アクティブネットワーク技術を用いた多重名前空間の実現法

東村邦彦[†] 加藤和彦[††,†††] 松原克弥[††]

複数のユーザが協調操作を行う分散協調処理環境では，各資源に対して分散透明かつ一貫した名前付けを行いつつ，ユーザごとに柔軟に名前空間の再構成を行えることが求められる．本論文では，単純な単一名前空間を多重化することにより，系統的な方法で名前空間をカスタマイズ可能にする多重名前空間を提案する．また，その名前解決を高速化するためのキャッシュ技術について述べる．この技術の特徴は，名前解決クライアント・サーバの両端点でなく，その通信経路上の機器で透過にキャッシングを行う点にある．本論文では，提案システムのプロトタイプの実装および実験を行い，サーバへの負荷，ネットワークのトラフィックと探索パケットのホップ数の削減が可能であることを示した．

Implementing Multiple Name Spaces Using an Active Network Technology

Kunihiko Toumura,[†] Kazuhiko Kato[††,†††] and Katsuya Matsubara[††]

In distributed cooperative processing environment, we need consistent, location-transparence naming scheme, while a user can reconfigure name space flexibly. In this paper, we propose a novel naming scheme that can systematically customize a name space by multiplexing single namespaces. And we also propose a caching technique for efficient name resolution. With the scheme, name-resolving information is cached in the machines on the communication path between a server and a client. We implemented a prototype system. The experimental result shows that the proposed system can reduce server load, network traffic and hop count of request packets.

1. はじめに

広域ネットワーク上に散在する各種の資源を指し示すために，Domain name system（DNS）[5),6)] や Uniform resource identifier（URI）[1)] などが利用されている．これらの名前付けに共通していえることは，単一でグローバルな名前空間が構成されているということである．この性質により，場所によらず一定の名前で世界中から資源を指し示すことが可能となっている．しかし，単純な単一の名前空間では，機能不足となる場合がある．そのような場合の 3 つの典型的な例を以下に示す．

第 1 の例は，異機種分散環境に起因するものである．広域ネットワーク上に存在する各種の資源は，プログラムであれば計算機アーキテクチャの違い，文書であれば言語・フォーマットの違いなどがある．このような場合，アーキテクチャや言語が異なっていても，その名前は単一として透明性を達成したいが，ユーザの望む資源の実体が参照できるようになっている必要がある．

第 2 の例は，計算機を用いた設計作業に関するものである．このような設計作業には，しばしば試行錯誤をともなう．その際に設計者グループなどを単位として，ファイルの修正などを実験的に行う必要が生じることがある．

第 3 の例は，利用する資源がユーザからは操作できないが，一時的に他の資源で置き換えたい場合である．たとえば，CD-ROM に代表される読み出し専用媒体や，書き換え権限を持っていない他人のファイルなどに個人的な annotation を加えたい場合などがこの場合にあたる．

本論文では，単一でグローバルな名前空間を提供しつつ，名前空間を多重化する方式について論じる．多

† 筑波大学大学院博士課程工学研究科
Doctoral Program in Engeneering, University of Tsukuba
†† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
††† 科学技術振興事業団
Japan Science and Technology Corporation

くの名前空間においては，単一の論理資源名を単一の物理資源名に写像することに加え，複数の論理資源名を単一の物理資源名に写像することにより，ある単一の資源に対し複数の視点からみた異なる論理資源名を与えることができる．それらの写像の方法に加え，本提案方式では，1 つの論理資源名を複数の物理資源名に写像することができ，かつ写像の方法を名前解決ごとに変更することを可能とする．

多重化された名前空間は，各名前空間の和としてユーザに示される．そのため，各名前空間はより下位に存在する名前空間との差分のみを保持すればよく，名前空間の多重化にともなう記憶域利用効率の低下は最小限に抑えることが可能である．

広域ネットワーク中で単一の名前空間を共有するためには，名前解決を行うために必要なサーバは地理的に分散することになる．そこで，問合せを行う際には，サーバへの負荷の軽減や，解決にかかる時間の短縮の観点から，キャッシュ技術などをもちいて無駄な問合せを避けることが重要である．

従来，キャッシュはサーバ側やクライアント側など，問合せのための通信経路の両端点で行うことが普通であった．しかし，近年，通信経路上に存在する機器において通過するパケットに対して処理を行わせるという，PLAN [3]，ANTS [10]，NetScript [11] に代表されるアクティブネットワーク研究が出現し，ルータなどのネットワーク経路上のマシンに何らかの処理を行わせることが考えられるようになってきている．

また，現在でも通常のワークステーションや PC をルータとして使用しているサイトや，専用のルータ上に Web のキャッシング機能を導入した製品などもあり，ルータ上で何らかの処理を行うことは現実的なものといえる．

提案方式では，名前解決を行うために必要なパケットが通る経路上に存在するルータなどの上に簡単なキャッシュ機構を導入し，名前解決の効率化を図る．名前探索システムの両端点にキャッシュ機構を埋め込むのではなく，途中経路でキャッシュすることには，以下の利点がある．第 1 に，ネットワーク上で透過的にキャッシングを行うため，端点にあるサーバとクライアントは名前解決のアルゴリズムだけを純粋に記述するだけでよくなり，プログラミング上の利点が得られる．第 2 に，キャッシュする場所を，サーバ・クライアント間の経路上で任意に設定できることにより，サーバに近い場所にキャッシュを置くことで一貫性を細かく制御したり，クライアントに近い場所に置くことで，レスポンスを向上させたりできるなど，キャッシュする内容によって柔軟な対処を行うことが可能となる．

