社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2005－ＡＶＭ－51（1
2005／12／1

高位レイヤの高速処理を実現する H/W 実装方式に関する一検討

水口　潤

三菱電機株式会社 情報技術総合研究所 〒247-8501 神奈川県鎌倉市大船 5-1-1

E-mail: Mizuguchi.Jun@bc.MitsubishiElectric.co.jp

あらまし　近年のネットワークは、レイヤ 2～レイヤ 4 による単純なルーティング機能だけでなく、より高度なサービスを、求められる品質で柔軟に提供することが求められてきている。しかし、従来レイヤ 5 以上の上位レイヤ処理は、S/W 処理をベースとしているため、高速大容量の通信システムに展開する上で処理能力がボトルネックとなり、H/W 処理をベースとした上位レイヤ高速処理技術の確立が必要となってきている。このため H/W 処理によるアクセラレーション、及び H/W 処理と S/W 処理の機能分割による上位レイヤ処理の高速化実現が課題である。本稿ではこれらを実現するための方式について提案する。

キーワード　ハッシュ, 並列化, 文字列検索

A study of hardware mounting method which achieve high-speed process on high-layer.

Jun Mizuguchi

Information Technology R&D Center, Mitsubishi Electric Corporation 5-1-1 Ofuna, Kamakura, Kanagawa, 247-8501 Japan

E-mail: Mizuguchi.Jun@bc.MitsubishiElectric.co.jp

Abstract　The network of recent years has been requested to provide not only a simple routing function by Layer2～Layer4 but also more advanced service flexibly by the requested quality. But, As for the high-layer processing of Layer5 or more, the processing performance becomes a bottleneck in developing the S/W processing with the telecommunication system for a high-speed large capacity because it is a base, and the establishment of the high-layer high speed processing technology based on the H/W processing is needed so far. Therefore, the speed-up achievement of the high-layer processing of Acceleration, the H/W processing, and the S/W processing by the H/W processing by the function division is a problem. It proposes the method to achieve these in this text.

Keyword　Hash, Parallelization, String Searching

1. はじめに

通信端末の高性能化、またそれらに実装され、各サービスを提供するアプリケーションの出現に伴い、ネットワークを構成する装置に対する高機能化、高速大容量化といった要求が日々高まってきている。それに伴い、従来のネットワーク装置のような、レイヤ 2～レイヤ 4 プロトコルによる単純なルーティング機能のみの実装から、近年ではルーティング機能はもちろんのことレイヤ 5 以上のテキストベースのプロトコルに対する識別機能まで実装されつつある。しかし、それら上位プロトコルに対する識別機能も、多くのネットワーク装置では、機能実装が容易な汎用プロセッサ上で動作するソフトウェアをベースとしたものが多く、高機能化を実現できたとしてもソフトウェア実装の弱点である高速大容量化という要求を満足できていない。そこで本稿では、文字列識別機能を汎用プロセッサ上で動作するソフトウェアで実装するのではなく、ハードウェア上に実装することでソフトウェアと比較し高速大容量化を図れる方式について述べる。

2. 文字列照合機能

文字列照合機能は、レイヤ 5～レイヤ 7 までの各プロトコル固有のコネクションを識別可能な文字列（識別子）の照合をおこなう機能である。

この文字列照合機能を実現するためには、本方式では以下の処理をおこなう。（図 1 参照。）

① データ内において文字列が記述されているヘッダフィールド位置を特定し文字列の抽出処

理、および文字列のハッシュ化

② ハッシュ化した文字列に対しての照合処理

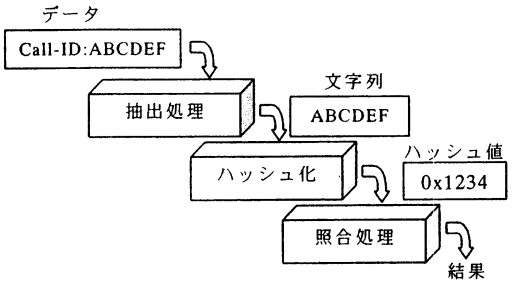


図 1 文字列照合機能

以下では、各処理ブロックにおける処理内容について説明する。

3. 文字列照合機能を実現するための処理

3.1. 抽出処理

抽出処理は、データ内におけるヘッダフィールドを示す特定文字列の照合処理をおこない、文字列が記述されているデータ位置の特定と、文字列の抽出を目的としている。(図 2 参照。)

