修 士 論 文

分散オブジェクト環境での
サービス管理機構に関する研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

冨田 順

2003年3月

修 士 論 文

分散オブジェクト環境での
サービス管理機構に関する研究

指導教官　堀口 進 教授

審査委員主査　堀口 進 教授
審査委員　日比野 靖 教授
審査委員　Shen Hong 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

110088 冨田 順

提出年月: 2003 年 2 月

目 次

第 1 章　はじめに **1**
1.1 研究の目的と背景 1
1.2 本論文の構成 2

第 2 章　大規模分散オブジェクトシステム **3**
2.1 はじめに 3
2.2 オブジェクト指向プログラミング 3
2.3 分散オブジェクト 5
2.4 代表的な分散オブジェクトミドルウエア 6
2.4.1 CORBA 7
2.4.2 HORB 7
2.5 分散オブジェクトにおける代表的なサービスとその役割 7
2.5.1 ネーミングサービス 8
2.5.2 イベントサービス 9
2.5.3 トランザクションサービス 11
2.6 要求される主な要件 12
2.6.1 負荷分散 12
2.6.2 サービスの冗長化 12
2.7 諸用件の実装に関する先行事例と問題点 12
2.8 まとめ 13

第 3 章　階層構造を導入したネーミングサービス (HCN) **14**
3.1 はじめに 14
3.2 HCN の概要 14
3.3 HCN の構成と手法 15
3.3.1 ネーミングサービスへの階層構造の導入 16
3.3.2 オブジェクト登録の手法 18
3.3.3 オブジェクト参照の手法 19
3.4 本システムが対象とする利用環境 20
3.5 まとめ 20

第 4 章　HCN を用いた大規模分散オブジェクトシステムの評価　21
4.1　はじめに 21
4.2　シミュレーション環境 21
4.2.1　クライアントの動作 21
4.2.2　サービスの動作 22
4.3　システム評価 24
4.3.1　クライアント数 256 個の場合 28
4.3.2　クライアント数 1024 個の場合 33
4.3.3　クライアント数 4096 個の場合 38
4.4　結果 43
4.5　まとめ 45

第 5 章　結論　46
5.1　まとめ 46
5.2　今後の予定 46

図 目 次

2.1 分散オブジェクトミドルウエアのアーキテクチャ 6
2.2 ネーミングサービスの動作 8
2.3 push 型イベントサービスの動作 10
2.4 pull 型イベントサービスの動作 10
2.5 トランザクションサービスの動作 11

3.1 階層構造の導入 16
3.2 階層情報の初期化 17
3.3 HCN でのオブジェクトの登録手法 18
3.4 HCN でのオブジェクトの参照手法 19

4.1 シミュレーション環境 22
4.2 クライアントの状態遷移図 23
4.3 サービスの状態遷移図 23
4.4 ネーミングサーバが 1 個の場合 (1 階層 HCN) 25
4.5 2 階層 HCN の場合 25
4.6 3 階層 HCN の場合 26
4.7 4 階層 HCN の場合 26
4.8 5 階層 HCN の場合 27
4.9 ネーミングサーバが 1 個の場合の平均リクエスト処理時間 (1 階層 HCN) . . 29
4.10 2 階層 HCN での各クライアントの平均リクエスト処理時間 29
4.11 3 階層 HCN での各クライアントの平均リクエスト処理時間 30
4.12 4 階層 HCN での各クライアントの平均リクエスト処理時間 30
4.13 5 階層 HCN での各クライアントの平均リクエスト処理時間 31
4.14 各階層における平均リクエスト処理時間 32
4.15 各階層における総リクエスト処理数 32
4.16 ネーミングサーバが 1 個の場合の平均リクエスト処理時間 (1 階層 HCN) . . 34
4.17 2 階層 HCN の場合での各クライアントの平均リクエスト処理時間 34
4.18 3 階層 HCN の場合での各クライアントの平均リクエスト処理時間 35
4.19 4 階層 HCN の場合での各クライアントの平均リクエスト処理時間 35
4.20 5 階層 HCN の場合での各クライアントの平均リクエスト処理時間 36
4.21 各階層における平均リクエスト処理時間 37

4.22 各階層における総平均リクエスト処理時間 37
4.23 ネーミングサーバが 1 個の場合の平均リクエスト処理時間 (1 階層 HCN) . . 39
4.24 2 階層 HCN の場合での各クライアントの平均リクエスト処理時間 39
4.25 3 階層 HCN の場合での各クライアントの平均リクエスト処理時間 40
4.26 4 階層 HCN の場合での各クライアントの平均リクエスト処理時間 40
4.27 5 階層 HCN の場合での各クライアントの平均リクエスト処理時間 41
4.28 各階層における全クライアントの平均リクエスト処理時間 42
4.29 各階層における全クライアントの平均リクエスト処理数 42
4.30 クライアント数に対する平均リクエスト処理時間 44
4.31 クライアント数に対する総リクエスト処理数 44

第1章　はじめに

1.1　研究の目的と背景

コンピュータネットワークの普及は、人間対人間のコミュニケーションに新しい手段を提供したが、コンピュータプログラムの世界においてもまた、ネットワークを用いたコミュニケーションを用いることで、新たな利用方法、利用場面が開拓されていった。人間対人間の世界においては、Web やメールなどの情報通信手段が挙げられる。これらの普及によって空間的・時間的制限を乗り越えて人々がコミュニケーションをとることが可能になった。コンピュータにおいて、プログラムの通信がネットワーク経由で行われるということは、プロセッサやディスクストレージなどの計算機資源が手元に無くても、ネットワークで接続されたマシンがそれらを提供してくれるのならば、ネットワーク経由でそれらが利用できるようになるということである。

このコンピュータネットワークのモデルに、近年注目されているオブジェクト指向プログラミングに適用して誕生したのが分散オブジェクトである。オブジェクト指向プログラミングは、クラスとインスタンスなどに代表される様々な新しい概念を導入したプログラミング手法である。分散オブジェクトでは、オブジェクトがプログラムの構成単位となって、ネットワーク上のマシンに分散して配置され、オブジェクト同士がネットワークを介してメッセージを通信することによって処理を行う。この分散オブジェクト環境を用いて、分散処理環境 [1] や企業における Web システムなど [7, p.22] が構築されている。大規模な分散システムを構築する際には、さまざまな案件に共通して発生する要求がある。これらに応えるために、分散オブジェクトフレームワークにおいて様々なサービス [4] と呼ばれる機能が提供されている。この中でも比較的重要度の高いものがネーミングサービスである。ネーミングサービスは分散オブジェクト中のオブジェクトリポジトリとして機能する。ネーミングサービスにオブジェクトとその名前を登録しておくことで、分散環境中のクライアントがネーミングサービス中に登録されているオブジェクトを利用することができるようになる。ネーミングサービスにオブジェクトリファレンスを登録しておけば、サービスのある場所をクライアント側で把握しなくても、ネーミングサービスにさえ接続できれば、サービスを利用することが可能になる。これ以外にも、作業中のスナップショットを保存したり、頻繁にバージョンアップが行われるオブジェクトを最新の状態に保つなど、様々な用途が考えられる。

しかし、重要度の高いネーミングサービスは、サービスダウンになってしまうと、分散オブジェクト環境全体に影響を及ぼしてしまう。また、クライアントの増加に対しても対

応できるように、負荷分散ができなければならない。このことを実現するために、通常、ネーミングサービスを提供するサーバであるネーミングサーバを複数台用意し、これらを協調して動作させる手法がとられる。この手法はフェデレーションと呼ばれるもので、フェデレーション動作が可能なネーミングサービス [2][3] がいくつか提供されている。

そこで本研究では、ネーミングサービスに階層構造を導入してフェデレーションを実現する手法である HCN(Hierarchcal Controlled Namingservice) を構築する。このため、ネーミングサーバ間へ階層構造を導入する。次にネーミングサーバ間で登録された内容について矛盾が生じないようなオブジェクト登録手法と、参照時の負荷を分散させる参照手法を提案する。

1.2 本論文の構成

本論文の構成は以下の通りである。第 2 章ではオブジェクト指向プログラミングと分散オブジェクトの概要を説明し、分散オブジェクト環境を構築する際に用いられるフレームワークと、分散オブジェクトを用いてシステムを構築する際に用いられるサービスについて紹介する。3 章ではネーミングサービスに階層構造を導入して、フェデレーションを実現するための階層型ネーミングサービス制御法 (HCN) を提案する。4 章では提案した HCN を評価するためにシミュレーション環境を構築し、これを用いて HCN の有効性を検証する。5 章では本論文のまとめと、今後の課題について述べる。

第2章　大規模分散オブジェクトシステム

2.1　はじめに

分散オブジェクト技術は、ネットワークを介して、リモートにあるオブジェクトを、あたかもローカルにあるオブジェクトのように扱うことを可能にする技術である。この技術は金融や物流などのシステムや、分散コンピューティングの基板技術として利用されている。[7, p.22]

分散オブジェクト技術を用いてシステムを構築する際に、オブジェクト間の通信を、個別の案件に対して実装するという事をしなくても、標準的なオブジェクト間通信をフレームワーク化し、実装したORB(Object Request Broker)が提供されている。しかしながら、分散環境を実現し、多数のマシンに同一のオブジェクトを配布したり、共有をしたいというような、要求が発生した場合、なんらかのサービスの力を借りない限り、事実上不可能である。

この要求を満足させるために、ORBはネーミングサービスを提供する。このサービスは、分散オブジェクト環境中でのオブジェクトリポジトリとして機能し、オブジェクト本体や、動作中のオブジェクトに対するプロキシが登録される。これにより、ネットワーク内でオブジェクトを共有し、利用することを実現する。しかし、システムの大規模化により、ネーミングサービスがクリティカルミッションを背負うことが考えられる。ノード同士のネットワーク的な距離が延長されることにより、ネーミングサービスに対して負荷分散の必要が生じてきている。

本研究では、負荷分散と信頼性を備えたネーミングサービスであるHCN(Hierarchical Controlled Namingservice)を提案する。2節では、オブジェクト指向プログラミングについて解説し、3節では、オブジェクトがネットワーク経由でメッセージ通信を行う、分散オブジェクトについて説明し、4節で、分散オブジェクト環境上で用いられる代表的なサービスを紹介する。5節では、分散オブジェクト環境を構築する際に用いられる代表的なサービスについて紹介する。システムが大規模化することにより発生する、分散オブジェクト環境に対しての要求を6節で示し、これらの要求に対する先行事例を7節で紹介する。

2.2　オブジェクト指向プログラミング

オブジェクト指向という概念が発明される以前、プログラムは手続き指向で開発されていた。しかし、手続き指向でのシステム開発は根本的な問題を内包していることから、

現在のシステム開発現場では、ほとんどのプロジェクトがオブジェクト指向言語を用いて開発されている。

オブジェクト指向言語以前手法である手続き指向言語では、プログラムが行うべき処理を「関数」という手続きの集合として記述する手法である。手続き指向言語と、手続きを構造的に積み重ねることで、ひとつの大きなプログラムを構築していく構造化プログラミングを組み合わせて、従来のシステム開発は行われた。この手法は、システムの実現に必要な機能を細分化し、細分化した機能の実装を分割して行う事を可能にした。このことで、役割の分担ができるようになり、作業効率が向上した。しかし、システムが大規模化するにつれ、次のようなデメリットが表面化してきた。

- グローバル変数の弊害

  手続き指向言語では、全ての関数から参照・書き換え可能なグローバル変数を用いてプログラムの制御を行っているものが珍しくない。このことは、一ヶ所でのバグの発生が、システム全体の動作を意図しないものにしてしまうことを意味するため、システム全体に対する潜在的な不安定要素となる。

- 変更余波の把握

  プログラム全体の静的・動的構造を把握するのが難しいため、コードの中の1ヶ所の変更が、システム全体に対してどこまで影響を及ぼすのかを把握することができなくなる。

- 再利用性の欠如

  手続き指向言語のプログラムは、変数に対しての依存が非常に大きい。そのため、過去に作成されている似たような機能を実装したコードをそのまま現在のシステムの中に組み込むことができない。実現したい機能が過去に実装されてあっても、毎回新しく作りなおさなければならないため、非常に再利用性が低い。

そこで、オブジェクト指向言語では、処理を「オブジェクト」という単位で分割させる手法をとっている。オブジェクト指向を導入することによって、以下のような利点がある。

- オブジェクトは、処理の手続きであるメソッドと、オブジェクトの状態を保存するプロパティから成っており、オブジェクトが動作するのに必要な変数を全てオブジェクトの中に持つ。このため、グローバル変数を必要としないでプログラムを構築することができる。

- オブジェクトはメソッドとプロパティから成っており、オブジェクトが動作するオブジェクト内部にあるプロパティは、外部のオブジェクトからは直接アクセスできず、アクセスする際はメソッドを経由しなければならない。インターフェースにアクセスを集中させることにより、オブジェクトの内部変更を、他のオブジェクトが意識しなくてもよくなる。

- オブジェクトは各々独立に変数を持っているため、周辺の環境に対する依存が非常に小さい。このため、再利用性が非常に高い。

システム全体の処理は、オブジェクト間でメッセージを交換し、お互いのメソッドを呼出し合うことで、処理を進行させる。このような原理に基づき、オブジェクトが異なるマシン上で動作し、オブジェクト間通信がネットワークを介して行われるものが分散オブジェクトである。

2.3 分散オブジェクト

分散オブジェクトとは、オブジェクト指向言語で開発されたあるひとつのプログラムを構成するオブジェクトを、多数のマシン上に分散させて動作させる手法である。お互いに異なるマシン上に存在するオブジェクト間の通信はネットワークを介して行われる。ネットワークを介したオブジェクト間通信は、単一のマシンでのオブジェクト間通信とは手続きが異なるために、これらを仮想化するためにORBが提供される。

オブジェクト指向登場以前の手続き指向言語では、RPC(リモートプロシージャコール)と言う形で、リモートにある計算機資源を共有する手段が提供されている。RPCでは、処理の単位は「関数」で表現されている。

それから1970年代にオブジェクト指向プログラミングが登場した。オブジェクト指向では、クラス、インスタンス、メソッド、メッセージなど、それまでの手続き型言語には無かった概念が大量に導入された。このために、RPCをオブジェクト指向のために拡張したのが分散オブジェクトである。

分散オブジェクトを構築するツールであるORBは以下のような透過性をプログラマに提供することが主な役割となる。

- 位置透過性

  リモートに存在するオブジェクトを利用したいときに、ローカルに存在するオブジェクトとは異なる手続きで利用しなければならなのならば、非常に使いにくいものになる。そこで、分散オブジェクトではプロキシモデルを利用して、リモートにあるオブジェクトを操作する場合、ローカルにあるプロキシに対して操作を実行する。このようにすることで、利用者はリモートオブジェクトがネットワーク上のどこに存在しているのかを意識すること無く分散環境を利用することができるようになる。

- 障害透過性

  オブジェクトが分散されている事により、障害が発生しても被害を最小限に抑えることができる。また、障害の発生したサービスに対するバックアップシステムも容易に追加、変更が可能になるため、メンテナンスが容易になると言う利点も生じる。

