ハードウェア階層型 Aho-Corasick マシンを用いた Web サービス高速処理技術

佐々木 靖 彦[†] 村 山 隆 彦[††] 多 田 好 克[†]

Web サービスを多様な分野に応用することを目的として，リソースが厳しく制限されたコンピュータ上で利用するための処理高速化技術を開発した．Web サービスの処理上のボトルネックとなるメッセージ解析処理に関し，階層型 Aho-Corasick マシンと呼ぶハードウェアモジュールによる方法を提案し，HTTP 処理および SOAP 処理について評価，検討を行った．専用のハードウェア機構を用い，さらに，複数の最適化方式を組み合わせることで，従来のソフトウェア処理に比べて 2 桁〜3 桁の性能向上が可能となることを確認した．

A High-performance Processing Technology for Web Services Using Hardware-mapped Hierarchical Aho-Corasick Machine

Yasuhiko Sasaki,[†] Takahiko Murayama[††] and Yoshikatsu Tada[†]

A high-performance processing technology for web services operating on resource-constrained systems has been developed. We propose the hardware-mapped hierarchical Aho-Corasick machine that drastically reduces the processing time in message-level communications for Web Services. Our approach combines several optimization techniques, and demonstrates a large speed up more than 100x to 1000x over software approach on processing of messages using HTTP and SOAP.

1. はじめに

Web サービスは，XML，SOAP（Simple Object Access Protocol），WSDL（Web Services Description Language）といった要素技術をベースとして，ネットワーク上の様々なサービスを統合処理することを可能とする技術として注目されている[1]．これは，各要素技術が持つ柔軟性と標準性の高さ（XML による任意タグ，WSDL によるインタフェース定義，SOAP によるエンベロープやスキーマなど）を活用することで，組織の壁を越えたオープンなサービスの連携技術 SOA（Service Oriented Architecture）として新しい応用を生み出す可能性があるからである（**図 1**）．

このような Web サービス技術の現状をとらえると，その利用は，情報・ソリューション分野を中心に展開がなされてきた．たとえば，国内における利用例[2)〜4)]や，海外においても Google[5)]，Amazon.com[6)] などの例があり，その裾野を少しずつ広げつつある．しかしながら，Web サービスは，情報・ソリューション分野における浸透は進展しているものの，それ以外の分野への広がりといった見方をすれば，いまだにあまり進んでいる状況にはない．たとえば，環境計測応用，産業プラント応用，自動車応用といった分野では，Web サービスは現在でもほとんど利用されていない．今後，Web サービスに関わる技術をより広い分野で活用していくためには，すなわち，我々が目標とする “分野の垣根を越えた浸透的なサービス連携技術（Pervasive Syndication）” の基盤として活用していくためには，新しい展開を検討する必要がある（**図 2**）．

さて，そうした検討には多様なアプローチが考えられるが，その中でも我々は特に，サービスの粒度に着目したアプローチを検討している．すなわち，提供するサービスの粒度を従来よりも著しく細かくするような基盤技術として，Web サービスのプリミティブ化を検討している．ここでのプリミティブ化とは，処理性能やコストが低いコンピュータを用いても Web サービスのテクノロジを利用できるようにすることを意味する．つまり，これは，現在の Web サービスで利用

† 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications
†† 日本電信電話株式会社 NTT 情報流通プラットフォーム研究所
NTT Information Sharing Platform Laboratories, Nippon Telegraph and Telephone Corporation

されているサーバや PC（数 GHz の周波数で動作する単一プロセッサやマルチプロセッサと大容量のメモリ/HDD を搭載するコンピュータ）ではなく，それらより 2 桁から 3 桁下の性能やコストのコンピュータであっても，SOAP や XML といった技術を用いてサービスの提供が行えるようにすることにほかならない．

本稿は，このような Web サービスのプリミティブ化を実現しようとする際に，特に重要となるメッセージレベルの処理を高速化する方法について述べたものである．以下，2 章では，Web サービス技術の構成要素と処理上の課題について説明し，これに対する我々のアプローチを概説する．3 章では，具体的な高速化手法として，少ないリソースで構成することが可能なハードウェアモジュールである階層型 Aho-Corasick マシンを用いた方法を提案し，その基本構造と最適化の方法などについて示す．特に，階層型の構造を持つハードウェアを用いる点や，複数の最適化の種類とそれぞれの効果について述べた点は，我々が，本稿で初めて明らかにする内容である．4 章では，上述の提案方法を実装した場合の結果を，シミュレータにより評価した結果について示す．最後に，5 章において，評価結果から得られた結論についてまとめる．

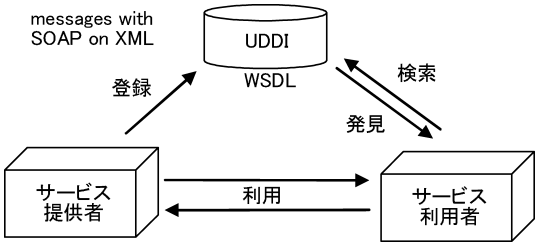


図 1 Web サービス
Fig. 1 Web services.