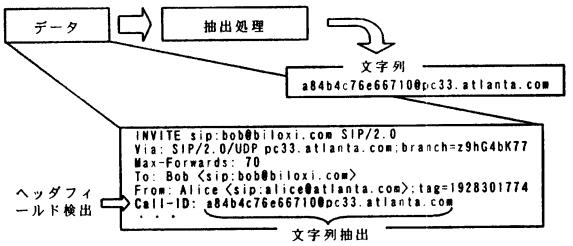


図 2 抽出処理

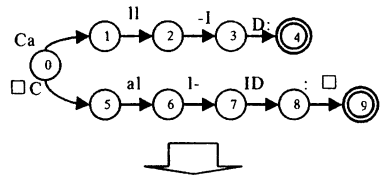
データ位置の特定には、各文字列を任意文字ずつ分割し、並列化処理を考慮して登録したオートマトンを使用し、また入力時における分割位置を考慮し分割位置が異なるパターンも登録される。(SIP のヘッダフィールド"Call-ID:"の登録例。図 3 参照。)

各ヘッダフィールドは、文字列の長さと比較して短いため、文字列そのものをオートマトンに登録する場合と比較しメモリ量が少なくて済む。さらにこの検出部を、並列に展開することによって検出処理を高速化する。(図 4 参照。)

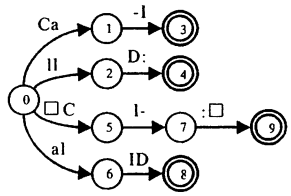
図 4 では、1 文字あたり 1Byte である文字を 4 Byte ずつ入力し、そのうち上位、下位 16Bit をそれぞれ異なる検出部に入力して並列処理をおこないヘッダフィールド検出をおこなっている例を示す。

アドレス部では、入力された文字と状態からメモリアドレスを計算する。メモリ部では、アドレス部から入力されたメモリアドレスをリードし、遷移情報を出力する。遷移情報には、次の遷移状態番号、ヘッダフィールド番号、分割されたヘッダフィールドの何番目かを示す分割順序番号、ヘッダフィールド分割数が記載されている。

(a) 分割位置を考慮して登録したオートマトン



(b) 文字の分割数を考慮して再構成されたオートマトン



(c) パイプライン処理＆並列化処理を考慮して登録したオートマトン

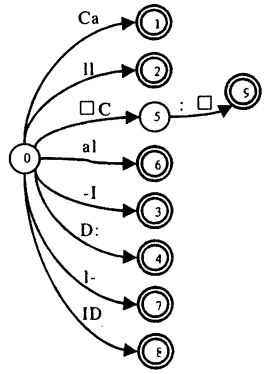


図 3 オートマトン例

※二重丸で示された状態では、次の入力に対する遷移が初期状態 0 から始まることを示す。また"□"は、スペースまたは、改行コードを示す。

カウンタ部では、遷移情報中にあるヘッダフィールド番号と分割順序番号を監視する。図 4 を用いて上記動作を補足する。Clock2 では、データ"Call"が入力されたところであり、入力データの上位("Ca")、下位("ll")を初期状態番号(例では"0")とともにそれぞれ別のアドレス部に入力する。Clock3 では、次のデータ"-ID:"が入力され、Clock2 と同様に処理される。

一方 Clock2 で入力されたデータは、アドレス部にて

リードするメモリアドレスが計算され、メモリ部に渡される。Clock4 では、次のデータ"□ABC"がアドレス部に入力され、メモリ部では Clock3 のデータによるメモリアドレスが計算される。またメモリ部では、Clock2 で入力されたデータによる遷移結果が出力され、カウンタ部に入力される。

カウンタ部では、分割順序番号および、同じヘッダフィールド番号で、かつ前の分割順序番号と連続している番号が通知された時にカウントアップをおこない、その値がヘッダフィールド分割数に達した場合には、ヘッダフィールドに一致したと判定する。

