並列処理をした場合にはセッション抽出に要する時間を $\frac{3}{2N}$ に抑えることができるため，PE 数が大きくなった場合にはさらなる効果が望める．

次にハッシュによる分割集計を行った際の実行時間を測定した．今回の実験では，全体を通して分割集計を行わない場合よりも性能がよくない結果に終わった．これは実験環境の影響によるものだと思われる．今回の実験環境では通信帯域がボトル

ネックになることはなく，LCS の計算に解析の多くの時間を費やしている．またコーディネータがボトルネックになる PE 数にも達していない．このため，分割集計のために LCS の送受信を行うことで CPU が使用され，LCS の計算速度が遅くなった分，解析全体の実行時間が増加したものと思われる．本実験の環境ではこの手法による効果が得られなかったものの，環境によってはその効果が期待される．異なる環境における実験については今後の課題とする．

6. おわりに

本稿では LCS を用いたアクセスログ解析の処理を分割し，性能を改善するため，インクリメンタルな解析手法と並列化手法を提案し，実験，考察を行った．

Web サイトのアクセスログは定期的に収集されるという特徴があり，これを活かした解析手法としてインクリメンタルな解析手法を提案した．この手法はある特定の時点からのアクセスログ解析を段階的に行う場合に有効であり，過去の解析結果を利用して解析を行う．定期的なアクセスログの収集に伴い，段階的にその解析を行うことで，ユーザのアクセス動向によって解析時間が変化するものの，従来手法に比べるとスケーラビリティを大幅に改善した．

また，より一般的な解析性能の向上のため，本稿では LCS を用いたアクセスログ解析の並列化を行った．3.3 節で述べたハッシュによる LCS 対象の削減を行う際に，ハッシュテーブルをビルドするコストの削減や，セッション抽出処理の分割を考慮した LCS 計算の割り当て手法を示した．また抽出された LCS をハッシュ関数により分割集計することで高頻度なアクセスパターンのみを結果として集計し，解析結果の送信，マージコストを小さくする手法も示した．

Java で実装した実験システムを用いて実際の Web サイトのアクセスログを解析し，上記の性能改善手法の有効性を示した．この結果，本稿で提案した 2 手法を用いることで大幅に実行時間を短縮することができた．特に並列化手法については，解析対象のアクセスログが大きくなるにつれ高い台数効果が得られ，並列化部分のみでは 8PE で約 7 倍，全体では約 2.6 倍の性能改善が確認された．さらにアクセスログが大きくなった場合を想定した実験では，解析全体について 8PE で約 7 倍の性能向上が確認できた．これにより長期間にわたり取得されたログ，大規模サイトで取得されたログに対しても利用が可能になると思われ，スケーラビリティの向上が達成されたと言える．

本稿では我々の研究室で公開している Web サイトのアクセスログを解析することで，提案手法の有効性を示したが，これは十分に大規模な解析対象であるとは言い難い．実験結果から，さらに大きな Web サイトのアクセスログでも大きな効果が期待できる．実際により大きな Web サイトのアクセスログに本手法を適用し，その有効性を検証することは今後の課題である．

また，アクセスログの解析結果から得られる頻出アクセスパターンの可視化といった，利用性の向上が必要である．特に大規模なサイトに関して，マイニングアルゴリズムを適用することで得られたパターンを分析し，Web サイトの再構成に活用する方法についても今後の課題である．

謝　辞

本研究の一部は，文部科学省科学研究費補助金特定領域研究 (15017233)，独立行政法人科学技術振興機構 CREST，および 21 世紀 COE プログラム「大規模知識資源の体系化と活用基盤構築」の助成により行われた．

文　献

[1] 宇根田純治 , 横田治夫. Web ログの共通シーケンス解析. In **信学技法**, *DE2002-2*. 電子情報通信学会, 2002.

[2] 戸田誠二 , 横田治夫. LCS を用いた Web ログ解析におけるスケーラビリティの向上. In **情処学会研究会報告**, **データベースシステム** *DBS-131-84*. 情報処理学会, 2003.

[3] 戸田誠二 , 横田治夫. Web ログ解析におけるスケーラビリティ向上手法の評価. **日本データベース学会** *Letters*, Vol.2(No.3):9–12, Dec. 2003.

[4] Ramakrishnan Srikant and Yinghui Yang. Mining web logs to improve website organization. In *Proc. of World Wide Web Conf.*, pages 430–437, 2001.

[5] J. Pitkow and K. Bharat. Webviz: A tool for world wide web access log analysis. In *Advance Proceedings First International World-Wide Web Conference*, pages 271–277, May 1994.

[6] Myra Spiliopoulou and Lukas C. Faulstich. WUM: a Web Utilization Miner. In *Workshop on the Web and Data Bases (WebDB98)*, pages 109–115, 1998.

[7] A. Apostolico. String editing and longest common subsequences. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 2 Linear Modeling: Background and Application, pages 361–398. Springer-Verlag, Berlin, 1997.

[8] Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava. Data preparation for mining world wide web browsing patterns. *Knowledge and Information Systems*, 1(1):5–32, 1999.

[9] A. Banerjee and J. Ghosh. Concept-based clustering of clickstream data. In *Proc. 3rd Intl. Conf. on Information Technology*, pages 145–150, Dec 2000.

[10] Sun Wu , Udi Manber , Gene Myers , and Webb Miller. An o(np) sequence comparison algorithm. Information Processing Letters , 35 : 317–323, 1990.