以下，2 章で多重名前空間の詳細，3 章で効率的な名前解決の方式，4 章で実装状況と実験，5 章で関連研究について述べ，6 章でまとめる．

2. 多重名前空間

本章では，グローバルに単一な名前空間を提供しつつ，状況に応じて動的に名前空間の構成を変更可能とする**多重名前空間**の構成方式を提案する．

1 つの名前空間は，ドメインネームや Unix のファイルシステムのような木構造をしており，グローバルで単一な名前付けが行われるものとする．提案方式では，そのような名前空間の存在を複数許し，それら複数の名前空間を重ね合わせることで 1 つの名前空間を仮想的に構成する．この仮想的に構成された名前空間のもとで，指定された資源名の名前解決を行う．ここで，各名前空間の名前は，グローバルで一意な名前付けが行われており，その名前空間の構造は特に構造を持たないフラットな名前空間であるとする．名前空間を重ね合わせる際に，それらの空間の間の全順序を指定する．この全順序の指定を**ビューパス**（view path）と呼び，N_i $(1 \leq i \leq m)$ をそれぞれ重ね合わせる名前空間の名前として，$\{N_1 \rightarrow N_2 \rightarrow \ldots \rightarrow N_m\}$ のように表記する．ここで指定するビューパスは，名前空間のユーザが名前指定のたびに指定する．ユーザは，ビューパスを適切に変更することで名前空間のカスタマイズを行うことが可能である．ユーザに見える仮想的な名前空間は，N_1 から N_m に至るまでのビューパス内に存在する m 個の名前空間の和をとったものとなる．ここで，「名前空間の和」とは，それぞれの空間に存在する論理資源名の集合の和として定義される．ただし，同一の論理資源名がビューパス内の複数の名前空間に存在している場合は，ビューパスのより上位に存在する論理資源名が選択され，選択された論理資源名と物理資源名の対応に基づいて名前解決を行うものとする．

名前空間に名前を追加するときは，ユーザが指定したビューパスの最上位の空間に名前が追加され，名前を削除する際には，ビューパスのより上位に存在する空間上の名前が削除される．

提案方式では，ユーザが資源名を指定する場合，必ずビューパスと資源名の組を指定する．たとえば，**図 1** に示した多重名前空間の場合，ユーザがビューパスを $\{\alpha \rightarrow \beta \rightarrow \gamma\}$ として，資源名/b/f を指定するときは，組

$$(\alpha \rightarrow \beta \rightarrow \gamma,\ /\mathsf{b}/\mathsf{f})$$

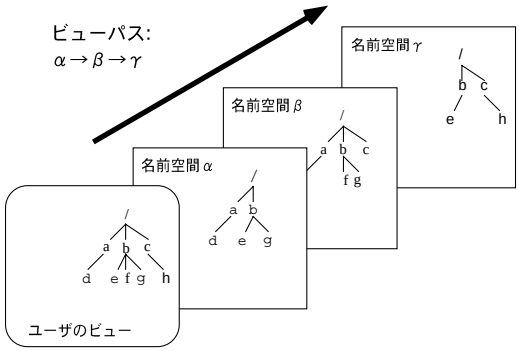

図 1 多重名前空間の例
Fig. 1 Example of multiple name space.

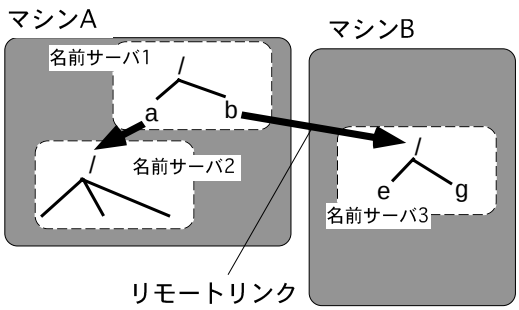

図 2 多重名前空間のサーバ構成
Fig. 2 Server configulation of multiple name space.

を指定する．この例の場合，名前解決の結果，名前空間 β の資源/b/f が選択されることになる．

提案方式では，上位の名前空間に存在する名前で下位の名前を隠蔽することによって名前空間のカスタマイズを行うため，同一ビューパス内に現れる各名前空間に配置する名前に関して同一の名前付けのルールが存在することを仮定している．名前付けに関するルールを重ね合わせた場合，それぞれのルールの違いから名前付けに関するルールが名前空間内で混在することになる．名前付けに関するルールは，論理的に同じ資源を同じ名前で表現するためには必要不可欠なものであるため，同一のルールが存在することを仮定することは多重名前空間の使用範囲を限定するものにはならないと我々は考える．

多重名前空間を構成する各々の名前空間は，システム内の複数のマシン上に存在する**名前サーバ**と，その間を結ぶ**リモートリンク**から構成される（**図 2** 参照）．各名前サーバには，名前空間の一部が保存される．リモートリンクは，各名前サーバに保存されている名前空間の続きにあたる名前空間へのリンクを保持している．リモートリンクの情報はシステム管理者が行うもので，ユーザはリモートリンクの存在を意識する必要はない．

多重名前空間を用いた名前空間のカスタマイゼーションは，大きく 2 つの方法に分けられる．第 1 の方法は，ビューパスを構成する名前空間を選択することにより，ユーザから見える名前空間を変更することである．第 2 の方法は，既存の名前空間に対して，ユーザの個人的な名前空間を上位に重ね合わせることにより，下位の名前空間内の資源を変更することなく，ユーザ独自の資源への変更処理を行うことによる．

第 1 の方法を用いることにより，1 章で述べた単一名前空間の欠点の第 1 の例である異機種分散環境での問題は以下のように解決できる．まず，機種（あるいは言語）依存性のない資源と機種依存性のある資源を別々の空間に分けておく．仮に，機種依存性のない資源の存在する空間名を “Neutral”，機種依存性のある空間名をその機種名を用いて “Arch-A”，“Arch-B” として生成する．ここで，各名前空間において名前の使用法に関する一定の convention があることを仮定する．すなわち，論理的に同一の資源を指し示す名前に対しては機種依存・非依存にかかわらず同一の名前が使用されることを仮定している．ユーザは，機種依存性のない空間の上位に各ユーザが使用する機種に適した空間を置くようにビューパスを設定する．たとえば，機種 Arch-A を使用するユーザは，ビューパスを {Arch-A → Neutral} と設定する．また，機種 Arch-B を使用するユーザは同様に {Arch-B → Neutral} と設定する．このビューパスの指定により，各機種のユーザは同一の論理名を使用しつつ，かつ適切な機種依存性のある資源を指定することが可能となる．提案方式である多重名前空間を利用しない場合は，論理資源名の中に機種情報を埋め込むことで機種依存のある資源を識別する（“/usr/Arch-A/bin” などの名前の利用）必要や，まったく別の名前空間を機種ごとに用意し，機種ごとにそれぞれの名前空間を使用するという ad hoc な方法を用いることになる．それらの方法には，前者では論理的に同一な資源が異なる名前で参照する必要があること，後者では機種依存性のない資源であっても複数の名前空間に同一のコピーを置く必要があることなどの欠点がある．提案方式では，単一な名前空間をそれぞれ用意するのではなく，重ね合わせとして名前空間を表現するため，機種依存性のない資源に関して複数の名前空間にコピーを置く必要はなく，機種依存性のない資源に関する管理のオーバヘッドの増大を防ぐことが可能である．