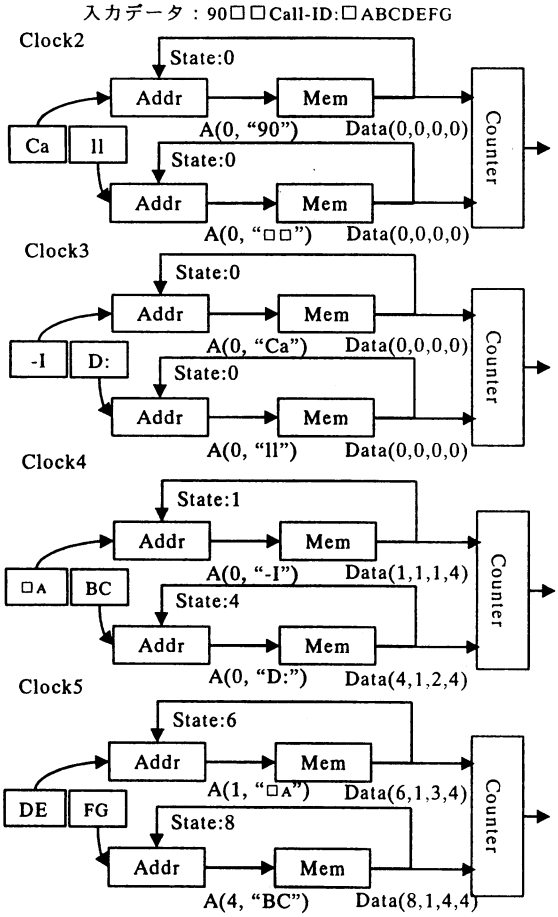


図 4　パイプライン化と並列処理動作例

※ A(状態番号, 入力文字列), Data(次の遷移番号, ヘッダフィールド番号, 分割順序番号, ヘッダフィールド分割数)を示す。

3.2. 照合処理

照合処理は、抽出処理で得た文字列をもとにした照合処理である。ところで、SIP ヘッダの一つである"Call-ID:"は、グローバルに一意な識別子となるようにランダムな可変長の文字列となっている。また近年のセキュリティ事情もあり文字列長が長くなる傾向にある。このため文字列を無加工で保持するには、メモリ使用量を圧迫してしまう。そこで文字列にハッシュを適用し、ランダムな可変長文字列を固定長のハッシュ値に圧縮し、そのハッシュ値をメモリ上に保持する方法が考えられる。これより、無加工の場合と比べてメモリ量を削減することができる。

3.3. ハッシュ値の衝突

しかし、ハッシュ適用による弊害として 1. 検索キーの衝突、2. エントリ追加時における衝突というハッシュ値の衝突に起因する問題が発生する。ここで、ハッシュ値の衝突とは、元の文字列は異なるが、ハッシュにより圧縮されたことで、同じハッシュ値が生成されることを指す。(図 5 参照。)

図 5　検索キーの衝突

そこで次に、このハッシュ値の衝突にともなう問題と解決方法について示す。

3.3.1. 検索キーの衝突

検索キーの衝突とは、検索キーとなる文字列は異なるがハッシュ関数により検索キーとなる文字列が圧縮されたことにともない、同じハッシュ値が生成され、その結果テーブル検索で誤一致が発生することを指す。(図 6 参照。)

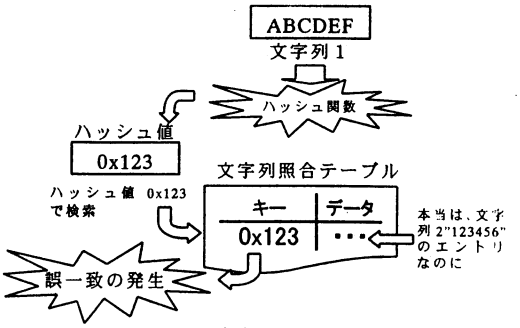


図 6　検索キーの衝突

この誤一致発生が起こらなくするためには、ハッシュ適用前の文字列をテーブルに保持し、ハッシュ値との照合一致後、再度テーブルの文字列とハッシュ適用前の文字列との間で照合をする必要がある。だが前述したように文字列を無加工で保持すると文字列長とエントリ量に比例したメモリ量を消費する。そこで、誤一致の発生確率を実用上問題のないレベルまでにするた

めに、文字列を特徴づける文字列特徴情報（ハッシュ値、文字列長等）をテーブルに保持し、文字列特徴情報とも照合することで誤一致発生確率を低くする。(図7参照。）ここで必要なビット幅については後述する。