- 異種透過性

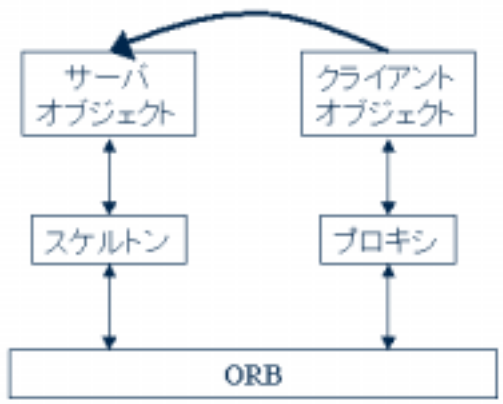

図 2.1: 分散オブジェクトミドルウエアのアーキテクチャ

分散オブジェクト環境が大規模化するにつれて、環境中で稼働するマシンは、様々なプラットホーム、OS、開発言語が入り交じるヘテロな環境になるのは必然のことである。このようなローカルとリモートの間での差異を吸収する機構が要求される。

このような透過性はORB(Object Request Broker) とプロキシ/スケルトンを提供することにより実現される。通常、分散オブジェクトは図 2.1 に示されている通り、オブジェクトレイヤー、プロキシ/スケルトンレイヤー、ORB レイヤーの 3 層に分割される。ORB はネットワークを用いてのオブジェクト間通信を実際に行うのが主な役割だが、そのほかにも、マーシャリング/アンマーシャリングを行うことでマシン間の差異 (プラットホーム、OS、文字コードなど) を吸収することも行う。プロキシとスケルトンは、ORB とオブジェクトの間に入って、オブジェクト同士が行うネットワーク通信をプログラマに意識させない環境を提供する。

2.4 代表的な分散オブジェクトミドルウエア

分散オブジェクト環境を実現する際に、個別のオブジェクトが直接ネットワークソケットを生成し、通信すると言った方法では、汎用性が無く、プログラムの構築も面倒で複雑になってしまう。そこで、分散オブジェクトを用いたプログラムを構築する際のオブジェクト間通信を担当するミドルウエアの仕様や実装が提供されている。以下ではその代表的なものについて紹介する。

2.4.1 CORBA

ORB のひとつで、OMG(Object Management Group) により標準化されている規格として CORBA(Common Object Request Bloker Association) がある**??**。対応している言語は、C、C++、Java など、現在用いられている言語のほとんどに対応していることや、規格が策定された時期が早かったために、現在多くのシステムで稼働している。

あくまで CORBA は規格が標準化されているだけなので、この規格を実装したものが製品やオープンソースで提供されている。

2.4.2 HORB

HORB(Hirano Object Request Broker)**??**は、産総研の平野によって開発され、現在オープンソースで提供されている ORB である。HORB は、PureJava で書かれているため、相互可用性に欠けているように見える。しかし、CORBA が ORB 間で通信する際に用いる IIOP(Internet Inter-ORB Protocol) 経由で通信するエクステンションパッケージが提供されているので、CORBA-HORB 間での通信を実現している。また、Java 自体プラットホームを選ばない言語であるため、非常に高い相互可用性がある。

2.5 分散オブジェクトにおける代表的なサービスとその役割

ローカルに全てのオブジェクトを置いておくのとは異なり、分散オブジェクトでは様々な問題が発生する。ローカルに全てのオブジェクトを置く場合と異なり、分散オブジェクトではオブジェクト間のメッセージ通信にかかる時間が長くなる。リモートオブジェクトが動作しているマシンが、何らかの理由でダウンしているような最悪の場合では、何のレスポンスも返ってこなくなる。また、ネットワーク上のマシンにオブジェクトが散在しているため、各オブジェクトを最新の状態にアップデートさせるためには、オブジェクトを管理する機構が必要になる。このような問題に対応するために、ユーザーが毎回プログラムを書くことは、非常に負担が大きい。そのため、分散オブジェクトミドルウエアでは様々なサービスを提供している。このサービスの中で代表的なものを以下に説明する。

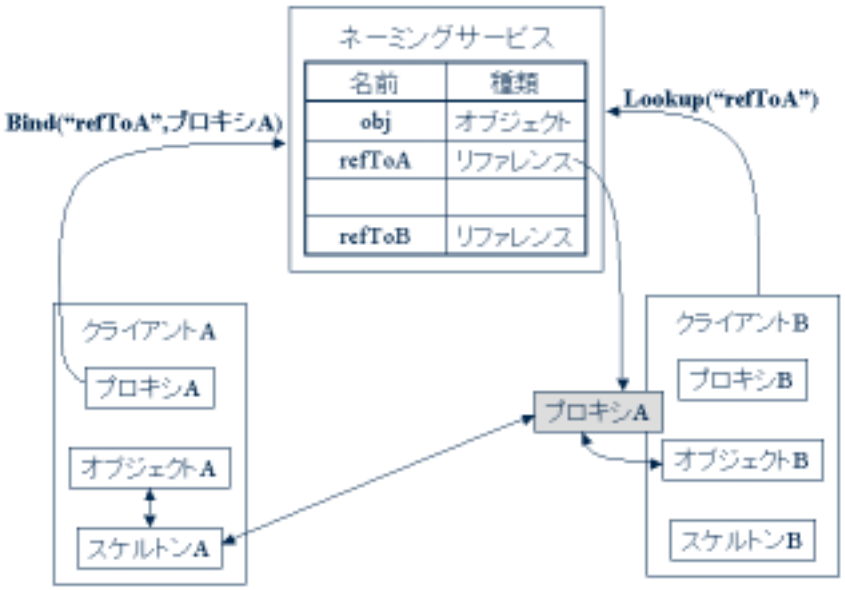

図 2.2: ネーミングサービスの動作

2.5.1 ネーミングサービス

ネーミングサービスは、オブジェクトとその名前を登録することで、名前によるオブジェクトの検索を実現するオブジェクトリポジトリである。ネーミングサービスの動作の様子を、図 2.2 に示す。クライアント A はまず、ネーミングサービスにオブジェクト本体と、オブジェクトを参照するときの名前を送信し、オブジェクトを登録する。こうすることで、ネーミングサービスに接続可能なクライアントの B は、オブジェクトを名前によって検索することが可能になる。このような情報はネーミングサービスの存在により、クライアント間で共有することが可能になる。それと同時に、管理者に対して、分散オブジェクト環境中に存在するオブジェクトを名前で管理する環境を提供する。

ネーミングサービスは、様々なサービスの中でも基本的なサービスであり、クライアントとサーバとの関係に位置透過性をもたらすことから需要も非常に高い。

2.5.2 イベントサービス

一般的なオブジェクト間通信は同期通信でメッセージを送受信する。しかし、同期通信はメッセージの返答を待つ間のプロセスは待ち状態となってしまい、パフォーマンスが上がらなくなってしまう。また、処理に例外が発生した場合や、サービスに障害が発生した場合の障害検知などでのイベントは本質的に非同期メッセージである。そこで、イベントの送信側と受信側はイベントチャネルを介してイベントサービスと接続することで、非同期のイベントの送受信を実現する。さらに、送信側と受信側が直接接続されなくなるために、お互いのポータビリティが向上する。

イベントサービスには 2 種類の動作モデルがる。それは push 型と pull 型である。図 2.3 で示した push 型では、イベントが発火するオブジェクトであるサプライヤオブジェクトが、イベントチャネルのメソッドである push を用いて、発火したイベントを登録すると、イベントを受け取る側であるコンシューマオブジェクトに通知されるモデルである。一方図 2.4 に示してある pull 型では、コンシューマオブジェクトがイベントチャネルを経由して、サプライヤオブジェクトでイベントが発火したかどうかを監視するモデルである。

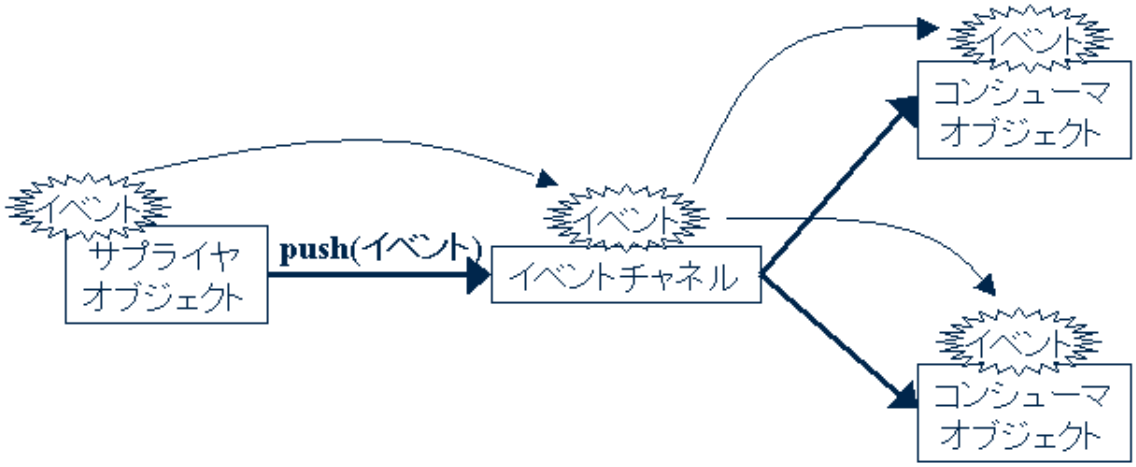

図 2.3: push 型イベントサービスの動作

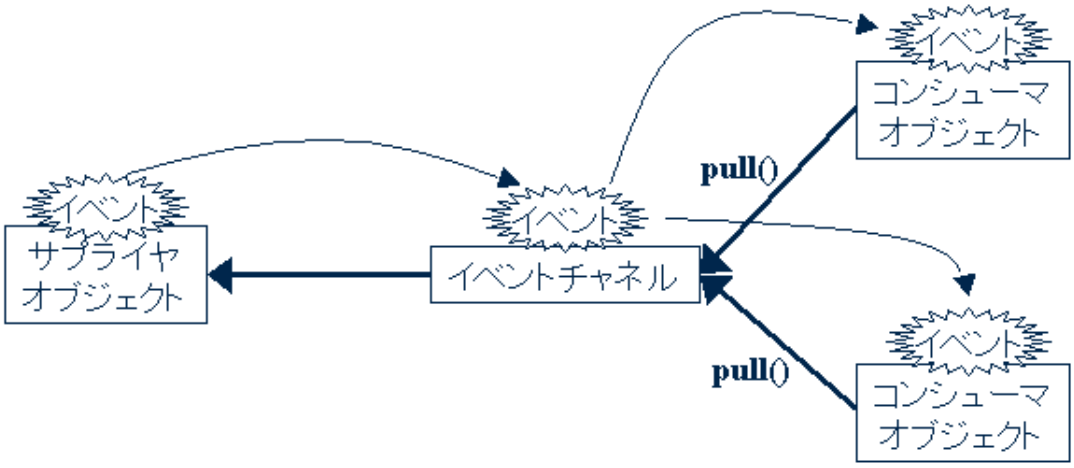

図 2.4: pull 型イベントサービスの動作

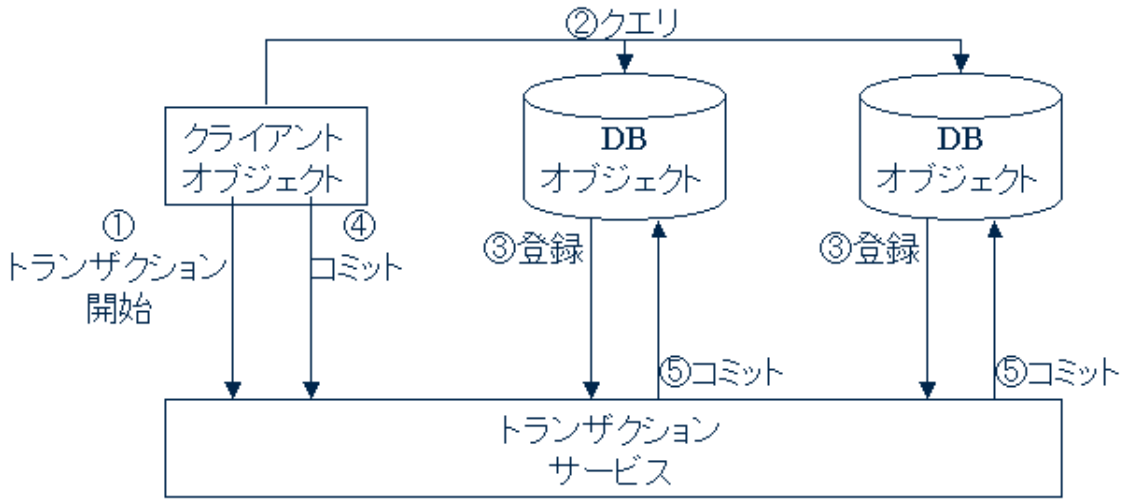

図 2.5: トランザクションサービスの動作

2.5.3 トランザクションサービス

分散オブジェクトを用いたシステムでは、ネットワーク上にある複数のデータベースに対して操作を行うトランザクションが発行される事がある。この場合、各々のクライアントでトランザクションを制御すると、ユーザー間でのデータの整合性が保たれなくなったり、データベースを障害から復旧する際に完全に元にもどらない可能性がある。このようなことを回避するために、クライアントはトランザクションサービスにコミットを代行させる事により、データの整合性を保証することがその役割である。

2.6　要求される主な要件

分散オブジェクトをシステム開発に導入する理由のひとつとして、処理を分散させるということがある。従来からの１台のマシンに集中して処理を行うモデルでは、マシンダウンがサービスダウンに直結してしまうため、メンテナンスが困難になってしまう。そのため分散オブジェクトでは、処理を分散させる事で高いパフォーマンスを実現するだけでなく、同時にリスクを分散させることが求められる。

2.6.1　負荷分散

近年、パソコンの処理速度が向上しており、何百台ものパソコンをネットワークで結合したクラスタマシンが、スーパーコンピュータの処理速度ランキングである TOP500 Supercomputer Organaization [9] にランクされている。このようなクラスタマシンはスーパーコンピュータと比較して、非常に安いコストで高いパフォーマンスを手にすることができる。分散オブジェクトミドルウエアは、オブジェクト間のネットワーク通信を隠蔽するため、クラスタマシンのように1ヶ所にマシンがまとまっていなくても、インターネット上にあるマシンをクラスタリングして動作させることが可能である。

また、負荷分散のもうひとつの利点として、処理をしているマシンがダウンしても、それを代行するマシンが存在することで、システムダウンを回避することが可能なことである。

このように、分散オブジェクトを用いることで、処理速度とシステムの安定性とを両立させることができるようになる。

2.6.2　サービスの冗長化

従来からサービスの信頼性を向上させるために、サービスを提供するサーバを複数用意して、稼働中のマシンに障害が発生した時点でバックアップのサーバが処理を代行するというシステムがよく用いられている。サービスの物理的なリスクとして、停電によるマシンダウンや火事、水害などの天災による不可抗力によってシステムダウンが引き起こされる事がある。これらのリスクを回避するためには、バックアップシステムを物理的に離れた場所に設置する必要がある。このようなシステムを構築する際に、分散オブジェクトミドルウエアはネットワーク通信を隠蔽してくれるため、低コストでサービスの冗長化を実現できる。

2.7　諸用件の実装に関する先行事例と問題点

ネットワークのサブネットを跨って複数のサービスサーバ同士を結合し、ひとつの大きなサービスを提供することをフェデレーションと言っている。フェデレーションは、大規

模な分散オブジェクト環境を構築する際には非常に重要な技術である。多数のサービスを結合する際には、サービス間の一貫性の保証や、効率的な結合手法など、様々な課題が存在する。