2. Web サービスとプリミティブ化の課題

2.1 Web サービスを実現するテクノロジスタック

Web サービスとは，単一の技術ではなく様々な技術を積み上げた集合技術である（**図 3**）．すなわち，物理的な信号のやりとりを可能とする信号伝送技術，デジタルレベルでの信号のやりとりを規定する通信上のプロトコル技術，そしてテキストレベルでのメッセージのやりとりを規定するメッセージング技術からなる．文献 7) では，これらを，(1) 物理階層，(2) 通信プロトコル階層，(3) Web サービス階層と 3 つの大きな階層に分けて，各階層の処理について説明している．いかなる機器であっても，Web サービスを実行しよう

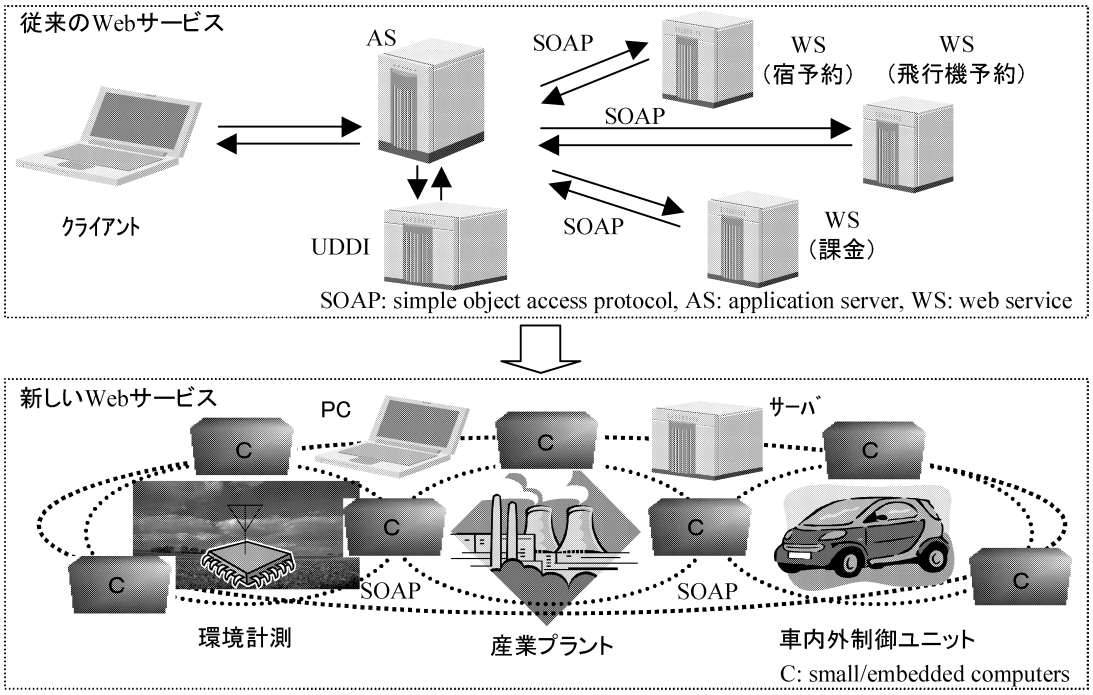


図 2 Web サービスのプリミティブ化による分野横断アプリケーション統合
Fig. 2 Application integration over various fields using Web services primitive technology.

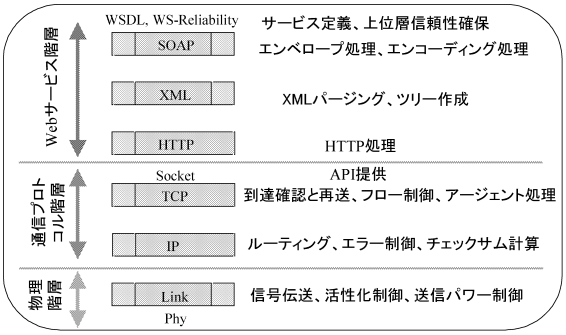


図 3 Web サービスにおけるテクノロジスタック
Fig. 3 Technology stack for Web services

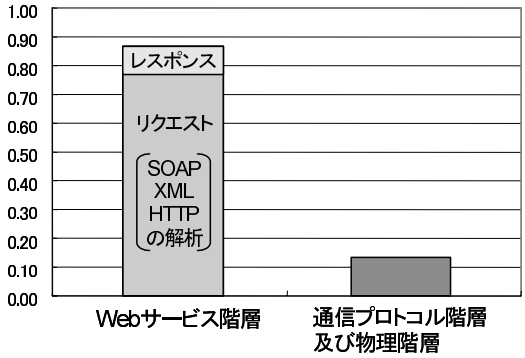


図 4 Web サービスの負荷分布
Fig. 4 Load distribution for Web services processing.