第 2 の方法を用いることにより，1 章で述べた単一

名前空間の欠点の第2の例である計算機を用いた設計作業に関する問題は，以下のように解決できる．まず，設計中のシステムで使われているビューパスの指定が $\{N_1 \rightarrow N_2 \rightarrow \ldots \rightarrow N_m\}$ となっていたとする．設計者は，一時的な変更を加えるためにこのビューパスの最上位に設計者の生成した名前空間 U を付加する．すなわちビューパスは $\{U \rightarrow N_1 \rightarrow \ldots \rightarrow N_m\}$ となる．このビューパスによって構成された名前空間に対し，設計者は変更を行う．設計者の行った変更は，最上位の名前空間である U に対して行われるため，設計中のシステムに対し変更を行うことなく，実験的な変更を行うことが可能である．設計した内容を設計中のシステムに永続的に反映するためには，名前空間の間でファイルのコピーを行う．提案方式では，名前解決ごとにビューパスの指定を変更することが可能である．よって，コピー元のファイル名の指定時のビューパスを設計者の名前空間 U とし，コピー先のファイル名の指定として N_1 から N_m の適切な名前空間を指定し，実験的な変更内容をシステムに反映することが可能である．1章で述べた単一名前空間の欠点の第3の例も同様に，ユーザの生成した名前空間を最上位に指定することで，オリジナルの資源に影響を与えることなくユーザ独自の変更を行うことが可能となる．

3. 名前解決の効率化方式

本章では，2章で述べた多重名前空間の構成方式を効率的に実現する方法について述べる．まず，3.1節で多重名前空間の単純な実現方式について述べ，その後3.2節でアクティブネットワークを利用した実現法を検討し，3.3節で単純な実現方式を改良した実現方式の詳細について述べる．

3.1 多重名前空間の単純な実現方式

2章で述べた多重名前空間の単純な実現方式は以下のとおりである．

クライアントからの要求

クライアントの要求するサービスに対応して，サーバに対して送るメッセージは**表1**に示すものになる．ここで，返却値 OK は要求に対する動作が行われたことを示し，返却値 LINK は，その名前がリモートリンク先にあることを示す．また返却値 NOTFOUND は探索した名前が存在しないことを示し，返却値 ERROR は何らかのエラーが発生したこと示す（すでに存在する名前を使って名前の追加要求を行った場合など）．以下，返却値を含んだパケットを返答パケットと呼ぶ．

クライアントは，表1に示されるメッセージを使用し，サーバへ要求を行う．この際，サーバの位置情報に関してクライアントは名前空間のルートサーバの位置情報のみを知っており，他のサーバに関する情報は，問合せの際にリモートリンク先としてサーバの位置情報を得る．サーバから返却値 LINK が返された場合はそのリモートリンク先に要求メッセージを送ることを繰り返す．そして，最終的に返却値 OK, NOTFOUND, ERROR のいずれかを得て，要求を完了する．

表1 クライアントからの操作要求
Table 1 Request from resolver client.

メッセージの種類	用途/返却値
ADDNAME	名前の追加 OK, LINK, ERROR
ADDLINK	リモートリンクの追加 OK, LINK, ERROR
LOOKUP	名前の探索 OK, LINK, NOTFOUND
DELNAME	名前・リモートリンクの削除 OK, LINK, ERROR

サーバの動作

クライアントからの要求に対するサーバの動作を**図3**に示す．

動 作 例

図2に示される単一名前空間内で，/b/g を探索した場合の通信内容について述べる．まずクライアントは，名前サーバ1へ/b/g の探索要求メッセージ LOOKUP を送信する．これに対し，名前サーバ1は/b 以下のファイルに関しては名前サーバ3を参照せよとの情報を持った返却値 LINK をクライアントに返す．そしてクライアントは改めて名前サーバ3に/g の探索要求メッセージ LOOKUP を送信する．最後に名前サーバ3は，この名前が存在することを示す返却値 OK とその物理資源名を返す．

多重名前空間に対する操作は，ここまでに述べた単一名前空間に対する操作をビューパスに指定された空間に対し順次実行することで行う．名前の追加は，ビューパスの最上位の空間に対して行うので，ビューパスの最上位の空間に対してのみ上記の名前追加の操作を行う．名前の探索は，ビューパスの最上位の名前空間から順に各名前空間内で探索を行い，最初に名前が見つかった空間で探索を中断し，その空間での論理資源名に対する物理資源名をユーザが求めるものとして返答する．また，名前の削除は，名前の探索と同様にビューパスの最上位の名前空間から順に各名前空間内で削除を行い，返却値 OK（あるいは ERROR）が返されるまで名前の削除を繰り返すことによって行う．

```
switch (メッセージの種類) {
case ADDNAME:
case ADDLINK:
  if (リモートリンク先で管理) {
    LINK(リンク先サーバ, リンク先で問い合わせる名前);
  } else if (名前が存在) {
    ERROR();
  } else {/* 名前が存在しない */
    名前/リンクを登録;
    OK();
  }
  break;
case LOOKUP:
  if (リモートリンク先で管理) {
    LINK(リンク先サーバ, リンク先で問い合わせる名前);
  } else if (名前が存在) {
    OK(物理資源名);
  } else {/* 名前が存在しない */
    NOTFOUND();
  }
  break;
case DELNAME:
  if (リモートリンク先で管理) {
    LINK(リンク先サーバ, リンク先で問い合わせる名前);
  } else if (名前が存在) {
    名前を削除;
    OK();
  } else {/* 名前が存在しない */
    ERROR();
  }
}
```

図 3 サーバ側の処理．LINK() などの表記は，その返却値を括弧内の値とともにクライアントに送信するという処理を示す
Fig. 3 Server-side procedure. Expression 'LINK()' means that server returns LINK messege with its arguments.