現在までにフェデレーションを実現した例として、Belaid 等 [8] によるトレーダーサービスのフェデレーションがある。トレーダーサービスは、オブジェクトにキーワードを付けることで、クライアントから必要なオブジェクトを検索するサービスである。Belaid 等は、CSG(Cooperating Server Graph) からブロードキャストツリーを生成し、これを元に各サーバを巡回する手法を用いている。ネーミングサービスに関して、フランスの IRIT[2] による CORBA 実装”SUMO” において、フェデレーションの手法が提案されている。この手法は、クライアントからのリクエストは、クライアントと同一セグメント内のネーミングサーバにまず入り、一致するものが無ければ他のネーミングサーバにリクエストをブロードキャストする手法である。

分散オブジェクトで用いられるサービスの中で、最も需要が大きいのはネーミングサービスである。そのネーミングサービスで IRIT によるフェデレーションの手法では、基本的なアプローチとしてブロードキャストを用いるので、フェデレーションの規模が大きくなるにつれて、負荷分散の効率が落ちてしまう。このため、フェデレーションの大規模化に対しても負荷分散効率を高く保つことができ、拡張に対しても柔軟な手法の導入が望まれる。

2.8 まとめ

本章では、オブジェクト指向プログラミングと分散オブジェクトの概要と、分散オブジェクト環境中で動作する各サービスについて説明した。分散オブジェクト環境が大規模化した場合に、各サービスに要求される要件を示し、これまでに提供されている先行事例を紹介した。先行事例では解決されていない問題について指摘した。次章では、階層構造を導入したネーミングサービスについて説明する。

第3章　階層構造を導入したネーミングサービス(HCN)

3.1　はじめに

これまでのネーミングサービスのフェデレーション手法は、ブロードキャストを用いて通信を行うために、クライアント数の増加とともに、ネットワーク全体に影響を及ぼしてしまうことになる。このために、ネットワーク通信にブロードキャストを用いないで負荷を分散する手法が必要になる。この章では、ネーミングサービスに階層構造を導入して管理する HCN(Hierachical Controlled Namingservice) について説明する。

3.2　HCN の概要

ネーミングサービスは分散オブジェクト環境において、非常に重要度の高いサービスであるために、分散オブジェクト環境の大規模化した場合に、負荷の集中を受けやすい場所でもある。そこで本論文では、ネーミングサービスを提供するネーミングサーバに仮想的な階層構造を導入し、負荷分散とサービスの冗長化を実現するフェデレーション手法である HCN(Hierachical Controlled Namingservice) を提案する。

HCN ではネーミングサーバの集合を、クライアントに対しては１つのネーミングサービスに見えるように仮想化する。これにより、これまでの１台のネーミングサーバでネーミングサービスを運用しているような分散オブジェクト環境でのアプリケーション開発と何ら変わりない環境を開発者に提供することができる。また、システム管理者に対しても、ネーミングサーバを冗長化するために、ネーミングサーバのダウンを原因とするシステムダウンの危険性を減少させる効果も期待できる。

HCN は ORB に HORB を用い、ネーミングサービスは HORB から提供されている HORB Naming Service を用いている。HORB は Java を開発言語として用いる ORB であるため、HCN の実装も Java を用いて行った。

3.3 HCNの構成と手法

HCNでは、クライアントからネーミングサービスに対しての登録・参照などの操作は、HCNプロキシに操作要求を発行する。要求を受けたHCNプロキシは、階層構造に従って各ネーミングサーバへ操作を行い、その結果をクライアントに返す。このような構造にすることで、クライアントからはネーミングサーバの階層構造と、そこで行われる処理を隠蔽することができる。さらに、HCNプロキシは以下の管理を行う。

- 一貫性の保証

  ネーミングサーバ間に登録されているオブジェクトが、一貫性を保っていなければならないことである。具体的には、ネーミングサーバにあるオブジェクトを参照したときに、同じキーで参照したら、同じオブジェクトを返さなければならない。

- 名前空間の管理

  ネーミングサービスにクライアントから「ある名前」でオブジェクトを登録しようとしたときに、既に「ある名前」でオブジェクトが登録されているかどうかをチェックしなければならない。もし、既に登録されているのならば、登録しようとしたクライアントに対して例外を投げる必要がある。この機構が正常に動作をしていないと、一貫性に問題が発生することになる。

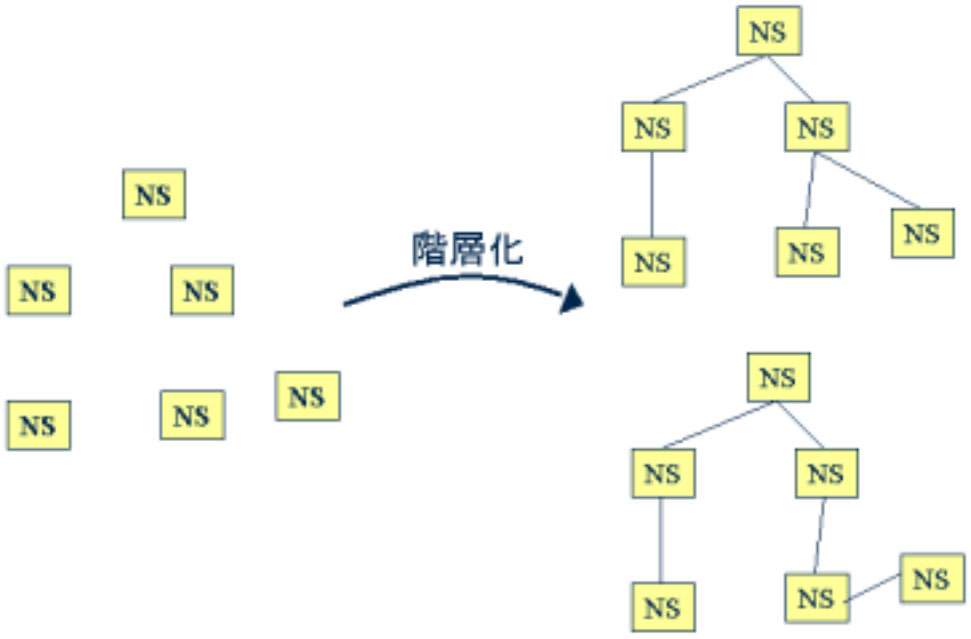

図 3.1: 階層構造の導入

3.3.1 ネーミングサービスへの階層構造の導入

HCN によるフェデレーションは、独立して動作する既存のネーミングサービスを階層化することから始まる。階層化する際の構造については、図 3.1 のようにユーザーが任意に階層構造を指定できる。

階層構造が決定されたら、クライアントが接続されているネーミングサーバから、ルートサーバまでの経路を、各々のクライアントで HCN プロキシのインスタンスを生成する際のコンストラクタに指定する。例として、図 3.2 のようにクライアントとネーミングサーバが構成されている場合では、HCN プロキシのインスタンスを生成する場合のコンストラクタ引数は以下のようになる。

```
HCNProxy hcn = new HCNProxy{"name2_1.jaist.ac.jp" , "name1.jaist.ac.jp"};
```

コンストラクタには、各ネーミングサーバの IP アドレスかドメイン名を入力する。そしてクライアントは、この HCN プロキシに対してネーミングサービスへの操作を指定する。

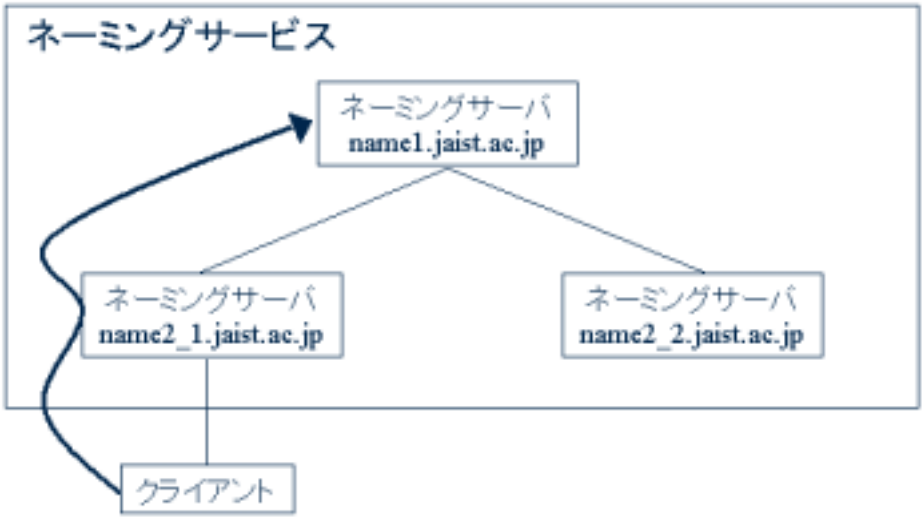

図 3.2: 階層情報の初期化

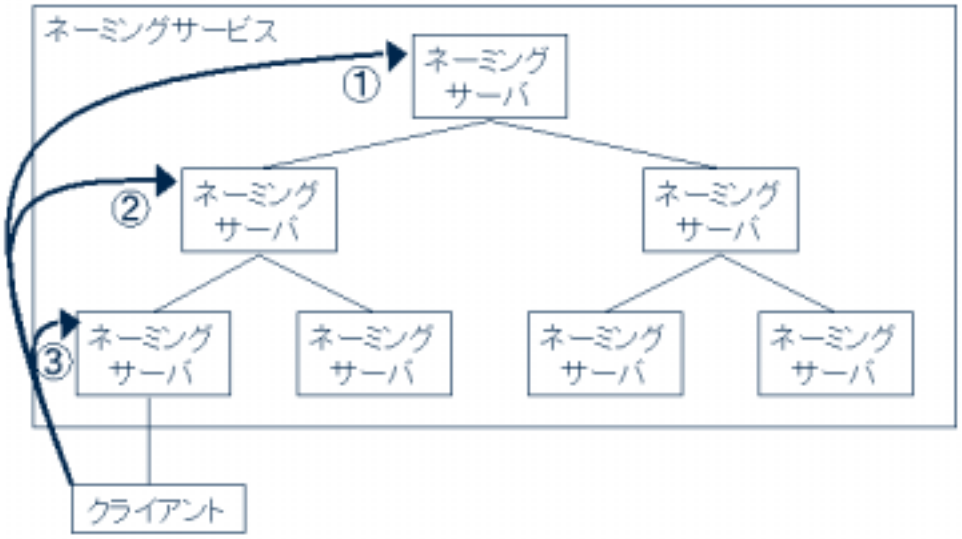

図 3.3: HCN でのオブジェクトの登録手法

3.3.2 オブジェクト登録の手法

クライアントから登録の命令を受けた HCN プロキシは、コンストラクタで指定された階層の最上位のネーミングサーバから図 3.3 に示したような順番で登録を行っていく。このような手法をとることで、ルートネーミングサーバに名前空間を統一させ、下位ネーミングサーバでの一貫性を保つことができる。これにより、もしクライアントが登録したいオブジェクトと同じ名前のオブジェクトが既に登録されていたら、ルートネーミングサーバに登録する段階で、クライアントに例外を返すことができる。また、クライアントからルートネーミングサーバまでの複数サーバに同じ内容を登録するため、あるネーミングサーバがサービス不能に陥っても、代替できるネーミングサーバが必ず存在できる。

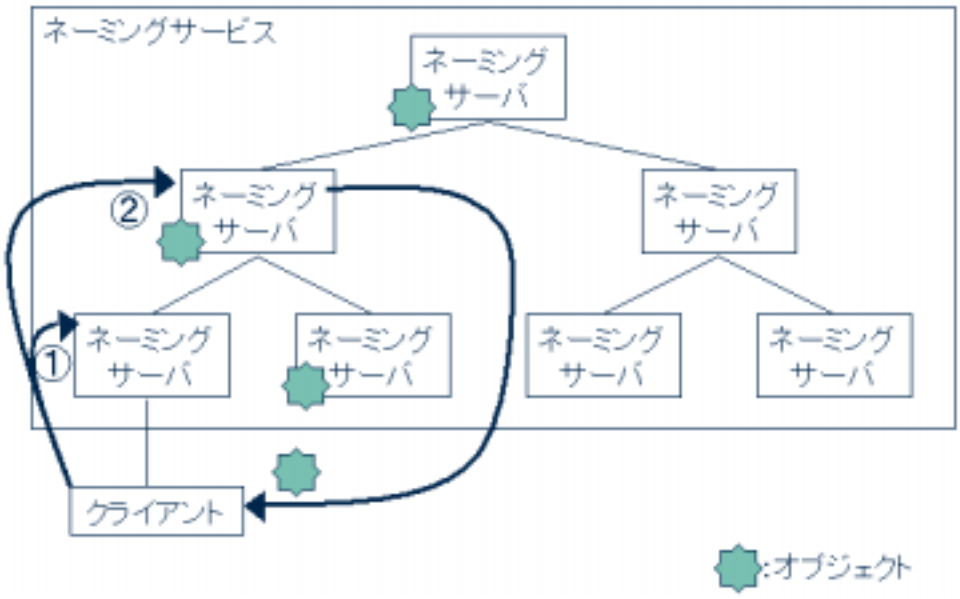

図 3.4: HCN でのオブジェクトの参照手法

3.3.3 オブジェクト参照の手法

オブジェクトを参照する場合、HCN プロキシはコンストラクタで指定された最下位のネーミングサーバから参照を開始し、クライアントの要求するオブジェクトが取得出来次第、参照処理を終了する。全ての登録内容はルートネーミングサーバに集約されているため、最悪の場合でも、ルートネーミングサーバに対して検索を行えば、オブジェクトを取得できる。

3.4 本システムが対象とする利用環境

本システムは、参照する際に上位階層まで到達しないとオブジェクトが無いような環境、つまり、ネーミングサーバの階層が深くて、参照するクライアントがネットワーク的に離れているようなリクエストに対して弱いと考えられる。そのため、あるサービスと同じネットワークドメインにあるクライアントからのリクエストが大多数で、ごくまれに外のネットワークドメインのクライアントが、あるドメインのサービスにアクセスするような利用のしかたをするアクセスパターンについて、その効果を発揮すると考えられる。

3.5 まとめ

本章では、階層構造を導入してネーミングサービスのフェデレーションを実現する HCN (Hierachical Controlled Namingservice) を提案した。まず、HCN を構成する方法と、階層構造の導入について説明した。次に、HCN の内部でオブジェクトの登録・参照がどのような手順で行われるのかを説明した。次章では、HCN のシミュレーションを用いた評価実験と、その結果について示す。

第4章　HCNを用いた大規模分散オブジェクトシステムの評価

4.1　はじめに

これまでに説明してきたネーミングサービスのフェデレーション手法であるHCNは、大規模な分散オブジェクト環境をターゲットにしたものである。そのため、HCNの評価を実際のネットワークとマシンを用いて評価することは、設備の制約から非常に困難である。そのため、シミュレーションを用いてHCNの評価を行った。

本章では、まず4.2節でシミュレーションの環境と、それを構成する要素の動作について説明する。そして、このシミュレーション環境を用いたシミュレーション結果を示し、得られた結果について考察する。

4.2　シミュレーション環境

本研究では、シミュレーションにより、HCNの有効性を示す。今回構築したシミュレーション環境の概要を図4.1に示す。シミュレーション環境はクライアント、ネーミングサーバ、サービス、ネットワークで構成されている。全体の動作は、まず待機状態のクライアントは、ある確率で活動を開始する。活動を開始したクライアントは、設定されたサービスを受けるためのプロキシを取得するために、ネーミングサーバへリクエストを発行する。ネーミングサーバからプロキシを取得することができたら、サービスに対してリクエストを発行し、クライアントがサービスから処理結果を受け取った段階で１つのリクエストの処理が完了する。これら一連の動作はステップ毎に区切られて実行される。