とすれば，少なくともこれらの機能をなんらかの手段を用いて実現することが必要となる．

2.2 情報・ソリューション分野以外への Web サービス活用における課題

情報・ソリューション分野で広く利用されているコンピュータ上では，Web サービスの機能（すなわち，2.1 節で示したテクノロジスタック）を実行させることは比較的問題なく実現できた．その背景には，動作周波数が数 GHz のプロセッサや容量の大きいメモリ/HDD などを装備したコンピュータを用いることができるという前提がある．一方，Web サービスを，先に述べたような広い分野のアプリケーションに適用しようとすると，高性能なプロセッサかつ大容量のメモリ/HDD といった，十分なリソースを有するコンピュータの使用は必ずしも当然の前提条件として規定できなくなる．特に，各種の環境計測応用や産業プラントへの応用などを考えた場合には，搭載できるコンピュータの性能やメモリなどは，現在の Web サービスが利用されているコンピュータにおけるものよりも 2 桁〜3 桁も低い（あるいは小さい）ものを想定する必要が生じる．

我々は過去に，Web サービス単独の機能を，リソース制限が比較的厳しい組み込みコンピュータボード上に実装する試みを行った[7]．ここでは，特別な新しい技術を導入することは行わず，既存のテクノロジを組み合わせることを前提に，極力リソースを削減することを試みた．そこでの実装の結果として，処理性能が大幅に不足しているという知見が得られており，このような処理性能上の問題は，2.1 節で述べた Web サービス階層，すなわちテキストレベルのメッセージ処理にある（ボトルネックとなっている）ことが分かっている．

図 4 は，このような処理時間の内訳について示したものであるが，Web サービス階層の中でも特に，リクエストメッセージの解析処理，すなわち，各種のテキスト処理（HTTP[8]，XML[9]，SOAP[10] の処理）が大きな負荷を占めていることが分かる．

2.3 Web サービスのプリミティブ化に向けた処理高速化

本研究で示すような著しく粒度の細かいサービスを Web サービスとして提供する場合，本来のサービスの処理は軽微であるから，いかにして Web サービス自体の処理を高速化するかが重要となる．このとき，処理性能の向上には，前節で述べたようにボトルネックとなるメッセージレベルの処理を高速化することが必須となる．

我々は，先に行った実装経験を通して，Web サービス処理において著しい高速化を実現するためには，汎用のプロセッサ上のソフトウェア的な工夫だけでは目的の達成が難しいと考えた．そこで，専用（非汎用）の Web サービス処理機能を提供するハードウェア（独立したチップ，あるいは汎用プロセッサと同一のチップ上の別モジュール）について検討を行った．つまり，本来のサービス機能を実行するために汎用プロセッサは利用し，他方，柔軟性の高いインタフェースを提供する Web サービスの機能は，別途専用に用意するハードウェアモジュールに処理させる方法である．

さて，このような専用のハードウェアモジュールの利用形態であるが，汎用プロセッサへの専用入力として直結するようなものであってもよいし，より汎用的な構成として，以下のようなバス上に接続される 1 デバイスとしての形態も可能である．すなわち，汎用プロセッサとの関係で見たとき，本モジュールは 1 つの周辺デバイスとして機能するものである．本モジュールは，Web サービスリクエストメッセージの解析処理が終了すると，汎用プロセッサへ割込み通知を行い，必要な情報はレジスタを介して受け渡しする．反対に

レスポンスの生成処理では，本モジュールは，汎用プロセッサからの情報をレジスタを介して受け取り，それらの情報を使ってメッセージの組み立てを行い，その送信を行う．

3. Web サービス高速処理技術

3.1 従来のテキスト/メッセージ処理技術

テキスト/メッセージ処理は，いわゆる言語処理系と類似の手続きが中心となる．すなわち，

(1) **トークン切出し処理**

(2) **構文解析処理**

が必要となる．トークン切出し処理は，キーワードにマッチするか否かを判断する．また，構文解析処理は，あらかじめ許されている構文に対して，合致するか否かを判断し，合致する場合にどの構文であるかを抽出する．これら（1）および（2）の処理により，SOAP などによるプロトコルに則った Web サービス RPC（Remote Procedure Call）のメッセージが解釈され，呼び出し関数名と引数情報を知ることが可能となる．

まず，トークンの切出し処理，すなわちキーワード（予約語）マッチ問題は，汎用 CPU 上でのソフトウェア処理としては，従来からアルゴリズム的にもよく検討されている．単一キーワードとのマッチに関しては，Boyer-Moore 法[11] や Knuth-Morris-Pratt 法[12] など，効率の良い方法が知られているが，一般に，SOAP，XML，HTTP といった Web サービスで用いられるメッセージの処理においては，複数のキーワードとの一致判断が必要となる．このような場合には，Boyer-Moore 法を複数回適用するといった手法は効率が悪いことが知られており，代わりに Wu-Manber 法[13] などが知られている．