ここまでに述べた単純な実装法の欠点は，ビューパスの下位に存在する名前空間のみに存在する資源の場合，その上位にある空間への探索がオーバヘッドとなり，効率的な探索を行うことができないという点である．また，同様の理由で，多数のリモートリンクをたどってたどり着くことができる名前の場合，その前の名前サーバへの探索がオーバヘッドとなる．このため，多重名前空間の実現においては，キャッシングなどを用いて効率的に探索を行うことが重要な要件となる．

3.2 アクティブネットワーク技術を用いた実現方式の検討

提案方式は，3.1 節で述べた実現方式に対して，通信経路上の機器を用いて名前探索のキャッシュ機構を導入したものである．

広域ネットワーク上に散在するサーバとクライアントの間の通信においては，直接的に通信ができる場合は稀で，一般にルータなどを介してパケットが受け渡されることで通信が行われる．普通はルータにおいて

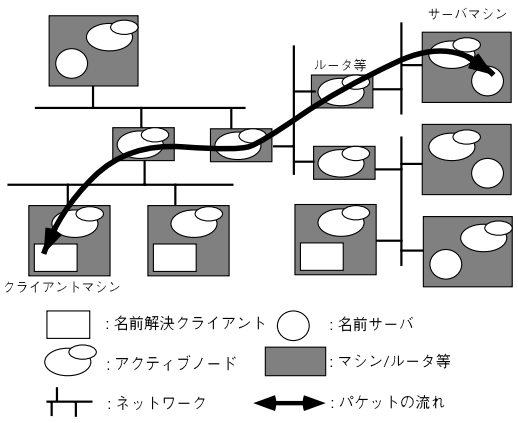

図 4 提案方式の構成
Fig. 4 Configulation of proposed method.

は，ルータ自身に送られてきたパケットを次に受け渡すべきルータに対して送信する処理のみが行われる．しかし，これらの機器に何らかの処理を行わせることで，以下に述べるような利点が生まれる．

第 1 に，これらの処理は端点のサーバ・クライアントに透明にすることが可能であり，従来から存在する通信プロトコルを経路上の機器に改良を加えることで容易に拡張することができる．第 2 に，経路上の機器で処理を完了することができた場合，それ以上先にパケットが伝播することがなくなるため，パケット数を減少させ，反応時間を短縮させることが可能となる．

提案方式の構成を**図 4** に示す．クライアントが送信した探索要求のパケットは，まずそのクライアントと同じマシンに存在する**アクティブノード**（active node）に送信される．アクティブノードとは，各マシンに 1 つずつ存在する仮想的な通信デバイスであり，通過するパケットに対して以下の処理を行う．アクティブノードは，必要ならばそのパケットの内部を解析したのち，目的のサイトへの経路を決定し，次のアクティブノードへ送信する．目的のマシンに到達すると，そのマシンのアクティブノードはマシン内のローカルな通信でサーバにパケットを配送する．サーバからクライアントへパケットが送られる際も同じ手順で送信される．これらの配送処理は，サーバとクライアントには透明なものとなっており，名前空間探索のキャッシュの処理は，このアクティブノードにおいて行われる．

アクティブノードにおける単純なキャッシュ構成方式は以下のようになる．クライアントとサーバの間で交換されるパケットがアクティブノードを通過する際にアクティブノードはその内容を解析し，サーバからの返答パケットであればキャッシュに情報を追加する

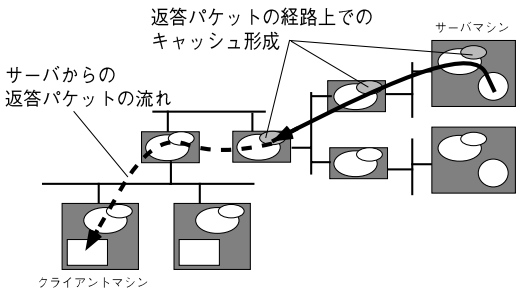

図 5 キャッシュの形成
Fig. 5 Constraction of cache entry.

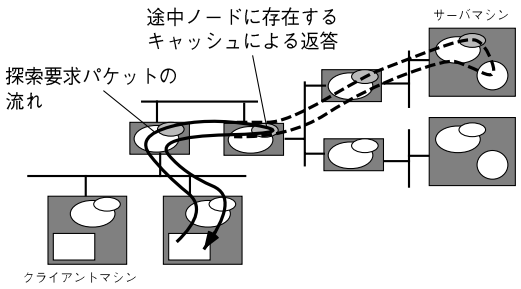

図 6 キャッシュの利用
Fig. 6 Using cache entry.

ことでキャッシュを形成する（**図 5**）．たとえば，メッセージ LOOKUP の返答値 OK を含んだ返答パケットから「ある名前がある名前空間に存在する」という情報を得てキャッシュに記録する．そして，クライアントからの探索要求パケットに対し，各アクティブノードは自分が持つキャッシュと探索要求を照合する．もし適切な返答が可能な場合は，そのアクティブノードがあたかもサーバであるような振舞いをしてクライアントに返答する（**図 6**）．このとき，探索要求パケットはサーバに転送されないため，パケットのトラフィックの減少と，クライアントの要求に対する反応時間の短縮が可能である．

しかし，この単純なキャッシュ構成方式では，クライアントとサーバの間にあるすべてのアクティブノードがキャッシュを保持することになる．経路上のすべてのアクティブノードがキャッシングを行うと，名前空間が更新された際にそのすべてのアクティブノードに対してキャッシュの無効化を行わなければならず，その際に交換されるパケットの数や，キャッシュの存在位置を記憶するためにかかるコストが増大する．これを解消するために，名前の更新頻度をパラメータとして，キャッシュするアクティブノードを制限する機能を導入する．これにより，更新頻度が高い名前に関してはキャッシュするアクティブノードを最小限にし名前の更新コストを下げ，更新頻度が低い名前に関しては最大限にキャッシュし，レスポンスの向上を図ることが可能となる．

提案方式では，サーバおよびクライアントには透明にキャッシュ機構を実現することを目指している．そのため，サーバおよびクライアントから更新頻度の情報を得ることは，透明性を損なうことになるので採用することはできない．提案方式では，情報をキャッシュに記録するノードを，キャッシュによって返答をした次のノードに限る．この方式により，情報の伝播を最低限に抑えることができ，かつアクティブノードでキャッシュがヒットするごとに隣のノードに伝播してゆくようにできる．すなわち，更新の頻度が高く，すぐに無効化される名前の伝播は最小限になり，無効化されない名前は次第に隣のノードに伝播してゆくためレスポンスを向上させることが可能となる．このキャッシュの伝播を我々はインクリメンタルなキャッシュの伝播と呼ぶ．