4.2.1　クライアントの動作

クライアントが動作する際の状態遷移図を図4.2に示す。クライアントはパラメータとして、待機状態から活動状態へ移行する遷移確率と、ネーミングサーバの階層構造を持っている。最初に待機状態にあるクライアントは、ある遷移確率にしたがって待機状態からネーミングサービスに問い合わせる活動状態に遷移する。活動状態に入ると、要求するサービスに対してのプロキシを取得するために、ネーミングサービスに問い合わせる。もし要求するサービスに対するプロキシが、現在問い合わせを行っているネーミングサービ

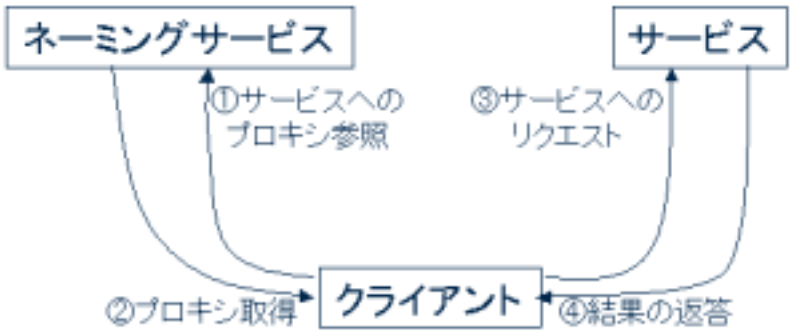

図 4.1: シミュレーション環境

スに含まれていない場合、保持しているネーミングサーバの階層構造にしたがって、上位のネーミングサービスに問い合わせを行う。プロキシが取得できたら、次にサービスへリクエストを送る。サービスが終了し、レスポンスが返って来た時点で待機状態にもどる。

4.2.2 サービスの動作

サービスが動作する際の状態遷移図を図 4.3 に示す。待機状態にあるサービスは、常に自身に接続されている仮想ネットワークを監視している。サービスがリクエストを検知すると、処理状態に移行する。処理が終了すると仮想ネットワークに対してクライアントへの返答を発行する。この時点で待機状態に移行し、次のリクエストが到着するまで待機する。

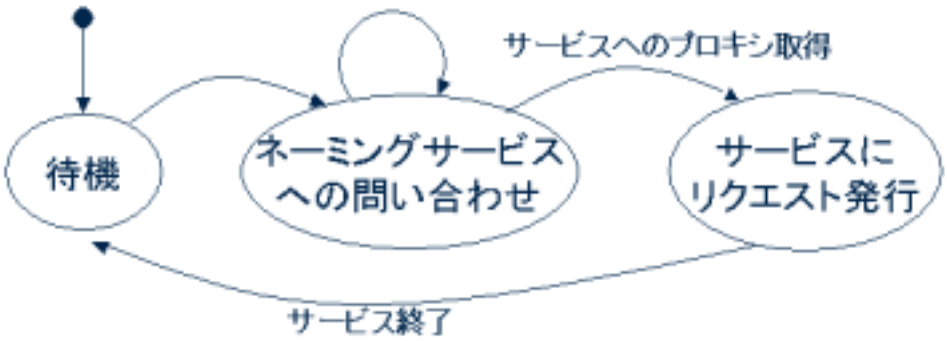

図 4.2: クライアントの状態遷移図

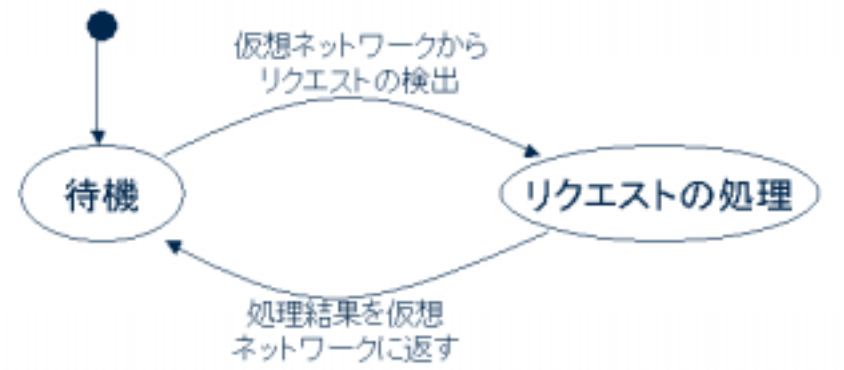

図 4.3: サービスの状態遷移図

4.3 システム評価

これまでに説明したシミュレーション環境を用いて、HCN の評価を行った。評価内容として、リクエスト 1 件の処理時間とシステムで処理された総リクエスト件数を測定した。

以下の図 4.4 から図 4.8 に、ネーミングサーバの階層の状態と、クライアントーネーミングサーバ間、クライアントーサービス間のネットワークディレイを示している。以下に実験の概要を示す。

- ネーミングサーバは二分木状に階層構造を形成しているものとし、クライアントは、階層構造の最末端のネーミングサーバからルートに向けて問い合わせを行うものとしている。
- サービスに対してのプロキシはネーミングサービスの階層構造の一番左側の経路に沿って登録されている。サービスを提供するサーバは 8 個用意し、クライアントは 8 個のサービス各々に均等にリクエストを送信する。
- クライアントは 256、1024、4096 個の場合で実験を行ない、末端のネーミングサーバからルートのネーミングサーバまでの各経路に対して、クライアントの個数が均等になるように配置されている。
- 1 ステップは実時間で 10msec に相当するものとして、各パラメータを決定している。ネーミングサービスのリクエスト処理時間は 2 ステップ、サービスのリクエスト処理時間は 10 ステップとした。

便宜上、従来法であるネーミングサーバ 1 個でネーミングサービスを提供する手法を、1 階層の HCN と呼ぶことにする。

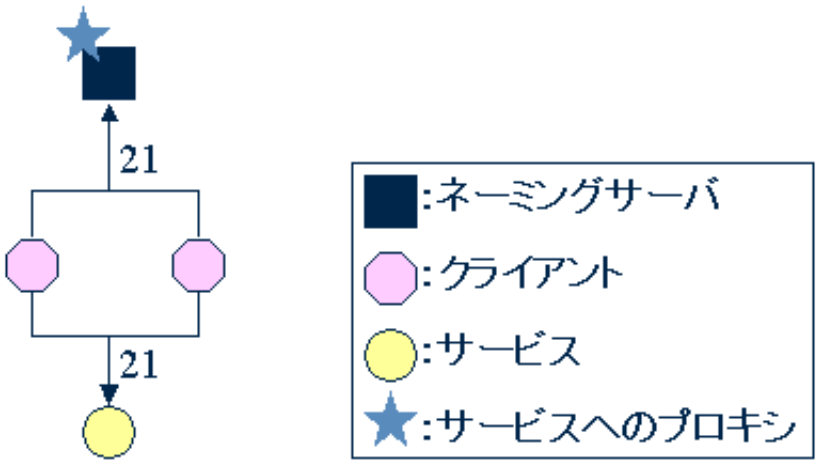

図 4.4: ネーミングサーバが 1 個の場合 (1 階層 HCN)

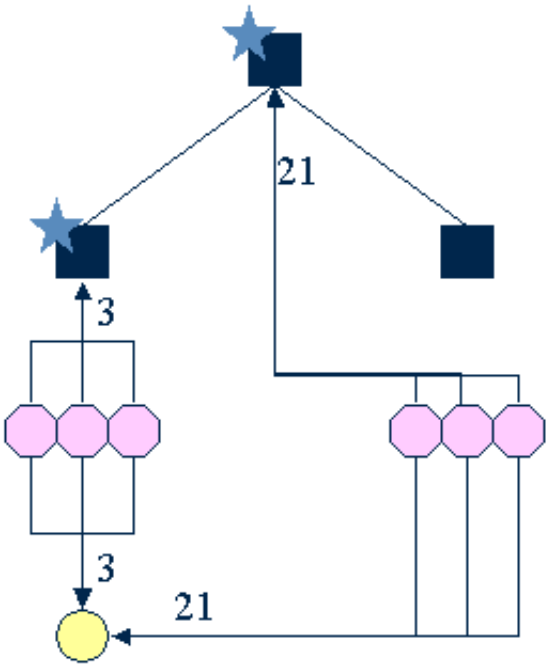

図 4.5: 2 階層 HCN の場合

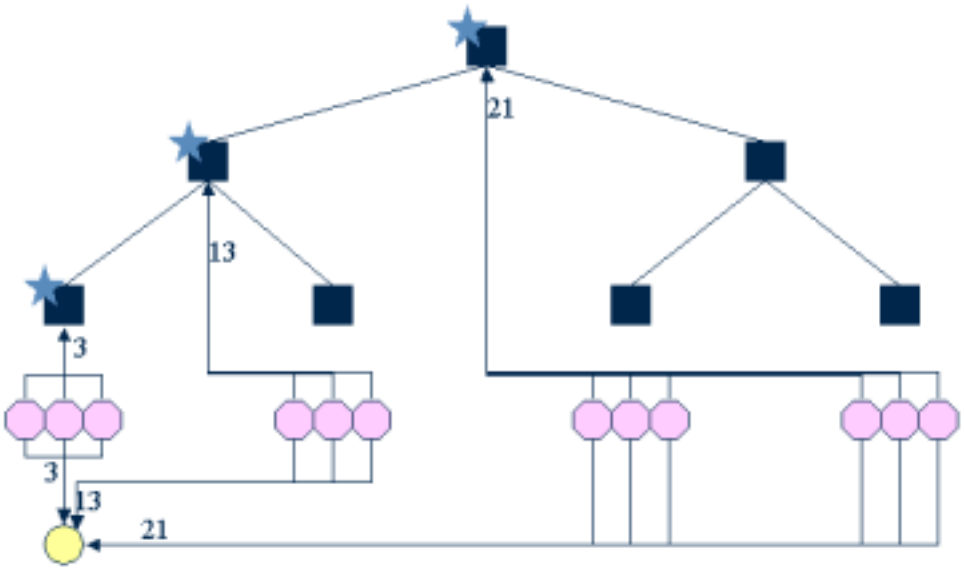

図 4.6: 3 階層 HCN の場合

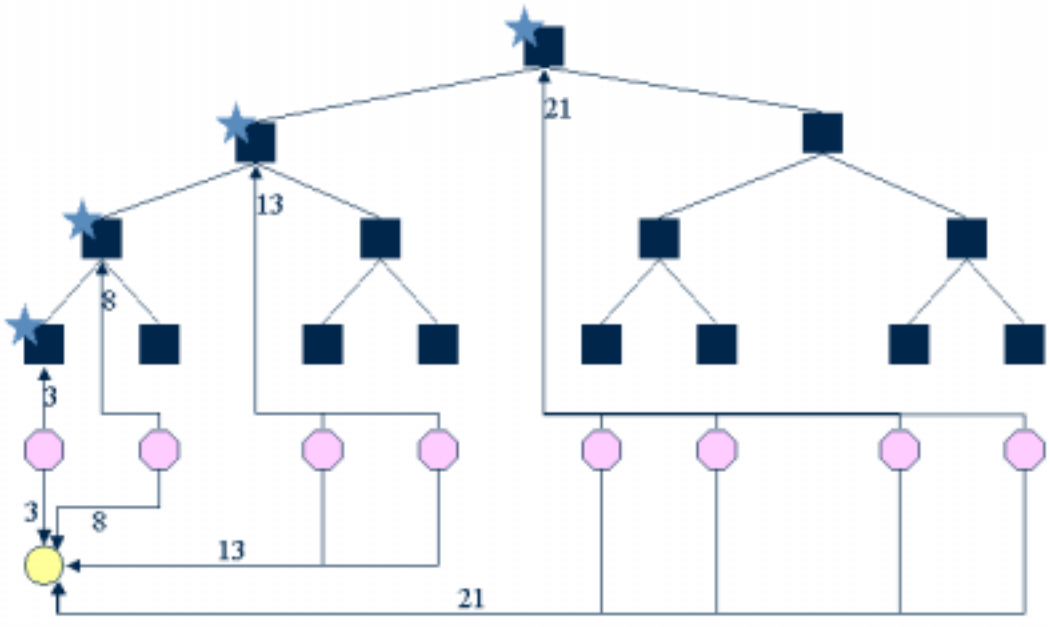

図 4.7: 4 階層 HCN の場合

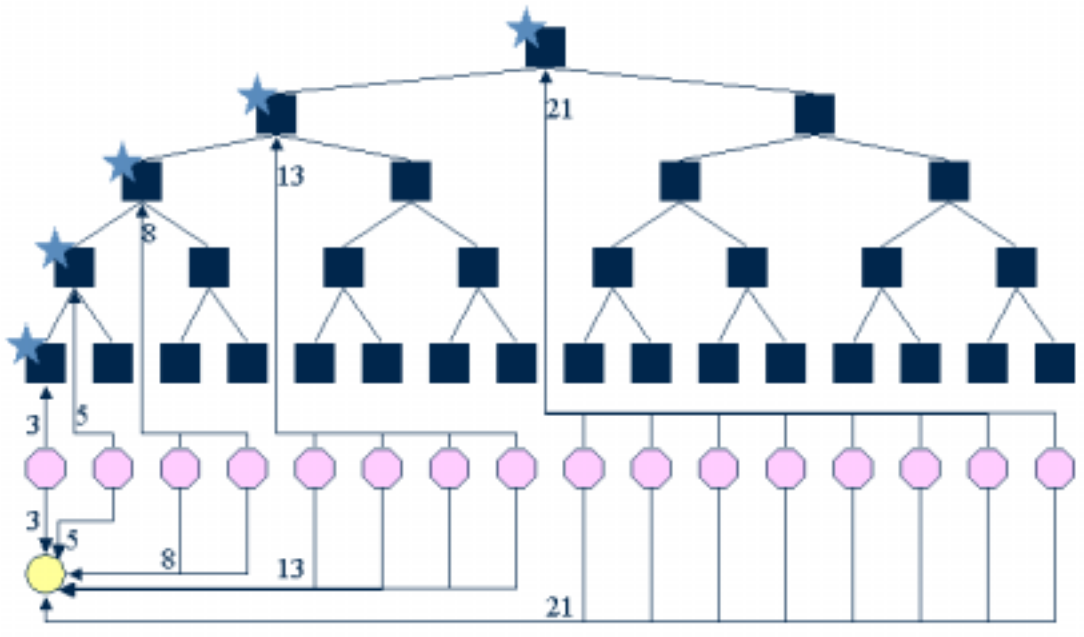

図 4.8: 5 階層 HCN の場合

4.3.1 クライアント数 256 個の場合

比較的小規模な分散環境の下で HCN の性能を評価するため、クライアントが 256 個の場合の実験を行った。図 4.9 から図 4.13 では、各階層数の場合でクライアントの発行したリクエストの平均処理時間をクライアント毎に示している。処理時間に占める、ネーミングサービスでのリクエスト処理時間と、サービスでのリクエスト処理時間の割合も示してある。図 4.9 では 1 台のネーミングサーバでネーミングサービスを提供しているため、全てのクライアントにおいて、均一な時間でリクエストの処理が行われるが、2 階層の場合である図 4.10 では、サービスのプロキシを持っているネーミングサーバへすぐにアクセスできるクライアントと、できないクライアントが存在するために、リクエスト処理時間に差が生じている。このような現象は 3 階層以降の場合でも発生していることが図 4.11 から図 4.13 に示されている。

図 4.14 では、各階層数において全クライアントの平均リクエスト処理時間を、図 4.15 では平均リクエスト処理数を示している。図 4.14 の結果を見ると、HCN を 2 階層用いた場合にリクエスト処理時間は最小になっているが、従来法である 1 階層の場合に比べて大きな有意差が見られなかった。また、図 4.15 からも、HCN の階層数が増加しても、処理リクエスト件数はほぼ変わらない結果が示されている。以上の結果から、クライアント数が 256 個の場合においては、HCN の導入における有意差は認められないことがわかる。