次に，構文解析の方法に関しては，やはり多様な方法が知られている[14]．一般には，トップダウン型やボトムアップ型，また，LL 解析や LR 解析といったものが存在する．

ただし，このようなトークン切出し処理や構文解析処理は，汎用のプロセッサでのソフトウェア処理として実行されることを前提としたものであり，我々が行おうとしているハードウェア利用を考慮したうえでのテキスト/メッセージ処理としては，必ずしもふさわしくない面がある．以下では，このようなハードウェア利用を考慮したうえでの処理高速化について示す．

3.2 ハードウェア化に向いた処理方法

従来の方法は，想定している利用形態が，汎用 CPU 上でのソフトウェア処理としての実行形態である．たとえば，トークンの切出し処理では，対象とするテキストがすでにメモリ上に存在することを前提として，予約されているパターンとの一致を調べるというものである．

本研究ではリソースが非常に限定された小型のハードウェアモジュール上での処理を想定しているが，このとき，従来方式のような SOAP メッセージをいったん全部モジュール内メモリに蓄えてから処理するといったことは難しい．さらに，従来の方式では，副次的なデメリットとして，通信路を介したメッセージの取り込みが完了するまで処理を開始できないため，レイテンシが増加するといった問題も生じる．また，Wu-Manber 法などは，ハッシュテーブルを利用するために，さらに容量の大きいメモリを必要とするといった問題がある．

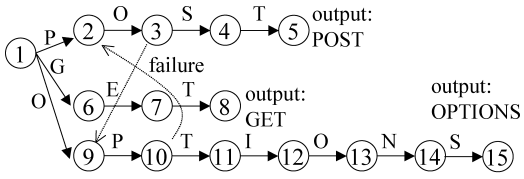
上記のような課題を解決するために，我々は，Aho-Corasick アルゴリズム[15] の利用を検討した．Aho-Corasick アルゴリズムでは，複数のキーワードに対応する 1 つの有限オートマトンを作成し，この上で入力文字列が受理されるか否かを判断してパターンマッチするかどうかを検証する．

我々が，他のアルゴリズムではなく，Aho-Corasick アルゴリズムを特別に選択したうえでこれをハードウェア化しようと考えた理由は，複数のキーワード検索に対して効率が高いといったことはもちろんであるが，さらに以下のような複数の点にもある．まず，Aho-Corasick アルゴリズムでは，入力された文字列中の文字それぞれに対応して状態遷移が行われるために，メッセージをいったんすべてメモリに取り入れる必要がないことである．このことは，リソースの削減に効果的である．さらに，入力が来ると同時に処理を開始できるため，レイテンシを小さくすることにも都合が良い．特に低速系の通信手段を利用する際には，より優位性が高まることを意味する．

3.3 Aho-Corasick マシンとその階層化

Aho-Corasick マシンは，入力文字に応じて遷移条件が与えられる状態遷移図を表現している．**図 5** は，その例を示している．このような表現のために，Goto ファンクション，Failure ファンクション，Output ファンクションの 3 つが用いられる[15]．

Goto ファンクションは，各入力に対して次にどの状態に遷移するかを示している．また Failure ファンクションは，入力に対応する遷移が存在しないとき（すなわち，遷移を与える条件の中に入力文字が含まれないとき），どの状態に移動すればよいかを示している．これは，サフィックスが一致する別の状態への遷移を与える．最後に，Output ファンクションは，ある状



to state	input	goto	failure	output	other
1	-	2	1	-	-
2	P	3	1	-	6
3	O	4	9	-	-
4	S	5	1	-	-
5	T	1	1	POST	-
6	G	7	1	-	9
7	E	8	1	-	-
≀					
10	P	11	2	-	-
≀					

"-" means "no element".

図 5 Aho-Corasick アルゴリズムと 3 ファンクション
Fig. 5 Aho-Corasick algorithm and three basic functions.

態に到達したときに，あらかじめ規定されたパターンがその状態で表現されるときに，そのパターンを示すものである．なお，図中の表では，1 つの状態からの複数の遷移枝を，other 列に記述された一連のリストの形で表現している．

構文解析処理についてもハードウェア化に適した方法を考える必要がある．先に示したような従来方法をハードウェア化してもよいが，我々は意識的にトークン切出し処理で用いた Aho-Corasick マシンを構文解析に対しても利用することを試みた．なぜなら，このような方法は，トークン切出しと構文解析とを同一の扱いで処理できるために，ハードウェアの構成を分かりやすくすることができるからである．このことは，後述する階層化を可能とするという点でも重要な役割を果たす．さらに，ハードウェア設計の観点からも，同種のブロックの繰返し配置で済むため，リソースの数変更や配分変更などに柔軟に対応できるという利点がある．これは，本ハードウェアモジュールを異なる複数の機器に適用するといった場合に有効となりうる．