3.3 インクリメンタルなキャッシュ構成方式

以下では，このインクリメンタルなキャッシュ伝播を実現するキャッシュ構成方式について述べる．

キャッシュの内容

アクティブノード内のキャッシュは，以下の組をキーとして検索する．

- 名前空間名
- 論理資源名

そして各キャッシュは，以下のデータを持つ．

- キャッシュの伝播先
- 物理資源名（論理資源名が存在する場合）
- リンク先のサーバ名（論理資源名がリモートリンクの場合）
- 存在しないという情報自体（論理資源名が存在しない場合）

名前が存在しないという情報をキャッシュする利点は，「ある論理資源名が存在しない」という情報を使うことで，無駄な名前空間の探索をスキップさせることが可能となる点である．

キャッシュの形成と利用

キャッシュがクライアントの要求に対してインクリメンタルにキャッシュが伝播してゆくために，アクティブノードが無制限にキャッシュ処理を行わないよう以下の処理を行う．

サーバから直接返答パケットを受け取ったアクティブノードは，その内容を解析しキャッシュに情報を加える．ここで，この情報が relay 先のアクティブノードより先でキャッシングされることを抑止するため，パ

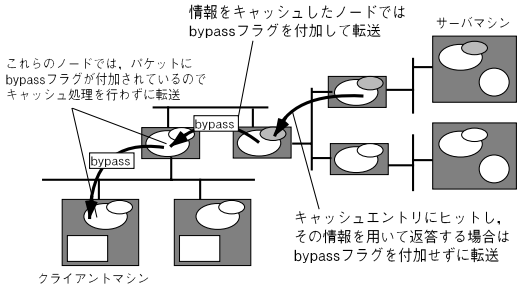

図 7 Bypass フラグの付加によるキャッシュ伝播の制限
Fig. 7 Restricted cache propagation by attaching bypass flag.

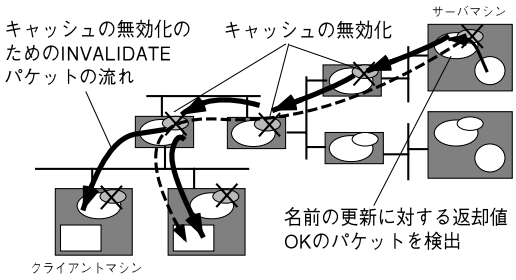

図 8 キャッシュの無効化
Fig. 8 Invalidation of cache entry.

ケットに「bypass フラグ」を付加する．bypass フラグが付加されたパケットを受け取ったアクティブノードは，そのパケットに対するキャッシュの処理を行わず，単に次のアクティブノードに relay する処理のみを行う．この処理により，サーバ・クライアント間の通信経路上のアクティブノードのうち，最初は最もサーバに近いアクティブノードにのみキャッシュが存在することになる．

クライアントからの探索要求がアクティブノードに到着した際，その要求がキャッシュを用いて解決できる場合，アクティブノードは返答パケットに bypass フラグを付加せずに返答する．そして，relay 先のアクティブノードで情報がキャッシュされたことを記憶するため，各キャッシュエントリに relay 先のアクティブノードを記録する．bypass フラグが付加されていない返答パケットを受け取ったアクティブノードはキャッシュの処理を行い，bypass フラグを付加して次のアクティブノードに relay する．これにより，キャッシュが 1 アクティブノード分伝播したことになる．

図 7 に，bypass フラグの付加によってキャッシュの伝播が制限される例を示す．

キャッシュの無効化

サーバに名前が登録されスキップしていた名前空間に名前が出現した場合などは，名前解決の整合性を保つためにこのキャッシュを無効化する必要がある．アクティブノードでは，メッセージ ADDNAME，ADDLINK，DELNAME に対するサーバからの返答値 OK を監視することで名前空間の更新を検出する．

名前空間の更新を検出したアクティブノードは，更新された情報をキャッシュしているアクティブノードに，該当するキャッシュの無効化を指示するメッセージ INVALIDATE を送信する．このとき，送信すべきアクティブノードを決定するために，前述したキャッシュに記録したキャッシュの伝播先を使用する．キャッシュの伝播先は，パケットを relay した先のアクティブノードのみ記録すれば十分であるため，この情報の保持によってキャッシュのデータ量が爆発的に増えるということはない．

たとえば，メッセージ ADDNAME によって名前が追加された場合，アクティブノードはその返却値 OK を検知し，クライアントへの返答パケットのほかに，キャッシュエントリに記録された情報の伝播先に対して，無効化すべき名前空間名と論理資源名の組を含んだキャッシュの無効化メッセージ INVALIDATE を送信する（**図 8**）．無効化メッセージ INVALIDATE を受け取ったアクティブノードは，該当するキャッシュを無効化するとともに，そのノードからキャッシュが伝播した先に同様に無効化メッセージ INVALIDATE を送信する．このようにアクティブノード間で無効化メッセージが伝播することでキャッシュの無効化が実現される．

ここまでに述べたアクティブノード内でのキャッシュ処理のアルゴリズムをまとめたものを**図 9** に示す．

キャッシュのオーバフロー

アクティブノード内のキャッシュが一定量を超えオーバフローを起こすなどの理由によって，キャッシュエントリの 1 つを消去する必要が生じる場合がある．キャッシュエントリ内には，キャッシュの無効化パケットを伝播させるために必要な情報が含まれているため，単純にエントリを消去するだけでは不十分である．そこで，消去すべきエントリが無効化されたものとして，上で述べたキャッシュの無効化の伝播処理を行った後，消去すべきエントリを消去する．

4. 実装および実験

提案方式に基づいた多重名前空間の実装を**表 2** に示す環境で行った．

4.1 パケット数に関する実験

本論文では，提案方式の有効性を検証するために，

```
if (bypass フラグが付加されている) {
  次のアクティブノードに relay;
} else {
  switch (パケットの種類) {
  case LOOKUP:
    キャッシュを検索
    if (マッチするキャッシュエントリが存在) {
      返答パケットの作成;
      クライアントに返答;
      送り先のアクティブノードを
        キャッシュエントリに記録;
    } else {
      次のアクティブノードに relay;
    }
    break;
  case ADDNAME:
  case ADDLINK:
  case DELNAME:
    次のアクティブノードに relay;
    break;
  case 返答パケット:
    キャッシュを検索
    if (マッチするキャッシュエントリが存在)
      if (受信したデータと矛盾) {
        キャッシュの伝播先に
         INVALIDATE パケットを送信;
        キャッシュの削除;
       }
    } else {
      受信した情報をキャッシュに記録;
    }
    Bypass フラグをパケットに付加;
    次のアクティブノードに relay;
    break;
  case INVALIDATE:
    if (マッチするキャッシュエントリが存在)
      キャッシュの伝播先に
        INVALIDATE パケットを送信;
      キャッシュの削除;
    break;

  }
}
```

図 9 インクリメンタルなキャッシュ構成法におけるアクティブノードでの処理
Fig. 9 Operation of an active node in the incremental caching scheme.