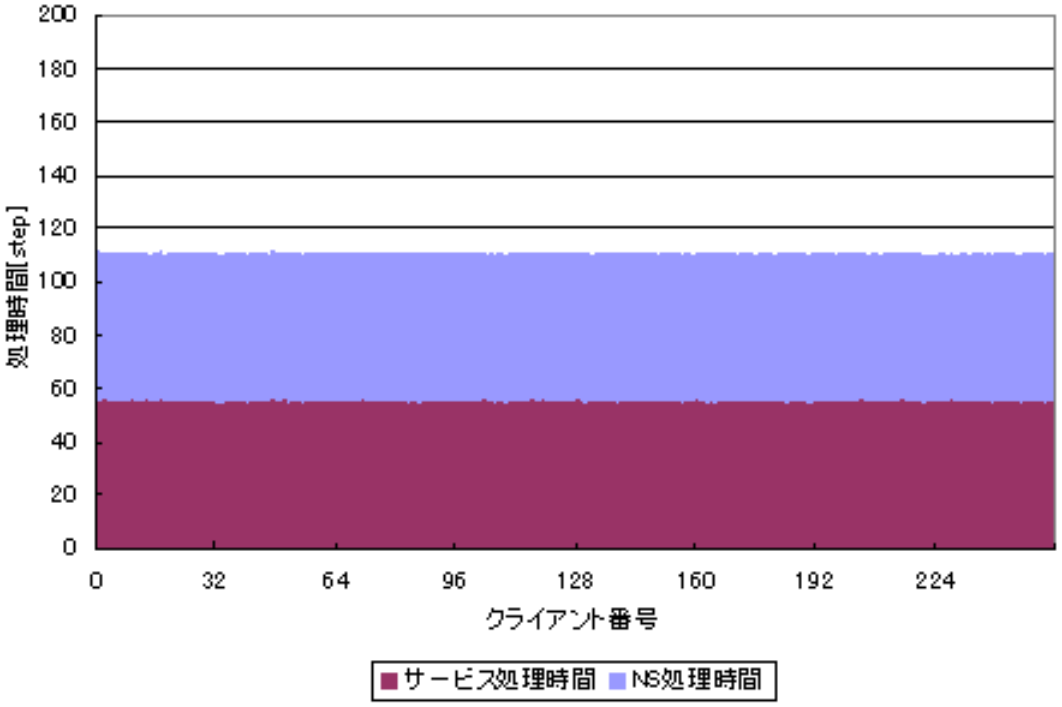

図 4.9: ネーミングサーバが 1 個の場合の平均リクエスト処理時間 (1 階層 HCN)