構文解析に対して Aho-Corasick マシンを利用する方法は，以下のようなものである．構文解析は，そもそも，構文ルールにのったトークンのつながりを判定するものである．見方を変えれば，これはトークンという塊を 1 つの文字として見たてた場合に，トークンという文字列のパターンとの比較判定にほかならない．よって，先の Aho-Corasick マシンを多少変形することにより，構文解析に対しても利用できるようになる．より汎用的に考えれば，このような方法は結果的に複数の階層的な構文をサポートする多階層の Aho-Corasick マシン（階層型 Aho-Corasick マシン，階層型 ACM）を構成することを意味する．

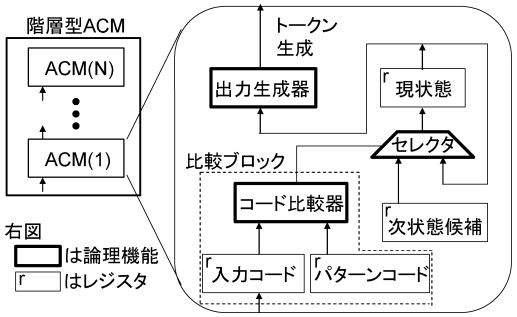


図 6 階層型 Aho-Corasick マシン
Fig. 6 Hierarchical Aho-Corasick machine (H-ACM).

なお，ハードウェアを利用するという観点からは，関連する技術が文献 16) に見られる．これは，XML 処理をハードウェアを使ってオフロード化し，高速化を狙うものであるが，具体的なハードウェア構成について示すものではない．また，XML 限定の処理であり，我々の方法のようにテーブル書き換えによって様々なプロトコルに対応するといったことができない．また，Aho-Corasick 法に着目したものとしては，文献 17) がある．これは，キーワード検索のみを扱う場合に対応し，本稿における 1 階層分の機能に対応する．しかし，Web サービス（特に SOAP によるエンベロープ記述など）で使われるような複雑な構文のルールに対応するためには，階層的なルールで記述されるメッセージの処理が不可欠であり，単にキーワード検索を行うだけでは十分でない．本稿で示すような階層的な構造を有するハードウェアではこのような問題を解決することが可能となる．

従来技術と比較して，本稿の方式は，次節以降で示すように，少ないリソースで動作させることに着目したシンプルな構成をとっていることにも特徴がある．既存技術とのこのような違いは，前提とする利用形態が根本的に異なっていることに由来するものである．

3.4 ハードウェアマッピング

図 6 は，ハードウェア階層型 Aho-Corasick マシンの構成を示したものであるが，できるだけシンプルで，少ないリソースでも実装できるようにすることを目的に開発されたものである．モジュールを構成する主要素は，文字/トークンの比較を行う比較ブロック，状態セレクタ，出力生成器，データ用レジスタ，3 つのファンクションデータを格納するテーブル用メモリ（以下，

3 ファンクションテーブルと呼ぶ），全体制御回路である（後 2 者は図で省略）．

最終的に上記のリソース群は，LSI として SoC, ASIC, FPGA などに実装されうるものであるが，それらの一覧を**表 1** に示す．レジスタや 3 ファンクションテーブル以外の各機能（比較ブロック，出力生成器，セレクタなど）は論理合成後に必要なゲート数が決定するため，本稿の評価の範囲を超えている．しかし，比較ブロックやセレクタなどは，サイズが 8〜16 ビット幅で済むため回路規模も比較的小さくて済むと予想される（8 bit は，ASCII コード，トークン数 256 以下の場合，また，16 bit は 2 バイト文字，トークン数 65,536 以下の場合）．一方，3 ファンクションテーブルについては，マシンの状態数に依存する．具体的なサイズについては，4.3 節に示す．

本方式は，いわゆる汎用のプロセッサではないが，3 ファンクションテーブルの内容を変えることで，同一のハードウェアであっても，異なるメッセージルールを処理することが可能であるという特徴がある．したがって，多様なプロトコルに対応できることはもちろん，使い方によっては，一連の処理の途中でダイナミックにテーブルの書き換えを行って，より少ないリソースで複数の種類のメッセージ処理を行わせるといったことも可能となる．

3.5 処理方式に依存した性能——リソース間トレードオフ

ハードウェアを構成する際に，比較ブロックの数に応じて方式を分類することが可能である．そのような分類の両極端のケースが，完全パラレル方式と完全シリアル方式である．完全パラレル方式は，1 つの状態からの遷移分岐数の最大数と同じ数の比較ブロックが用意されている場合に相当する．一方，完全シリアル方式は，比較ブロックは 1 つだけ用意し，遷移分岐のそれぞれに対応した比較は逐次的に行うものである．

実際のアプリケーションでは，コストに応じて用意できるリソース数（比較ブロックの数）が変化するため，両方式の中間的なアプローチをとることが要求される．このような処理方式による性能の違いについては，4.3 節において実験結果とともに考察する．

3.6 マシン構造を利用した最適化

階層型 Aho-Corasick マシンの構築には自由度がある．すなわち，ある状態から次の状態に遷移する場合に，複数の入力に対応して遷移の枝（以下，エッジ）が複数存在するが，これらをどのような順番で配置するかに関しての自由度がある．