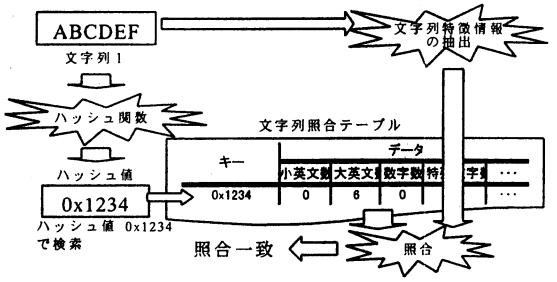


図 7　検証処理

3.3.2. 新規登録エントリの衝突

新規登録エントリの衝突とは、コネクション開始と判定されたフローを識別するための文字列をテーブルに登録する際、登録する文字列をハッシュ関数によって圧縮したことにともない、既存のテーブルに登録されているハッシュ値と一致してしまうことを指す。（図8参照。）

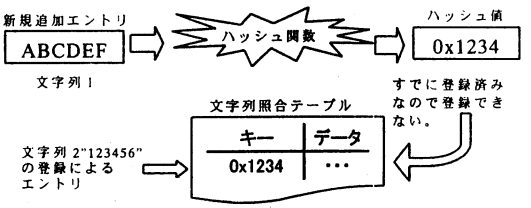


図 8　登録エントリの衝突

このため、ハッシュ値衝突を解決するための手段が必要となる。そこで、ハッシュ値の衝突回数毎にテーブルを分けることで、ハッシュ値衝突を解決する方式を提案する。（図9参照。）

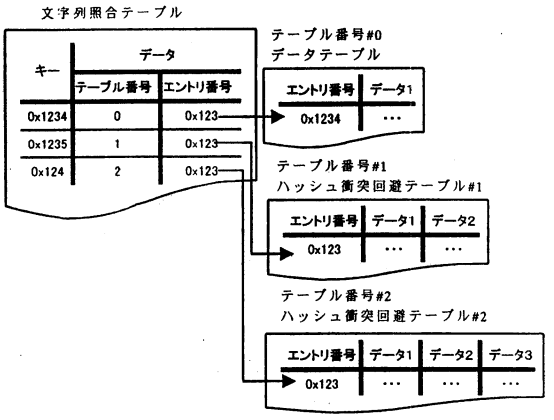


図 9　テーブル構成

文字列照合テーブルでのテーブル番号は、検索キーであるハッシュ値に対して何回衝突したかを示している。また、エントリ番号は各テーブル中で一意となる番号であり、データへのポインタを示している。

このように、衝突回数毎にテーブルを分離することで、①最大衝突回数のエントリのデータ量にあわせずに済むためメモリ量を節約できる。②必要なデータ分だけバースト転送できる。という利点がある。

また、後述の見積もりにより、ハッシュ値衝突回数が求められるため、各テーブルの総エントリ数を予測することができる。

4. 見積もり

4.1. ハッシュ値衝突確率

4.1.1. エントリ登録時

まず、エントリ登録時におけるハッシュ値の衝突確率を見積もった。本稿では、文字列照合の最大数を1Mエントリとして見積もった。また、ハッシュ値の衝突確率計算には、ポアソン分布（Poisson distribution）を用いた。[1]

ここでポアソン分布の密度関数は式(1)のように求めることができる。

$$f(x)=\frac{e^{-\lambda}\lambda^{x}}{x!}\ (\text{ただし, } x=0,1,\cdots)\ \cdots(1)$$

ここでλ=生起確率×エントリ数であり、Hをハッシュビット幅、Nをエントリ数とすると、$\lambda=2^{N-H}$となる。式(1)より最大数のエントリを登録した場合に，あるハッシュ値にエントリがない確率は$f(0)$、1個だけエントリがある確率は$f(1)$、エントリが2個あるのでハッシュ値の衝突が発生する確率は$f(2)$と表される。

上記式を用いて、文字列照合テーブルの検索キーであるハッシュ値のビット幅を24ビット、最大エントリ数を1Mエントリとした時のハッシュ値衝突確率を表に示す。（表1参照。）