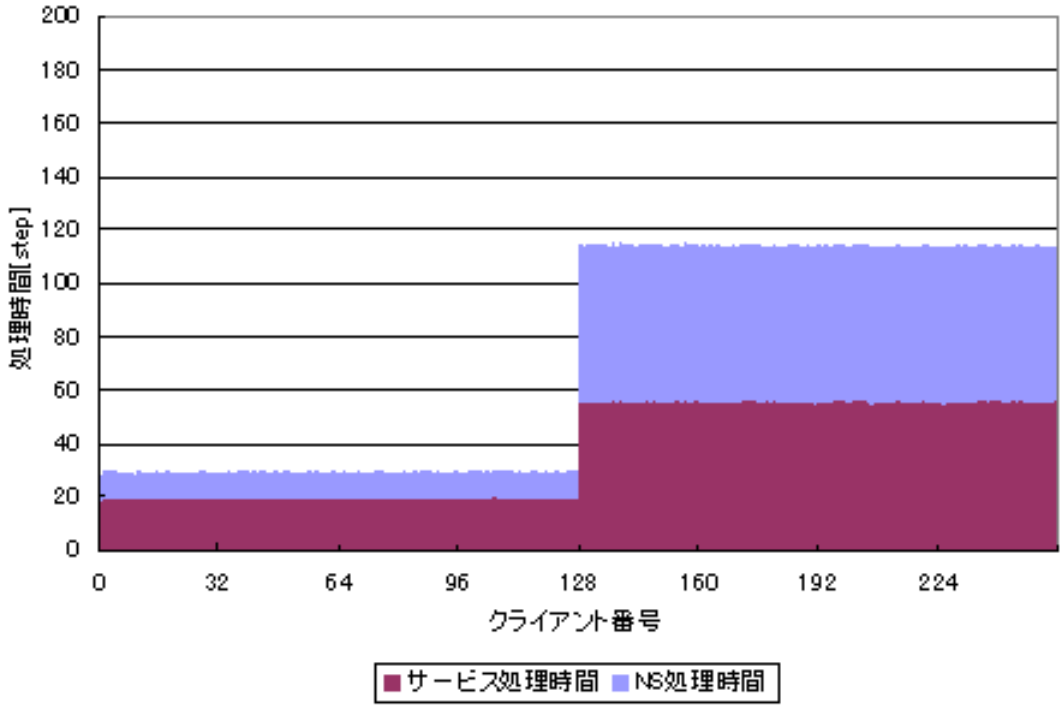

図 4.10: 2 階層 HCN での各クライアントの平均リクエスト処理時間

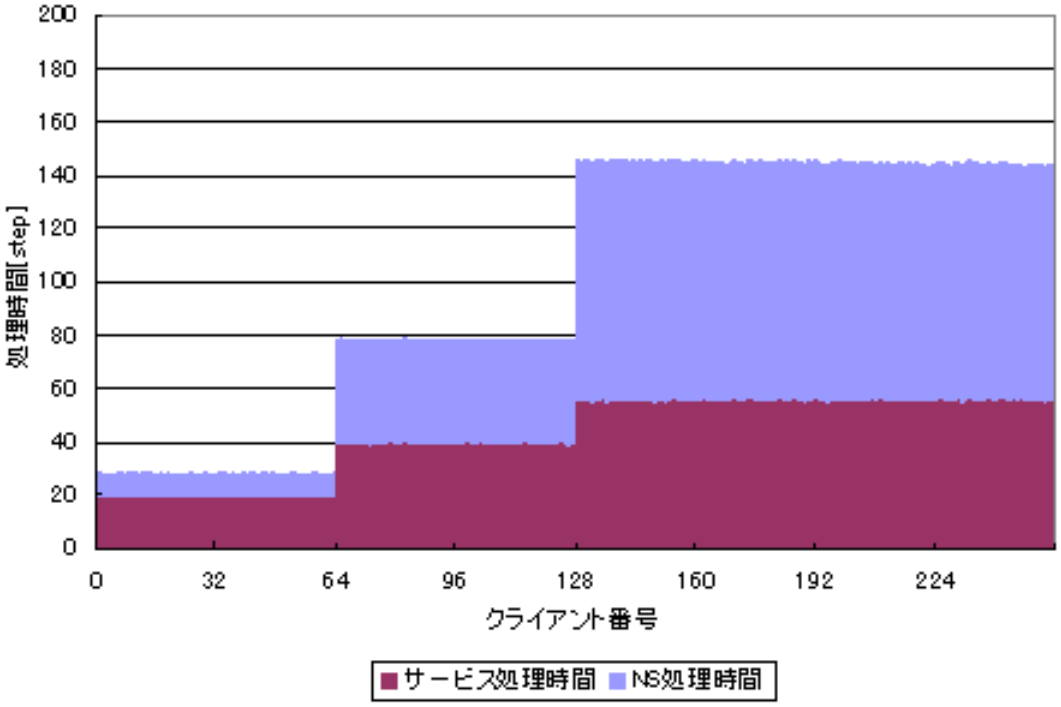

図 4.11: 3 階層 HCN での各クライアントの平均リクエスト処理時間

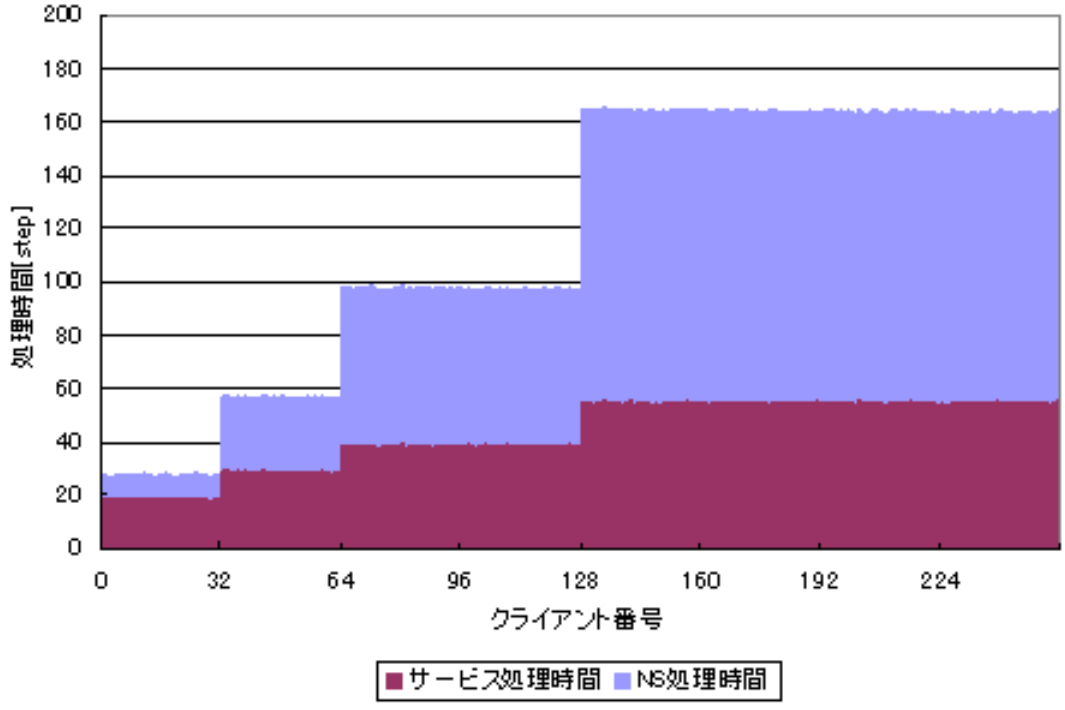

図 4.12: 4 階層 HCN での各クライアントの平均リクエスト処理時間

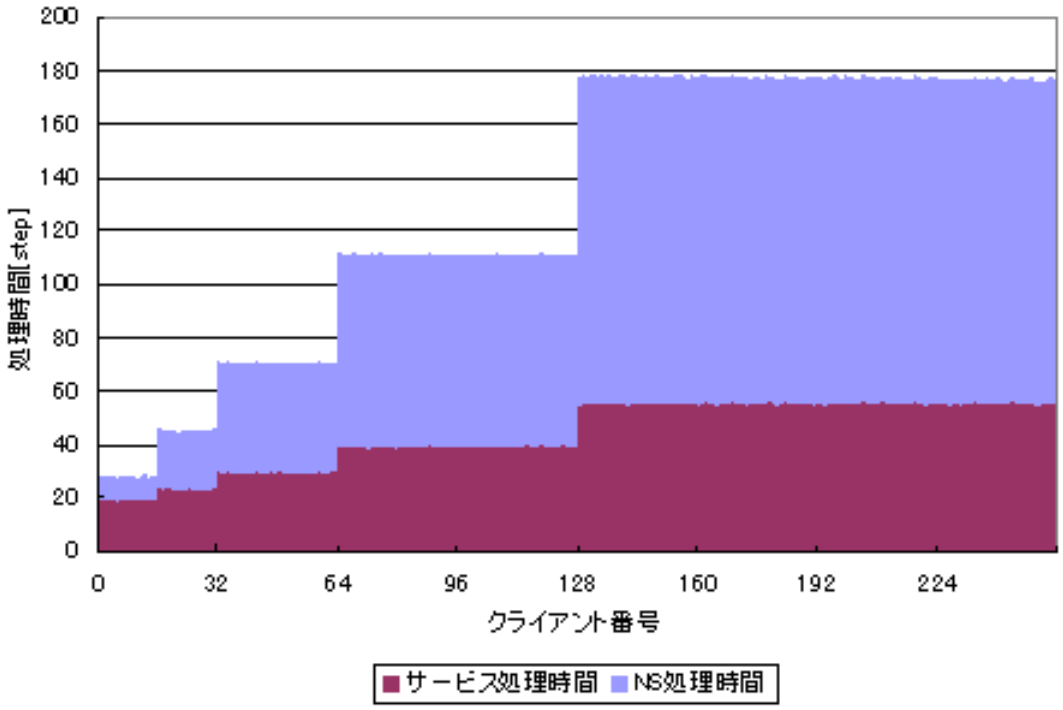

図 4.13: 5 階層 HCN での各クライアントの平均リクエスト処理時間

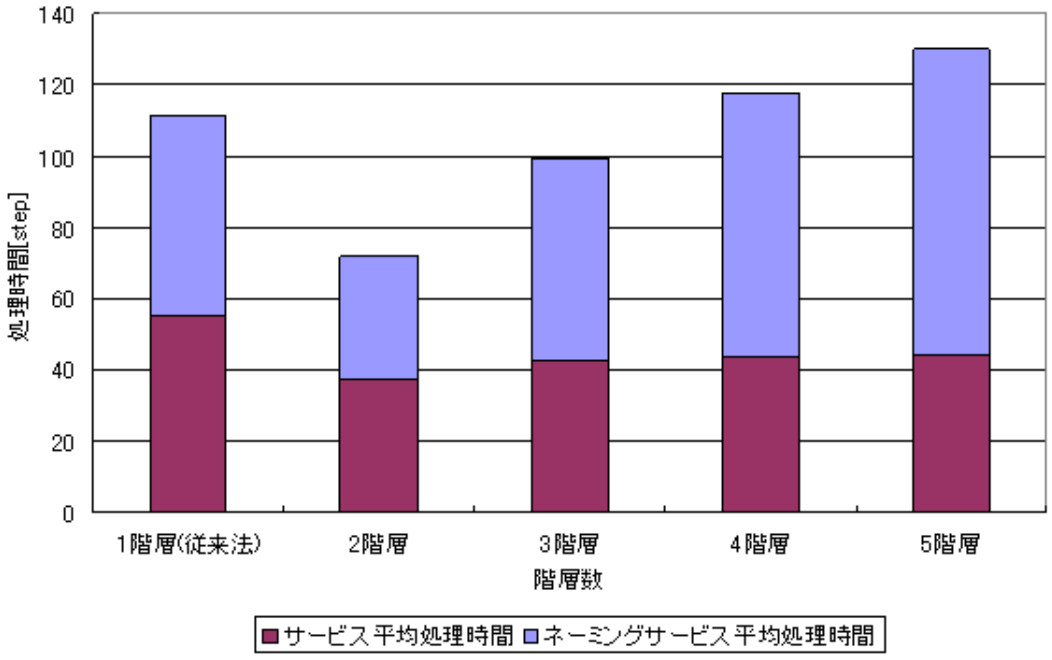

図 4.14: 各階層における平均リクエスト処理時間

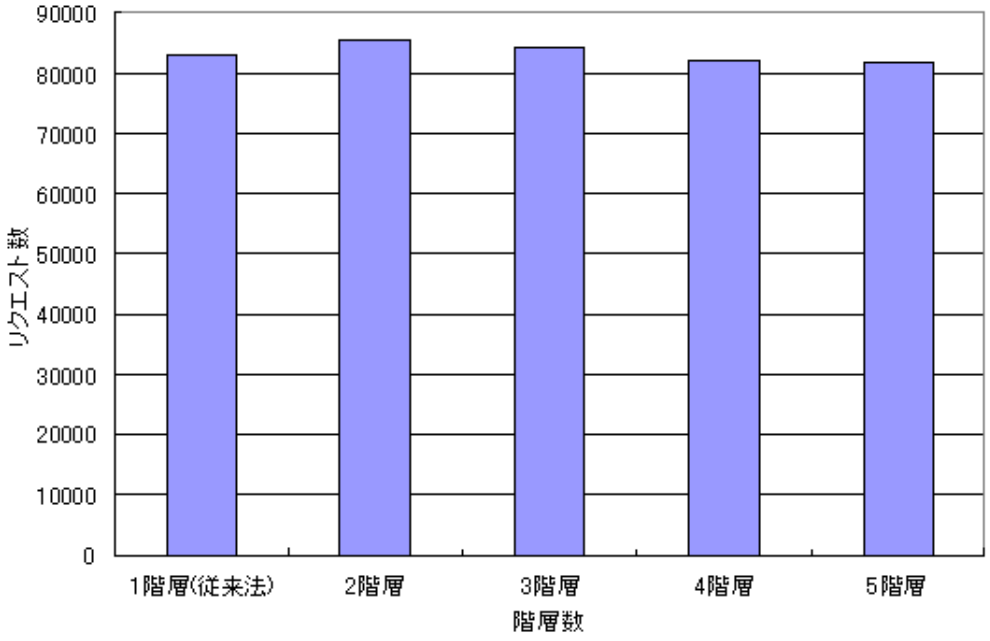

図 4.15: 各階層における総リクエスト処理数

4.3.2 クライアント数 1024 個の場合

次に、クライアント数を 1024 個に増やし、中規模の分散環境を想定した実験を行った。図 4.16 から図 4.20 では、各階層数の場合でクライアントの発行したリクエストの平均処理時間をクライアント毎に示している。処理時間に占める、ネーミングサービスでのリクエスト処理時間と、サービスでのリクエスト処理時間の割合も示してある。これらの結果は、256 個のクライアントを用いて実験を行った場合と同様に、サービスのプロキシを取得する際に、時間がかかるクライアントとすぐに取得できるクライアントとの間で、ネーミングサービスの処理時間に差が生じている。クライアントが増加した分、発生するリクエストも増加するため、256 個のクライアントを用いた実験よりも、ネーミングサービスの処理時間が長くなっている。

図 4.21 では、各階層数において、全クライアントの平均リクエスト処理時間を示している。図 4.21 のグラフから、HCN の階層数が 2 階層と 3 階層の場合で、サービス処理時間とネーミングサービス処理時間が逆転している。この現象は、2 階層の場合においてはネーミングサービスの処理速度がボトルネックとなっているために、サービスでの待ちが発生せず、すぐに処理が完了していることを意味し、逆に 3 階層の場合においては、ネーミングサービスの処理がすぐに完了する一方で、サービスの処理の際に待ち時間が発生している事を示している。つまり 2 階層と 3 階層の間で、処理のボトルネックがネーミングサービスからサービスへと移行していることが示唆されている。

図 4.22 からも、3 階層の HCN までは処理リクエスト件数は増加しているが、4 階層からは処理件数が頭打ちとなっているため、サービスの処理が限界に達していると考えられる。これらの結果から、3 階層の HCN を導入することにより、ネーミングサービスのボトルネックを解消することができる。

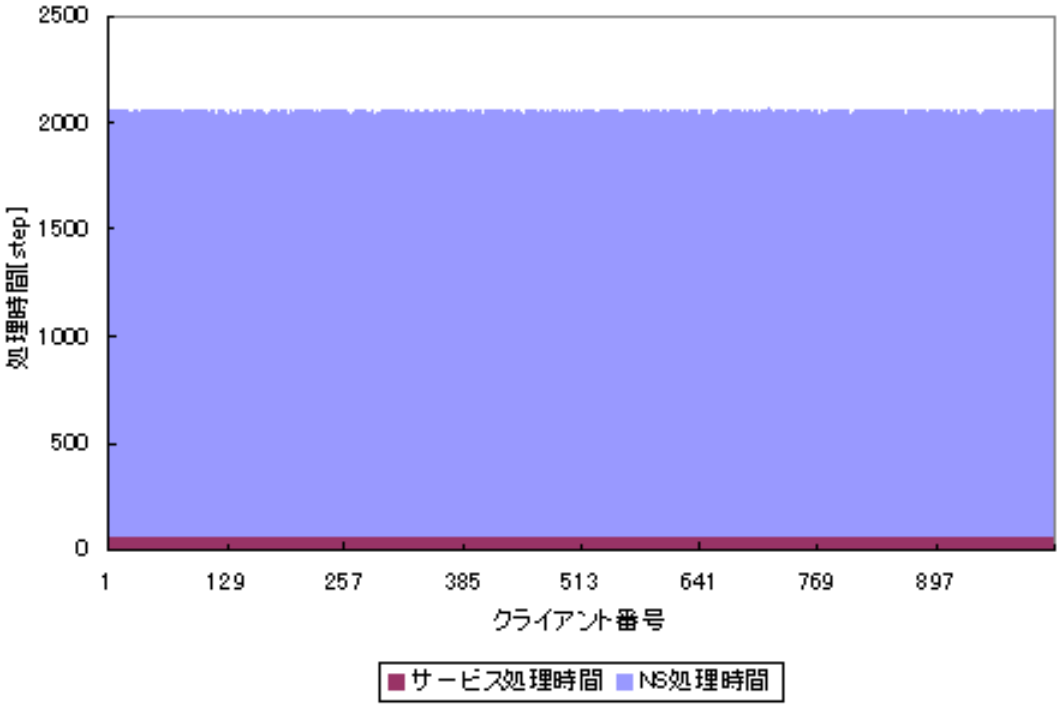

図 4.16: ネーミングサーバが 1 個の場合の平均リクエスト処理時間 (1 階層 HCN)