部分的にでもシリアル方式を採用する場合，このようなエッジに関する順番（以下，エッジオーダ）が変化すると，処理性能も変化する．このことを利用して，入力の遷移条件を調べる際に，遷移確率が高い入力ほど，その枝の優先度を高くすることにより，様々な入力パターンに対して総合的な意味での性能を向上させることが可能となる．

また，サポートする仕様項目の取捨選択により，当然ながらマシン構造が変化する．このようなマシン構造の変化を利用した最適化についても，実験結果をふまえて 4.3 節で再考する．

4. 実験と評価

4.1 評価対象と方法

Web サービスの機能に対して，提案方式を用いた場合の処理性能を，本研究のために開発したシミュレータにより検証した．具体的には，単一のテキスト引数

表 1 リソース一覧（1 階層）
Table 1 List of resources (for one layer).

		処理幅 (bits)	個数
比較ブロック	コード（文字/トークン）比較器	8	N
	入力コード（文字/トークン）レジスタ	8	N
	パターンコード（文字/トークン）レジスタ	8	N
次状態セレクタ		(8/16) N	1
出力生成器		(8/16)	1
現状態レジスタ		(8/16)	1
次状態候補レジスタ		(8/16)	N
3 ファンクションテーブル		8(5+M)	1
入力 FIFO		8	1
全体制御回路		(8/16)	1

次状態セレクタ，出力生成器，状態レジスタは状態数 $Ns \leqq 256$ では 8 ビット幅 $Ns \geqq 256$ では 16 ビット幅として 8/16 と表記（厳密には $log_2 Ns$ ビット幅で可能）
比較ブロックは ASCII およびトークン数 256 以下で 8 ビット幅の場合
N はパラレル度，M はトークンの最大文字数

表 2 性能–リソース間トレードオフ
Table 2 Trade-off between performance and resource cost.

		完全パラレル	完全シリアル
HTTP	サイクル数	201	1,617
	比較ブロック数	12	1
SOAP	サイクル数	498	6,088
	比較ブロック数	17	1

を受信し，単一の数値情報を返信する RPC を SOAP の規則に則ってメッセージ化し，これに対して，処理サイクル数やリソース量を評価した．

なお，以下でサブセット仕様とは，上記 RPC の処理に必要な最小のトークンセットと最小の構文ルールを用いた場合を示している．他方，フルセット仕様とは，トークンセットについては対象仕様中のすべてのキーワードを考慮したものであり，構文ルールについてはサブセット仕様と同一である．

4.2 処理モジュールシミュレータ

シミュレータはサイクルアキュレットに性能を見積もることができるようにしたものである．1 サイクルで行う処理は，パラレル，シリアル，いずれの方式の場合も，入力文字（トークン）とパターン文字（トークン）のレジスタ挿入，文字（トークン）比較による状態の遷移に対応する．完全パラレル方式はすべてのパターン文字（トークン）候補との比較をいっせいに行うが，完全シリアル方式はパターン文字（トークン）候補との比較を逐次的に行う．

4.3 評価結果と考察

評価は 3 つの観点から行った．すなわち，(1) 処理性能とリソースのトレードオフ，(2) 最適化の方法とその効果，(3) ソフトウェア処理との比較，について評価を行った．

4.3.1 処理性能とリソースのトレードオフ

まず，SOAP メッセージを HTTP で受信した場合に（サブセット仕様），HTTP と SOAP のそれぞれの処理に必要となるサイクル数を調べた（**表 2**）．表には，完全パラレル型と完全シリアル型の場合の総処理サイクル数が示されている．また，それぞれの場合に必要となる比較ブロックの数も示されている．明らかに，性能とリソース数の間には大きなトレードオフが存在することが分かる．上記のような評価をシステム設計の際に事前に行うことにより，アプリケーションごとに存在するコスト制約や目標性能から最適な搭載リソース数を決定することが可能となる．

完全パラレル方式と完全シリアル方式の中間的な方式について，その望ましい設計指針を得るために，Aho-Corasick マシンの各状態からの分岐数の分布を

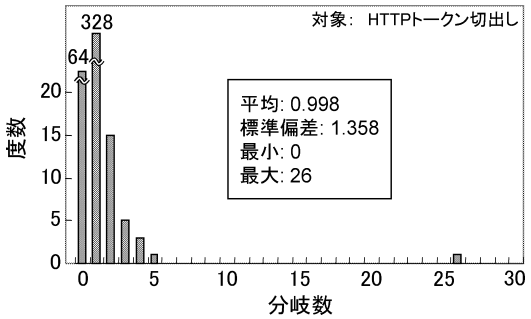


図 7 遷移分岐数の分布
Fig. 7 Distribution of number of transitions.

表 3 マシン構造に依存した処理性能の変化
Table 3 Performance dependency on hardware structure.