表 1　ハッシュ値衝突確率

衝突回数	エントリ数	確率
0	985046	93.94
1	61565	5.87
2	1924	0.18
3	40	0.004
⋮	⋮	⋮

これにより、ハッシュ値の衝突を解決しなければならない各テーブルのエントリ数は、以下の表のように求めることができる。（表2参照。）

表 2　各テーブルの総エントリ数

テーブル名	エントリ数
文字列照合テーブル	1M
データテーブル	0.94M
ハッシュ衝突回避テーブル#1	30K
ハッシュ衝突回避テーブル#2	1K
ハッシュ衝突回避テーブル#3	10
⋮	⋮

4.1.2. 検索時

次に検索時におけるハッシュ値衝突確率を見積もる。見積もりでは、回線速度が 10Gbit/s について見積もった。10Gbit/s におけるショートパケット（64byte の最短フレーム。Inter Frame Gap と Preamble を含めると 84byte）の packet/s は以下のようにして求められる。

10G[bit/sec]÷(84[Byte/packet]×8[bit/Byte])
≒ 14.881M[packet/sec] …(2)

これにより 1M エントリに対して T[sec]間に試行がおこなわれる回数は、以下のようにして求められる。

14.881M[packet/sec]×T[sec]×1M[times/packet]
≦ 2^{44}×T[times] …(3)

上記のように、文字列照合テーブルの検索キーであるハッシュビット値は 24 ビットであるため、(3)の結果から、

2^{44}×T[times]÷2^{24} = 2^{20}×T[times] …(4)

と、T[sec]では文字列照合テーブル全体で 2^{20}×T[times]もの誤一致が生じる。そこで、前述した誤一致発生確率を低くする文字列特徴情報のビット幅をこの結果を元に決定する。仮に T[sec]を 100 年と決定するならば、

60[sec/minute]×60[minute/hour]×24[hour/day]×
365[day/year]×100[year] ≦ 2^{32}[sec] …(5)

よって、100 年の間に 2^{52} 回試行がおこなわれると求められる。次にベルヌーイの試行から、試行回数が 2^{52} 回と多いため大数の法則に従うと仮定した。これにより 1 エントリあたりのハッシュ値衝突による平均誤一致回数 E(x)を求める。

E(x)=np=2^{52}×(1/2)^{24}=2^{28}[times] …(6)

これより、(6)の結果と表 1 と同じ式から再度、各ハッシュビット幅における衝突確率を求める。
（表 3 参照。）

表 3　ハッシュビット幅毎の衝突確率

衝突回数	確率		
	32ビット	40ビット	48ビット
0	93.9413	99.9756	99.9999
1	5.8713	0.02441	0.000095
2	0.1835	0.000003	0.00000000005
⋮	⋮	⋮	⋮

上記表 3、およびビット幅増大によるメモリ量の増加を鑑みて、本稿では文字列特徴情報ビット幅を 48 ビットと設定した。

4.2. メモリ量

1 エントリ当りのデータ幅を 16Byte と仮定すると、文字列特徴情報 48 ビット(=6Byte)を含め、各テーブルの必要メモリ量は以下のように求められる。

- 文字列照合テーブル
  1M[entry]×4[Byte/entry]= 4M[Byte]
- データテーブル
  0.94M[entry]×22[Byte/entry]≒20.7M[Byte]
- ハッシュ値衝突回避テーブル#1
  30K[entry]×{22[Byte/entry]×2}≒1.3M[Byte]
- ハッシュ値衝突回避テーブル#2
  1K[entry]×{22[Byte/entry}×3}≒0.07M[Byte]
- ハッシュ値衝突回避テーブル#3
  10[entry]×{22[Byte/entry]×4}≒0.01M [Byte]

これにより、総メモリ量は 26Mbyte 程度になることが見積もられる。

5. まとめ

本稿では、文字列照合機能をソフトウェアではなく、ハードウェア上で実装する場合の検討をおこなった。それにより、高速な抽出処理、および 1 M エントリものの大容量の文字列照合がハードウェア上で実現可能である見通しをつけた。この文字列照合機能は、負荷分散装置等に応用することができると期待できる。このため今後、シミュレーションによる性能、機能評価をおこない、次世代ネットワークプラットフォームとして適用していきたいと考えている。

文　献

[1] 平野幸男, 東方敦司, 伊藤修治, “ネットワークプロセッサにおけるテーブル検索時間の考察”, 信学技報 CS2003-167, pp. 55-60, 2004 年 3 月