表 2 実装環境
Table 2 Implementation platform.

マシン	M1: Sun Ultra-30 M2–M5, M1', M2': Sun Ultra-60
OS	Solaris 2.6
実装言語	M1–M5: Java JDK 1.1.7 M1', M2': Java JDK 1.1.3
プロトコル	UDP/IP

アクティブノードにおけるキャッシングを行う場合と行わない場合に対して以下の値を調べた．

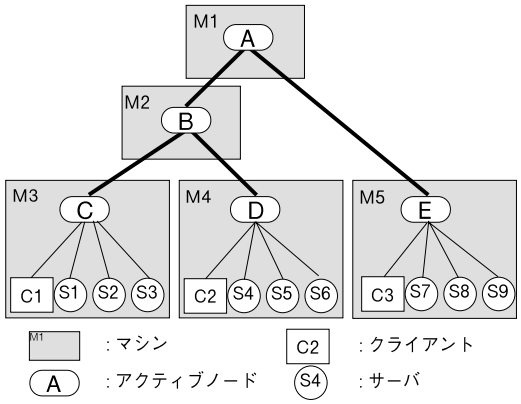

図 10 実験環境（サーバなどの配置）
Fig. 10 Experiment environment (deployment of server and client).

- アクティブノードを通過するパケットの数
- 名前解決サーバに到着するパケットの数
- 1 つの問合せが完了するまでのパケットの hop 数

第 1，第 2 の値によって，問合せ時にサーバやアクティブノードにかかる負荷がどのように減少したかが検証できる．また，第 3 の値によって，名前解決にかかる時間を短縮できるかが分かる．

実験環境を**図 10** に示す．3 つのマシンにそれぞれ 1 つのクライアントと 3 つのサーバを置き，それらをアクティブノードを介して接続している．実際にはこれらのマシン群は同一ネットワークに接続しているため，直接的な通信を行うことが可能であるが，今回の実験では，アクティブノードが行うルーティングを図 10 のネットワークに制限し，仮想的なネットワークを構築した．

これらのサーバ上に，X，Y，Z の 3 つの名前空間を作成した．それぞれの名前空間には**図 11** に示す名前が登録されるようになっている．単一名前空間内の単純な探索方式では，つねに名前空間のルートから探索が開始される．そのため，名前空間のルートに近いサーバほど問合せが集中する．今回の実験では負荷をサーバマシンに均等に分散させるため，名前空間とマシンの割当てを**表 3** のように割り当てている．名前空間 X の割当て法を説明すると，X のルートサーバを S1 に，X 内の名前のうち/a をプレフィックスとして持つものは S4，/a/a をプレフィックスに持つものは S7 に配置している．

実験では以下の処理を各クライアントに行わせて，上記の値を計測した．

- 名前空間に 40 個の名前をクライアント 1 が登録．
- 各クライアントが名前探索・登録・削除の要求を

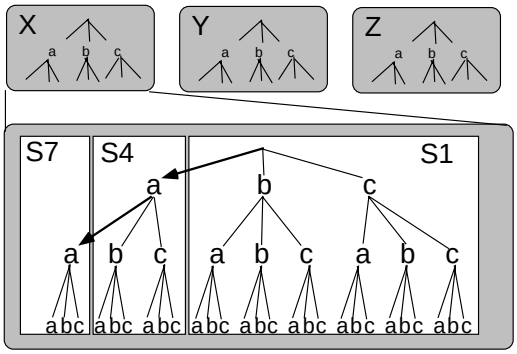

図 11 実験環境（名前空間の構成）
Fig. 11 Experiment environment (configuration of multiple name space).

表 3 名前空間へのサーバの割当て
Table 3 Name space assignment for server.

	/	/a	/a/a
X	S1	S4	S7
Y	S5	S8	S2
Z	S9	S3	S6

1000 回発行.

このとき，登録する名前は/a/a/a，/a/a/b，...，/c/c/c の中からランダムに抽出したものを用いている．また登録する名前空間も X，Y，Z の中からランダムに選んだ．名前と名前空間名の対は $3\times3\times3\times3=81$ 通りであるため，今回の実験ではすべての名前のうち約半数の名前が存在する空間ということになる．

各クライアントが発行する要求は，実験 1 では名前探索・登録・削除の割合がそれぞれ 90%，5%，5%，実験 2 では 20%，40%，40%ずつとした．ここで探索などを行う名前は，上で用いたものと同様に/a/a/a, /a/a/b, ...，/c/c/c の中からランダムに抽出したものを用いている．またその際のビューパスは，$\{X\}$，$\{Y\}$，$\{Z\}$，$\{X\to Y\}$，$\{X\to Z\}$，$\{X\to Y\to Z\}$，$\{X\to Z\to Y\}$ ...などすべての組合せから要求ごとにランダムに選んだ．

4.2 実験結果

図 12 は，今回の実験で各アクティブノード /サーバを通過したパケット数を示している．ここに示されるとおり，キャッシュなしの場合に比べ，キャッシュありの場合のパケット数が送信と受信のいずれにおいても大きく減少していることが分かる．この結果から，経路上でキャッシュ処理を行うことで，すべてのアクティブノードにおいてパケット数が減少していることが示される．また，サーバに届くパケットの数も大きく減っており，サーバの負荷を軽減する観点からも，提案方式の有効性が示された．また，実験 1 と比べて

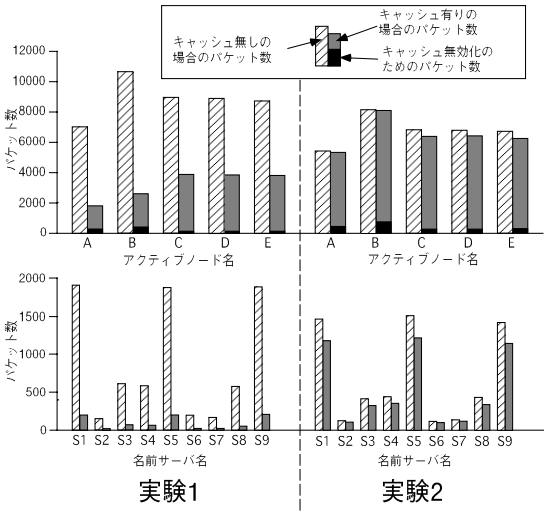

図 12 実験結果：通過パケット数
Fig. 12 Result: numbers of packets passed.