	エッジオーダ 1	エッジオーダ 2
サブセット仕様	1,617	1,649
フルセット仕様	3,055	3,239

対象：HTTP トークン切出し

調べた（**図 7**）．分岐数の平均はほぼ 1 であり，また標準偏差は 1.36 程度である．これは，分岐の数が，1 ないし 0（パターン末尾）のものが多く，状態によって大きな分岐が存在している（最大 26）ことに由来する．標準偏差（σ）が比較的小さいことを考えると，パラレル方式を一部利用する場合に，比較ブロック数は比較的少ない数に抑えても，多くの状態をカバーできるため高性能化できることを意味する（2σ 程度，すなわち本ケースでは比較ブロック数 4 で，全状態数の 95%以上をカバーする）．

4.3.2 最適化の方法とその効果

次に，完全シリアル型において，階層型 Aho-Corasick マシンの構造を変化（ここではエッジオーダとサポートする仕様範囲を変化）させた場合に，性能がどのように変化するかについて示す（**表 3**）．エッジオーダに関しては，まったく逆順のエッジオーダである 2 つのケースを比較している．表が示すように，同じ完全シリアル型であっても，エッジオーダによって処理サイクル数に変化が現れることが分かる．事前に，どの入力に対する遷移確率が高いかを予測できる場合には，その情報を活用してエッジオーダの組み換えを行うといったことが有効であることを示している．

さらに，別の重要な点として，仕様のサブセット化の検討があげられる．一般に，Web サービスを利用する機器やシステムを設計する場合に，サービス提供者の意思決定により，仕様で与えられるメッセージルールのサブセット（HTTP のサブ仕様や SOAP のサブ仕様）だけを使うといったことが可能である．このよ

うな場合に，たとえば，トークン切出しの対象となる文字パターンの量を減らすことが可能となるため，階層型 Aho-Corasick マシンの規模を小さくすることが可能である．パラレル型を導入することを考えた場合にも，サブセット化により 1 つの状態からの遷移分枝数が削減されるため，リソース制約を満たしたうえで性能向上を図る際に重要なポイントとなる．

HTTP のトークン切出し処理において，サブセット仕様の場合とフルセット仕様の場合について，3 ファンクションテーブルのサイズを比較した（**表 4**）．これらは，RAM や ROM として実装可能であるが，いずれのセットの場合も，回路規模はサーバなどのコンピュータのメモリと比較すると相当に小さい．

4.3.3 ソフトウェア処理との比較

最後に，ソフトウェアで処理を行った場合[7] と本ハードウェアモジュールを利用して処理を行った場合とで，性能比較を行った結果を示す（**表 5**）．本方式（ハードウェア方式）は，完全シリアル方式および完全パラレル方式の双方を用いた場合について示している．比較は，同一条件となるように，Web サービス階層の受信メッセージ処理の時間だけを抽出したものに対して行った．本実装の結果では，得られる数値はサイクル数である．よって，まず，文献 7) での結果をサイクル数に換算した上で比較を行った．さらに，処理時間の絶対値を得るために，サイクルタイム（周波数の逆数）を 2 種類に変化させた場合について示した．サイクルタイムは，半導体のテクノロジノードやモジュールの実装方法により変化するが，現在（65〜90 nm ノード）の値としては，保守的な値を用いた．

結果によれば，本方式では，従来のソフトウェア処理に対して，非常に少ないサイクル数で実行できることが分かる．すなわち，Web サービス階層のリクエストメッセージの解析処理に関して，およそ 2 桁〜3 桁程度の高速化を実現できることが分かる．ただし，通信プロトコル階層以下の処理に 10 数%の処理時間が，また Web サービス階層のレスポンスのメッセージ生成処理に 9%程度の時間がかかっているために，これらの部分が次なるボトルネックとなり，全体としての高速化は 5 倍程度になる．ただし，現状では最適化を行っていない，これら通信プロトコル階層以下の処理や Web サービス階層レスポンス生成処理についても，より高速な方式に置き換えることにより，さらに高速化することが可能であると考えている．

完全シリアル型の場合，処理時間の絶対値に関しては，動作周波数が 240 MHz 相当で 32.1 μs，また 10 MHz 相当でも 775 μs といった時間で Web サービスの受信メッセージ解析を処理できることになる．また，完全パラレル型の場合は，同様の動作周波数の場合でそれぞれ，2.9 μs，66.9 μs である．なお，回路規模が小さいため，より高周波数で動作させることも可能であろうが，本評価では，そのことによる高速化（処理時間短縮）の効果は考慮に入れていない．本研究の目的からは，コストや電力も重要となる．コスト的には，数 10 MHz 以下といった程度の周波数で動作するチップが適用分野として応用範囲を広くできると考えられる．また，電力の観点からも，電源電圧を低く保ち，周波数を抑えた方が好ましいと考えられる．

表 4 3 ファンクションテーブルサイズ
Table 4 Storage size of 3-function tables.