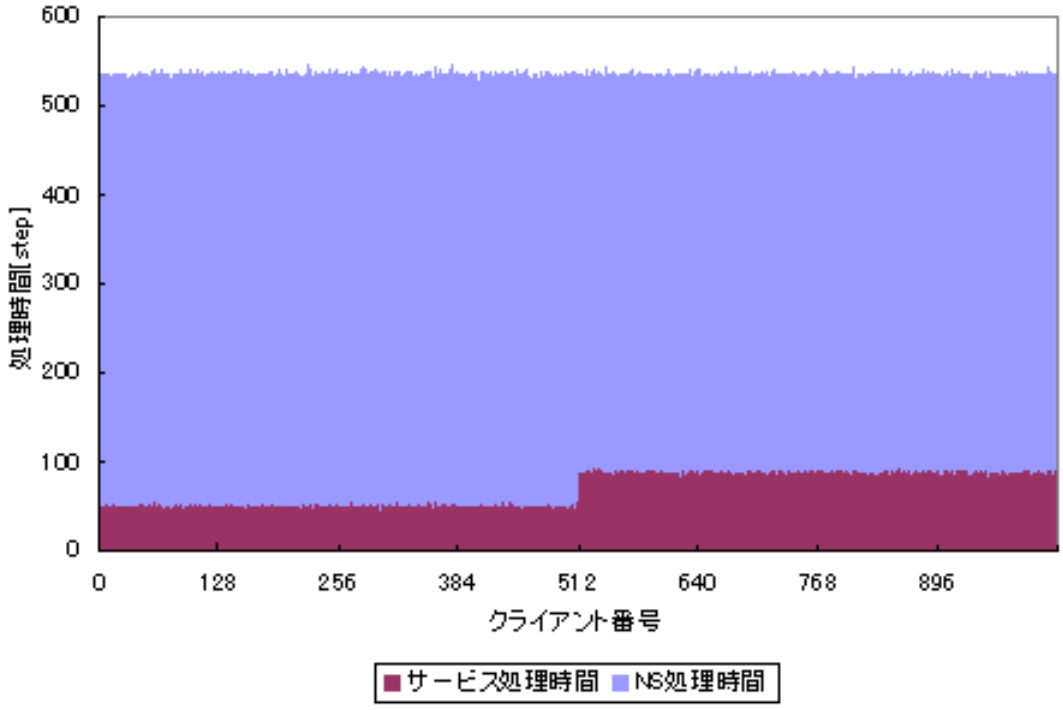

図 4.17: 2 階層 HCN の場合での各クライアントの平均リクエスト処理時間

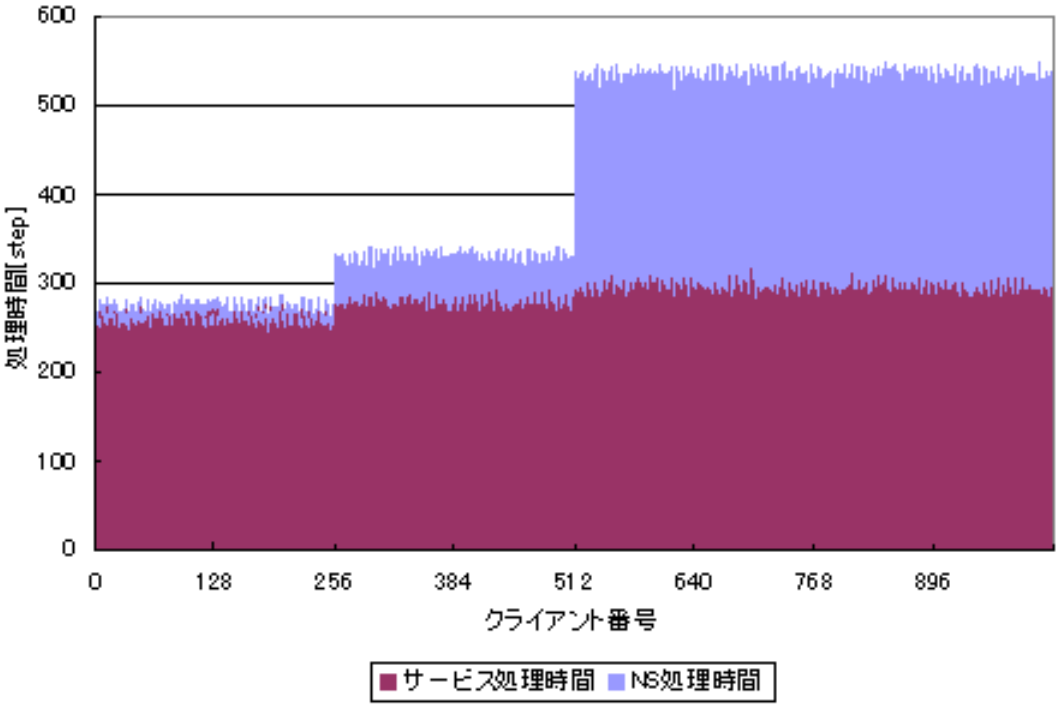

図 4.18: 3 階層 HCN の場合での各クライアントの平均リクエスト処理時間

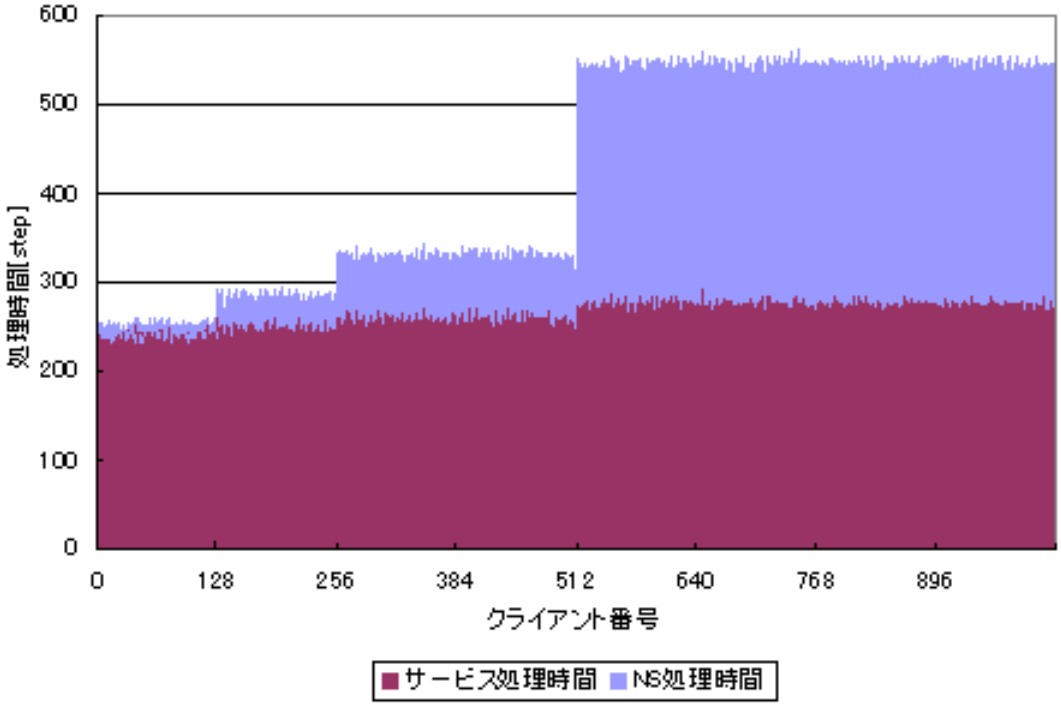

図 4.19: 4 階層 HCN の場合での各クライアントの平均リクエスト処理時間

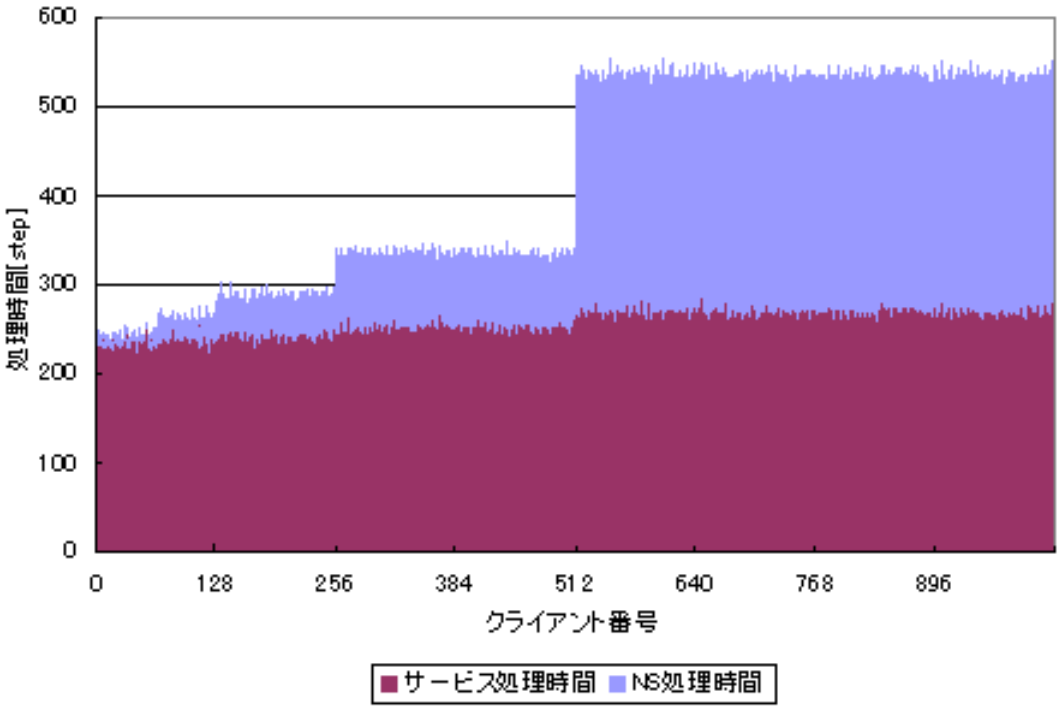

図 4.20: 5 階層 HCN の場合での各クライアントの平均リクエスト処理時間

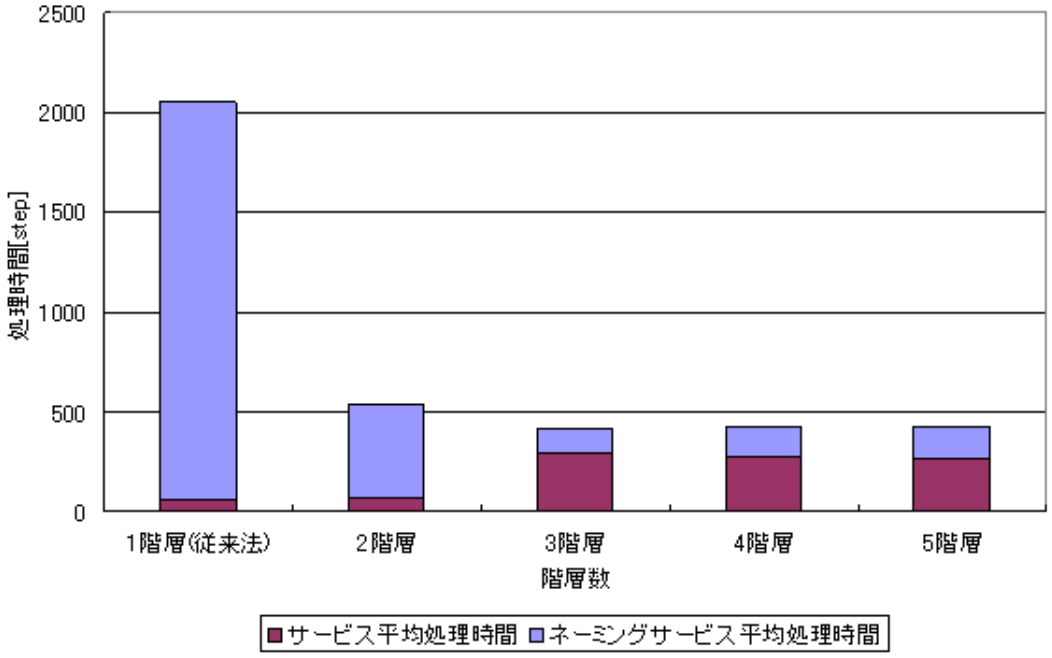

図 4.21: 各階層における平均リクエスト処理時間

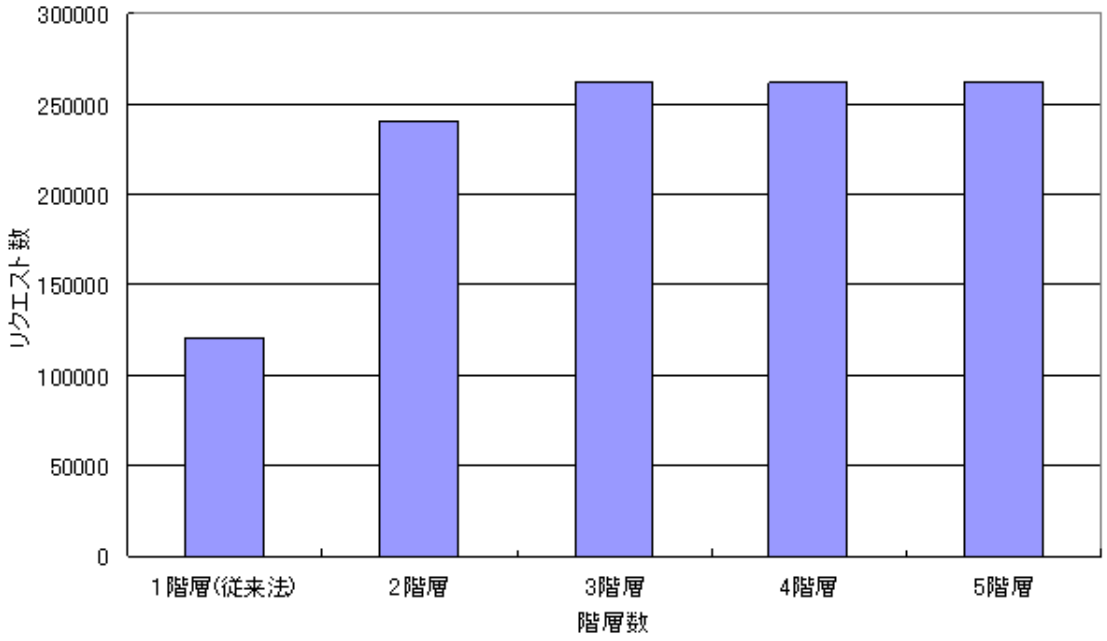

図 4.22: 各階層における総平均リクエスト処理時間

4.3.3 クライアント数 4096 個の場合

次に、クライアント数を 4096 個として、大規模な分散環境の下での実験を行った。これまでの実験と同様に、図 4.23 から図 4.27 では、各階層数の場合でクライアントの発行したリクエストの平均処理時間をクライアント毎に示している。処理時間に占める、ネーミングサービスでのリクエスト処理時間と、サービスでのリクエスト処理時間の割合も示してある。1 階層および 2 階層の HCN を用いた結果である図 4.23 と図 4.24 では、リクエスト処理のほとんどがネーミングサービスの処理時間になっている。しかし、3 階層以降の結果である図 4.25 から図 4.27 では、リクエストの処理時間は、サービスの処理時間が大半を占めるようになる。

図 4.28 では、各階層数において全クライアントの平均リクエスト処理時間を、図 4.29 では平均リクエスト処理数を示している。4096 個までクライアント数が増加すると、1 階層の場合、つまり、ネーミングサービスに対して負荷分散を適用していない場合においては、完全にネーミングサービスが破綻しており、ほとんどリクエストを処理することができなくなっている。この場合でも、サービス処理時間とネーミングサービス処理時間が 2 階層と 3 階層の場合で逆転している。

このような結果から、3 階層の HCN を用いることにより、ネーミングサービスのボトルネックを解消することが可能になる。

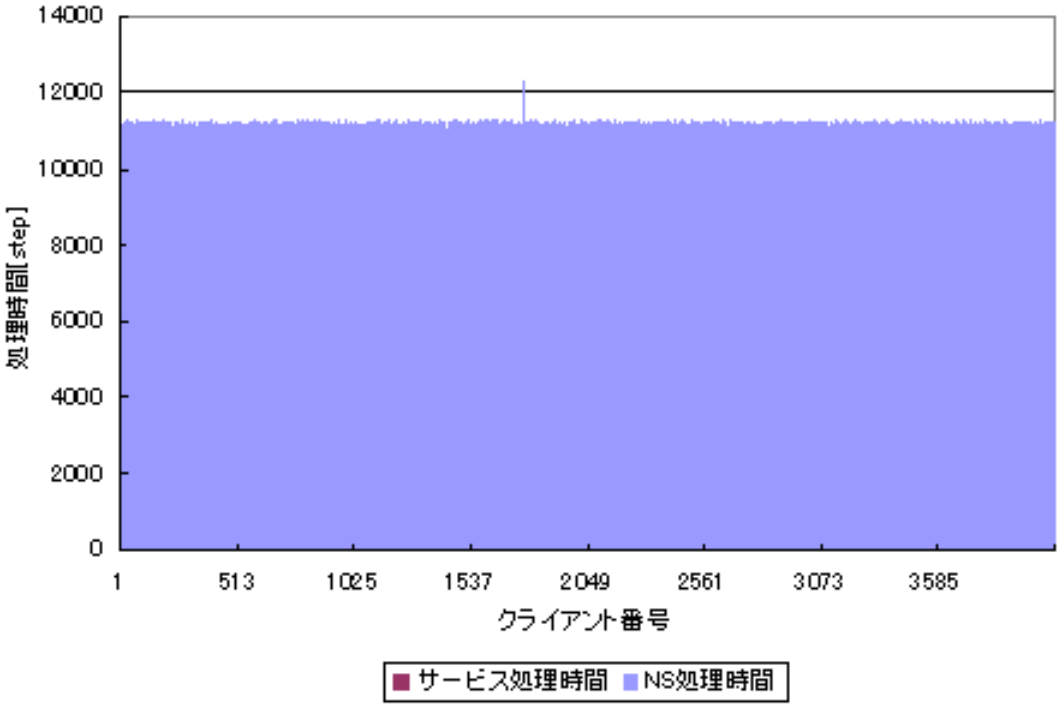

図 4.23: ネーミングサーバが 1 個の場合の平均リクエスト処理時間 (1 階層 HCN)