表 4 実験結果：平均 hop 数
Table 4 Result: average hop count.

	実験 1		実験 2	
	キャッシュ		キャッシュ	
	有	無	有	無
C1	2.79	7.34	6.60	7.38
C2	2.86	7.33	6.54	7.30
C3	3.12	8.01	7.34	8.08

名前の更新頻度が名前の探索頻度に比べて多く，キャッシュの有効性が低いと思われる実験 2 の場合であっても，キャッシュなしの場合より良好な結果が得られていることが分かる．これは，キャッシュの伝播を最小限に抑え，キャッシュの無効化にかかるコストを抑える処理を行ったことによるものであると考えられる．

表 4 は，各クライアントが問合せを行った際に，返答パケットが返ってくるまで平均何 hop かかったかを示したものである．Hop 数は，パケットが何回転送をされたかを示す値である．たとえば，図 10 においてクライアントの要求とそれに対する返答が C1 → C → C1 という経路で戻ってきた場合は 2 hop と数えることとする．同様に C1 → C → B → D → S4 → D → B → C → C1 のときは 8 hop となる．

この結果から，キャッシュ処理により hop 数が減っており，クライアントの要求に対してより速やかに返答できるようになったのが分かる．これによって，名前解決機構の平均的レスポンスを向上させることができると考えられる．

4.3 応答時間の比較

本節では，前節で行ったものと同じ実験を広域ネッ

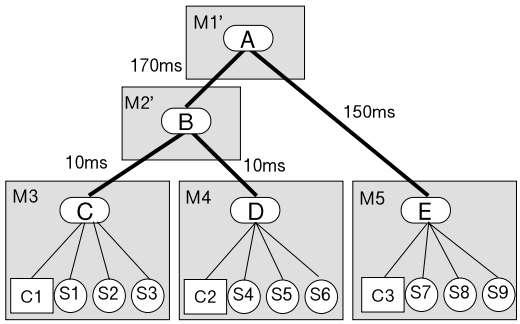

図 13 広域ネットワークでの実験環境．ノード間に付記してある数字は，各ノード間のパケットの遅延時間
Fig. 13 Experiment environment in wide-area network. Numbers between nodes represents packet delay time.

表 5 実験結果：ローカルエリアネットワークでの平均反応時間（単位：ms）
Table 5 Result: average response time in local-area network (unit: ms).

	実験 1		実験 2	
	キャッシュ		キャッシュ	
	有	無	有	無
C1	251.7	500.7	358.4	386.7
C2	227.2	501.4	350.4	385.2
C3	231.8	497.9	365.6	390.8

表 6 実験結果：広域ネットワークでの平均反応時間（単位：ms）
Table 6 Result: average response time in wide-area network (unit: ms).

	実験 1		実験 2	
	キャッシュ		キャッシュ	
	有	無	有	無
C1	326.5	817.4	598.6	627.2
C2	340.1	838.1	598.6	613.9
C3	517.0	1357.1	1015.9	1055.1

トワークを用いて行い，ローカルエリアネットワーク，広域ネットワーク双方の場合での平均応答時間の比較を行った．ここで応答時間とは，クライアントの要求が発行されてから完了するまでの時間とする．

広域ネットワークを用いた実験では，前節での実験環境において，マシン M1' を University of Uppsala（スウェーデン国），M2' を通商産業省工業技術院電子技術総合研究所，残りのマシンを筑波大学におき，パケットの遅延を大きくした環境で実験を行った（**図 13**）．

結果を**表 5**，**表 6** に示す．レスポンスタイムには，パケットのロストが発生したときの待ち時間を除外しているため，この値はパケットが正しく送受信された場合のものとなっている．更新率を低く設定した実験 1 の場合，パケットの遅延時間が大きい場合はキャッシュの効果が大きく現れることが分かる．これは，途中のアクティブノードがキャッシュを用いて返答することで，そのノードより先のネットワークの遅延時間が削減できたことによる．また，更新率を高く設定した実験 2 の場合であっても，キャッシュの無効化のコストが抑えられているため，性能の向上が見られた．

前述したように，今回の実験ではパケットのロストにおける再送のコストを含めていない．現実の広域ネットワーク上の通信ではパケットのロストが発生するため，提案方式におけるパケット数の減少は，パケットのロストにによるパケット再送などのコストを最小限にできる．このため，キャッシュの有無で実際の平均反応時間はさらに大幅な差が現れることとなる．

5. 関 連 研 究

単一な名前空間をユーザがカスタマイズ可能とするものとして，種々の手法が提案されている．Prospero [7] は，他の資源へのリンクを，フィルタや探索プログラムとして定義することで，ユーザ独自の名前空間を作成することを可能としている．Active Names [9] では，複数の名前解決ルーチンを組合せ可能とし，ユーザの環境や要求に合わせてカスタマイズすることを可能としている．これらの研究と我々の提案する多重名前空間との相違点は，名前空間を重ね合わせるという非常に簡単な操作でユーザが名前空間のカスタマイズを行えるという点である．

また，単一名前空間を重ね合わせることで 1 つの名前空間を作るというものでは，Translucent file service [2] や 3-D file system [4] がある．これらはともに Unix ファイルシステムの名前空間を拡張するものであり，ディレクトリ単位で重ね合わせ方を指定可能としている．これは，非常に細かな指定が行える反面，管理が煩雑となる欠点がある．我々の提案方式では，重ね合わせを名前空間単位に限定することで名前空間の管理を単純化しつつ，資源名の指定のたびに名前空間の重ね合わせを変更することを可能とすることで使用時の柔軟性を達成している．