	サブセット	フルセット
状態数	52	416
テーブルサイズ	1.9 KB	15.2 KB

対象：HTTP トークン切出し

5. 結　　論

Web サービスを多様な分野に応用することを目的として，これを処理性能やリソースが著しく小さいコンピュータ上で利用するための処理高速化技術を開発した．

特に，Web サービスの処理上のボトルネックとなるメッセージ解析処理に対し，ハードウェア階層型 Aho-Corasick マシンを提案し，HTTP 処理および SOAP 処理について評価，検討を行った．専用のハードウェア機構を用いることで，従来のソフトウェア処理に比べて 2 桁〜3 桁の性能向上が可能となることを確認した．Web サービス全体としても約 5 倍の高速化が可能となる．

表 5 処理性能の比較
Table 5 Performance comparison between software and hardware processing.

	ソフトウェア処理	ハードウェア処理			
		240 MHz	10 MHz	240 MHz	10 MHz
処理サイクル数	5,746,080	7,705	7,705	699	699
処理時間	23.9 ms	32.1 μs	775 μs	2.9 μs	69.9 μs

ソフトウェア処理は 240 MHz 動作の汎用 CPU を使用

また，本提案の方式では，最適化に複数の方法が存在し，これらを組み合わせることにより，その性能に大きな変化が見られることも確かめられた．今後は，提案方式を詳細実装し，評価，改善を行う予定である．

参 考 文 献

1) http://www.w3.org/2002/ws/
2) GMO 総合研究所：Web サービス最新動向調査，シードプラニング（編）(2003).
3) 村山隆彦，藤崎恵美子，由良俊介，畑島 隆，簗 栄司，唐沢裕明，加藤 順：Web サービス技術によるファンクラブサービス構築の試み，電子情報通信学会総合大会，D17-7 (2003).
4) 松山憲和，大場みち子：道路交通情報サービスを使った複合 Web サービス実証実験，情報処理学会研究報告 DD51-6 (2005).
5) http://www.google.com/
6) http://www.amazon.com/
7) 佐々木靖彦，村山隆彦，多田好克：分野横断アプリケーション統合に向けた Web サービスのプリミティブ化に関する実装と考察，情報処理学会研究報告 DD53-3 (2006).
8) Fielding, R., Gettys, J., Mogul, J.C., Frystyk, H. and Berners-Lee, T.: Hypertext Transfer Protocol — HTTP 1.1, RFC2616, W3C/MIT (1999).
9) W3C: Extensible Markup Language (XML) 1.0 (3rd Edition) (2004). http://www.w3.org/TR/2004/REC-xml-20040204/
10) W3C: Simple Object Access Protocol (SOAP) 1.1 (2000). http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
11) Boyer, R.S. and Moore, J.S.: A fast string searching algorithm, *Comm. ACM*, Vol.20, pp.762–772 (1977).
12) Knuth, D.E., Morris, J.H. and Pratt, V.B.: Fast pattern matching in strings, *SIAM J. Comput.*, Vol.6, pp.323–350 (1977).
13) Wu, S. and Manber, U.: A fast algorithm for multi-pattern searching, Technical Report TR-94-17, Department of Computer Science, University of Arizona (1994).
14) Aho, A.V. and Ullman, J.D.: *Principles of compiler design*, Addison-Wesley Pub. (1977).
15) Aho, A.V. and Corasick, M.J.: Efficient string matching: An aid to bibliographic search, *Comm. ACM*, Vol.18, pp.333–340 (1975).
16) http://www.idealliance.org/proceedings/xml04/papers/246/xml-accel.html
17) http://www.business-i.jp/sentan/jusyou/2004/fbi_ nec.pdf

(平成 18 年 1 月 26 日受付)
(平成 18 年 5 月 29 日採録)

佐々木靖彦（正会員）

1991 年東京大学大学院工学系研究科電子工学専攻修士課程修了．同年（株）日立製作所入社．半導体における集積回路技術および設計技術の研究開発に従事．1998 年から 1999 年にかけて，米国スタンフォード大学客員研究員．2005 年より電気通信大学にて，Web 技術をはじめとする情報システム学に関する研究に従事．IEEE 会員．

村山 隆彦（正会員）

1986 年東北大学大学院工学研究科情報工学専攻博士前期課程修了．同年日本電信電話株式会社入社．現在 NTT 情報流通プラットフォーム研究所勤務．Web サービス，セマンティック Web に関する研究開発に従事．2004 年より電気通信大学大学院情報システム学研究科客員助教授．電子情報通信学会，人工知能学会各会員．

多田 好克（正会員）

1985 年東京大学大学院工学系研究科情報工学専門課程博士課程修了．工学博士．同年電気通信大学電子情報学科着任．1992 年より電気通信大学大学院情報システム学研究科．並列・分散システムの記述法に興味を持ち，オペレーティングシステムをはじめとするシステムソフトウェアの実現法に関する研究に従事．ACM，電子情報通信学会各会員．