現在，世界で活発に行われているアクティブネットワーク研究は，2 つに大別することが可能である．まず，第 1 にパケットにプログラムを組み込み，それがアクティブノードを通過する際に実行されるというものである．もう一方は，アクティブノード上のパケット処理プログラムを動的に配置可能とし，ルータなどの機能を拡張するものである．

前者の研究としては，PLAN [3] や ANTS [10]，後者の研究としては NetScript [11] を代表例としてあげる

ことができる．これら2つの研究は，セキュリティや実行効率の観点，またその上で利用するアプリケーションによって，どちらのアプローチが適しているかが異なってくる．前者の研究においては，パケットごとにルータの振舞いを変化させることが可能であるため，非常に柔軟性が高い反面，実行時のコスト削減やセキュリティに対する防御を厳密に行う必要がある．同様に後者では柔軟性という点では劣っているが，アクティブノード内である程度固定されたプログラムが動くため，最適化やセキュリティに対しては有利である．

我々の提案している名前解決方式は，名前解決ごとにプログラムを入れ替えるような柔軟性は必要としていない．我々の今回の実装では，あらかじめパケット処理のプログラムを各ノードに用意していたが，さらなる柔軟性のためにプログラムを事前または実行直前に動的に途中経路上のアクティブノードに配置することも考えられる．そのような場合は後者のようなアクティブネットワークの実行時システム上に提案方式の名前解決機構を実装することになる．

パケットの途中経路上でのキャッシングを行うソフトウェアとしては，UnxSoft 社の Clarity Web Cache[8] がある．このソフトウェアは，ルータ上で透過的に Web ページのキャッシングを行うものである．我々の提案方式では，単に途中経路で透過的にキャッシングするだけでなく，各ノードが名前空間の更新情報を交換することでキャッシュの一貫性を保持している点が異なる．

6. おわりに

単一な名前空間を，一貫した方法でユーザがカスタマイズすることを可能とする多重名前空間を提案した．また，その実現に際し，アクティブネットワーク技術を応用に通信経路上でキャッシュを行うことで名前解決が効率化できることを示した．

今後の課題には以下のものが考えられる．第1に，アクティブノードで行うキャッシュ処理の高度化によるさらなる性能の向上とノードの故障への対処がある．第2に，UDP/IP レベルのパケットを直接 snoop するような低レベルな実装を行うことがある．現在，提案方式の実装には Java を用いているが，トラフィックが非常に大きくなると考えられる基幹のルータなどに実装する場合は，低レベルな部分で実装を行うことで最大限の高速化を行い，パケットのスループットを向上させることが必須といえる．第3に，提案方式のキャッシュ手法の既存プロトコル（Domain Name System[5),6)] など）への適用があげられる．DNS は，ローカルのサーバで解決できない問合せをすぐにルートサーバへ転送する（もしくは，forwarder として上位のサーバを静的に設定しておく必要がある）ため，ルートサーバへの負荷は膨大なものとなりがちである．これらをパケットの送信経路上で解決できれば，サーバの負荷やネットワーク全体の負荷を低減できると考えられる．

謝辞 広域ネットワーク環境での実験環境構築に協力していただいた通商産業省工業技術院電子技術総合研究所の一杉裕志氏，ならびに University of Uppsala の Christian F. Tschudin 氏に深く感謝の意を表する．

参考文献

1) Berners-Lee, T., Fielding, R. and Masinter, L.: Uniform Resource Identifiers (URI): Generic Syntax, Request for Comments: 2396 (1998).
2) Hendricks, D.: A Filesystem For Software Development, *USENIX Summer Conferance*, Anaheim, California, pp.333–340 (1990).
3) Hicks, M., Kakkar, P., Moore, J.T., Gunter, C.A. and Nettles, S.: PLAN: A Packet Language for Active Networks, *Proc. 3rd ACM SIGPLAN International Conference on Functional Programming Languages*, pp.86–93, ACM (1998).
4) Korn, D.G. and Krell, E.: The 3-D File System, *USENIX Summer Conference*, pp.147–156 (1989).
5) Mockapetris, P.: Domain Names – Concepts and Facilities, Request for Comments 1034 (1987).
6) Mockapetris, P.: Domain Names – Implementations and Specification, Request for Comments: 1035 (1987).
7) Neuman, B.C.: Prospero: A Tool for Organizing Internet Resources, *Electronic Networking: Research, Applications and Policy*, Vol.2, No.1, pp.30–37 (1992).
8) UnxSoft Ltd.: The Clarity Web Cache: A Transparent Caching Proxy Server. http://www.unxsoft.com/clarity.html.
9) Vahdat, A., Dahlin, M., Anderson, T. and Aggarwal, A.: Active Names: Flexible Location and Transport of Wide-Area Resources, *Proc. USENIX Symposium on Internet Technologies and Systems* (*USITS*), USENIX (1999).
10) Wetherall, D.J., Guttag, J.V. and Tennenhouse, D.L.: ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols, *IEEE OPENARCH '98*, San Francisco, CA (1998).
11) Yemini, Y. and da Silva, S.: Towards Pro-

grammable Networks, *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, L'Aquila, Italy (1996).

(平成 11 年 12 月 15 日受付)
(平成 12 年 4 月 6 日採録)

東村　邦彦

1973 年生．1995 年筑波大学第三学群情報学類卒業．1997 年同大学大学院博士課程工学研究科前期課程修了．現在同大学院博士課程在学中．オペレーティングシステム，プログラミング言語，モーバイルエージェントシステムに興味を持つ．日本ソフトウェア科学会会員．

加藤　和彦（正会員）

1962 年生．1985 年筑波大学第三学群情報学類卒業，1987 年工学修士（筑波大学大学院工学研究科），1992 年博士（理学）（東京大学大学院理学系研究科）．1989 年東京大学理学部情報科学科助手，1993 年筑波大学電子・情報工学系講師，1996 年同助教授，現在に至る．オペレーティングシステム，プログラミング言語システム，データベースシステム，分散システム，モバイルオブジェクト計算に興味を持つ．電子情報通信学会，日本ソフトウェア科学会，ACM，IEEE 各会員．

松原　克弥

1971 年生．1994 年筑波大学第三学群情報学類卒業．1996 年筑波大学大学院修士課程理工学研究科修了．1998 年より筑波大学電子・情報工学系助手，現在に至る．オープンネットワークのための分散システムソフトウェアに興味を持つ．USENIX 会員．