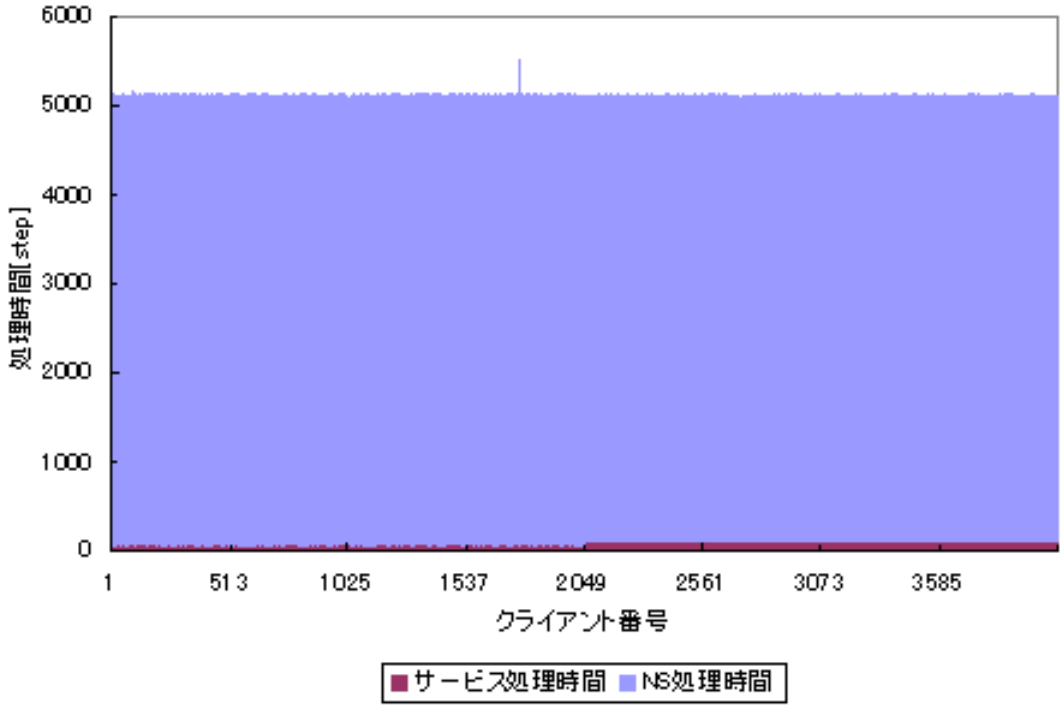

図 4.24: 2 階層 HCN の場合での各クライアントの平均リクエスト処理時間

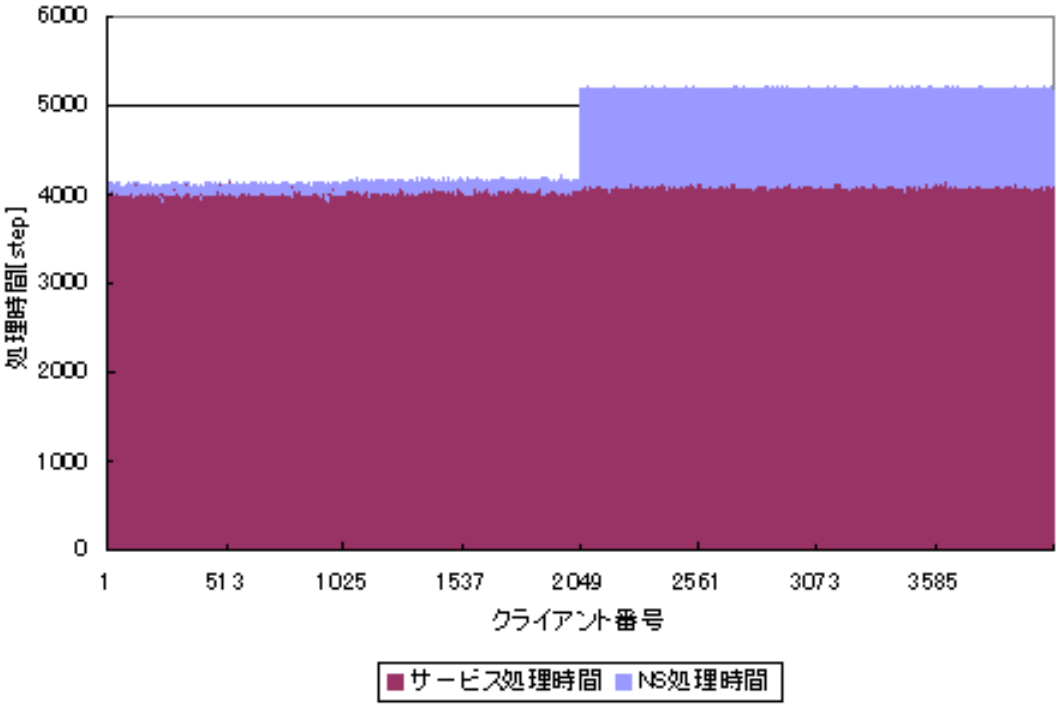

図 4.25: 3 階層 HCN の場合での各クライアントの平均リクエスト処理時間

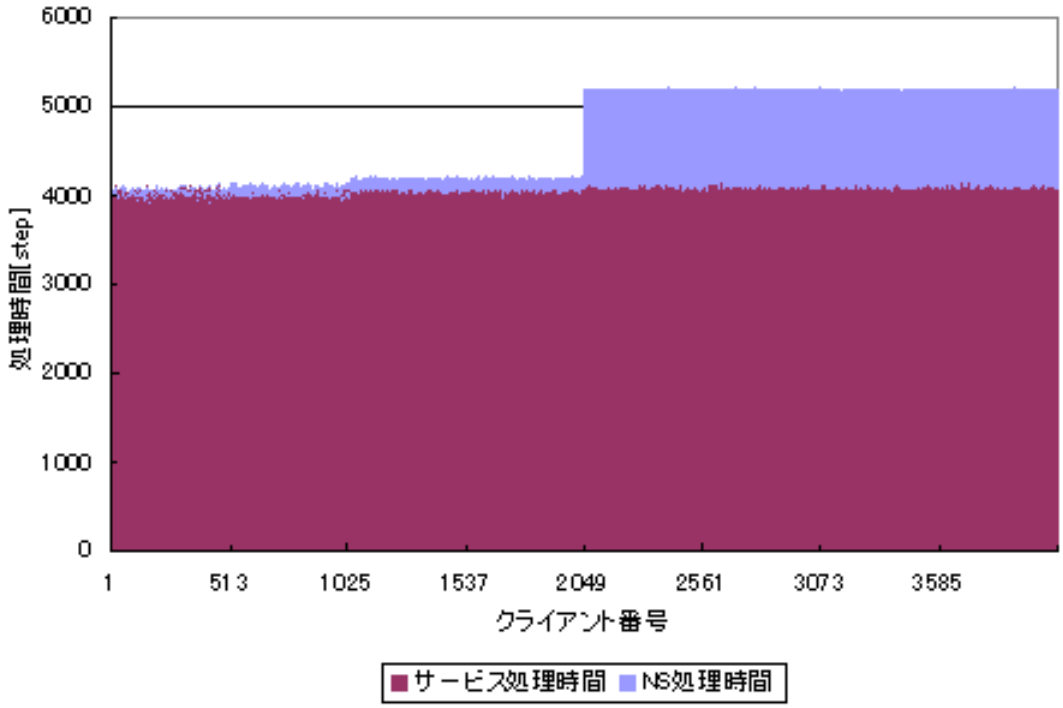

図 4.26: 4 階層 HCN の場合での各クライアントの平均リクエスト処理時間

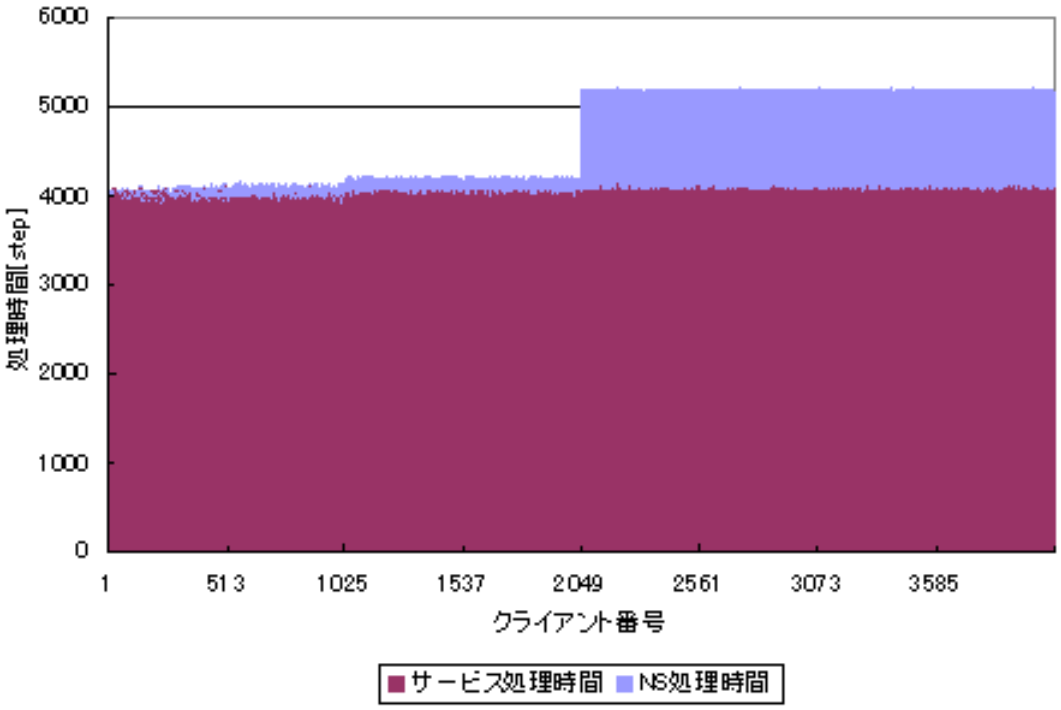

図 4.27: 5 階層 HCN の場合での各クライアントの平均リクエスト処理時間

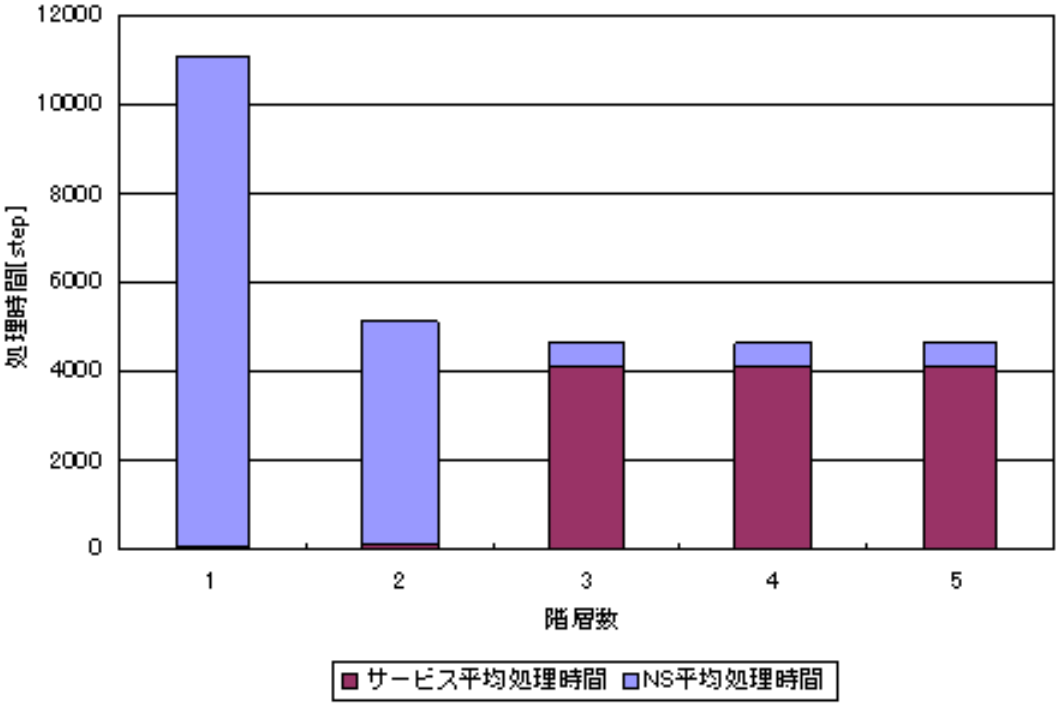

図 4.28: 各階層における全クライアントの平均リクエスト処理時間

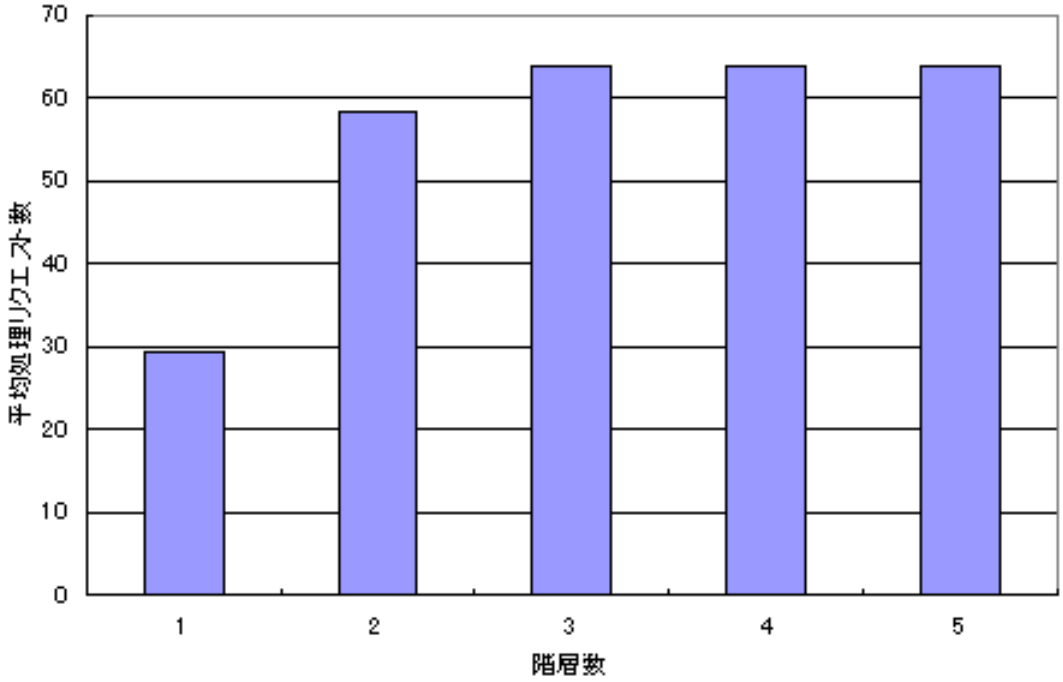

図 4.29: 各階層における全クライアントの平均リクエスト処理数

4.4 結果

これまでに行った実験により、各 HCN 階層において、リクエスト 1 件の処理に要する時間をまとめたものを図 4.30 に、システムにおいて処理されたリクエスト件数をまとめたものを図 4.31 に示す。これらの結果から、今回のシミュレーションパラメータの下では、従来法と比較して、HCN を導入したことにより、リクエスト処理時間、リクエスト処理件数共に改善されたことが示された。また、ネーミングサービスが原因となって発生する、システムの性能低下を回避するためには、3 階層の HCN を導入することで十分な改善が行われることが示された。

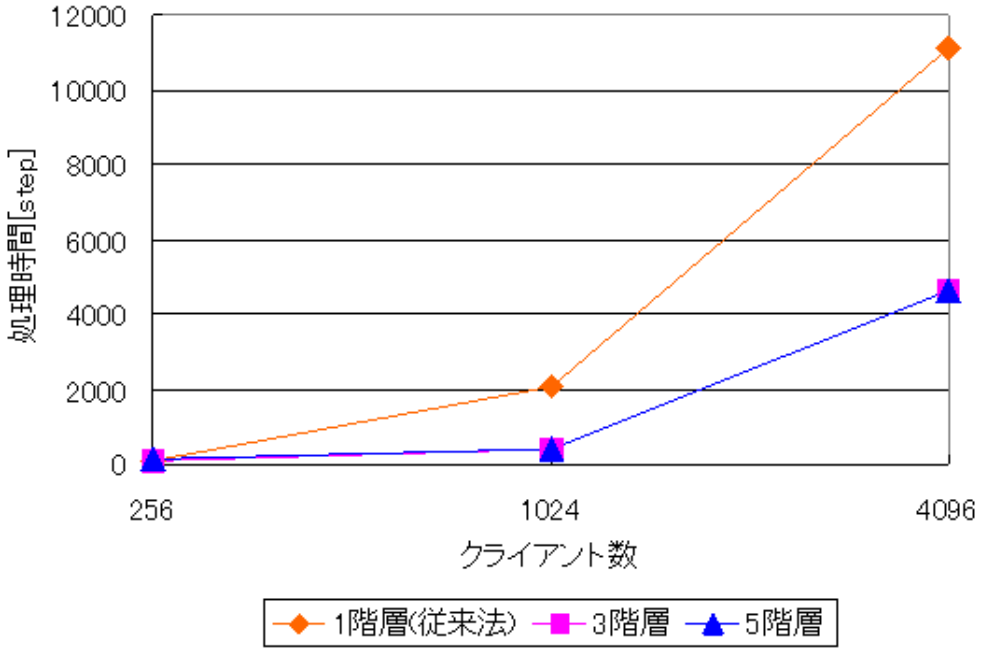

図 4.30: クライアント数に対する平均リクエスト処理時間

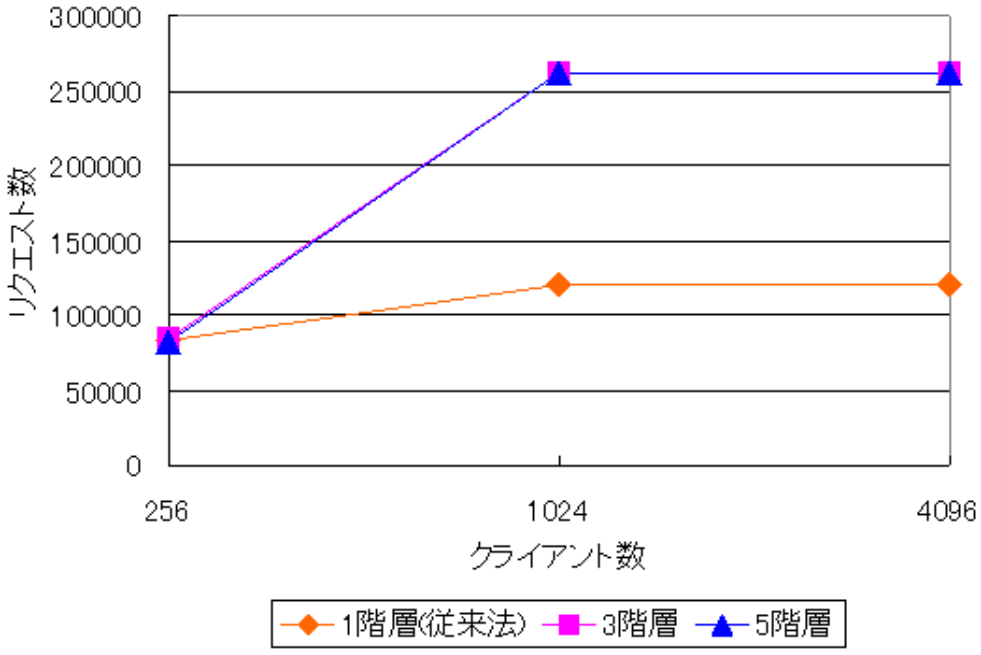

図 4.31: クライアント数に対する総リクエスト処理数

4.5 まとめ

本章では、シミュレーションを用いてHCNの評価を行った。シミュレーション環境の全体的な説明と、シミュレーション環境を構成するクライアント、サービスなどの動作を説明し、実際にHCNの評価を行った。

比較的小規模な環境の場合は、HCNを導入したことによる従来法に対しての大きな有意差は認められなかった。しかし、中から大規模な環境になった場合に、1リクエストに占めるネーミングサービスの処理時間とサービスの処理時間が逆転することが確認された。

このことから、クライアント数が1024以上の大規模な環境の下ではHCNがネーミングサービスの負荷分散に有効な手法であることが認められた。

第5章　結論

5.1　まとめ

ネーミングサービスは、様々な種類があるサービスの中でも、分散オブジェクト環境を構築する際によく用いられるサービスである。このような重要なサービスに対して、大規模な環境でも十分に負荷分散が可能なネーミングサービスのフェデレーション手法であるHCN を提案し、シミュレーション環境を用いて性能評価を行った。

2 章では、分散オブジェクトの基礎となるオブジェクト指向プログラミングについて説明し、分散オブジェクト環境を構築するためのフレームワークと分散オブジェクト環境を支える各種サービスについて述べた。

3 章では HCN を提案し、HCN を構築するための階層構造を導入する手法と内部での動作について説明した。

4 章では性能評価のために構築したシミュレーション環境について、その概要と構成要素の動作について説明した後、実際にシミュレーション環境を用いて HCN の有効性を評価した。HCN を導入することで、クライアント数が増加した際にも負荷分散を行うことで、ネーミングサービスがボトルネックになることで発生するシステム全体のスループットの低下を回避できることが明らかになった。

5.2　今後の予定

本論文で提案した HCN では、オブジェクトを登録する際にそれを保持するサーバは、登録を行うクライアントが接続されたネーミングサーバからルートネーミングサーバまでのパスだけである。この手法の弱点として、ルートネーミングサーバまで問い合わせを行わないと、オブジェクトが取得できないような場合、どうしても時間がかかってしまう。そこで、上位のネーミングサーバの内容を下位サーバでキャッシングする手法が有効だと考えられる。しかし、この手法の実現のためにはキャッシュされたものが元のオブジェクトと異なってしまう可能性が発生する。今後は、下位サーバでキャッシングを行なった場合でも一貫性が保たれるようなネーミングサービスフェデレーションの手法について検討する。

謝辞

本研究を行なうにあたり、御指導、御鞭撻をいただいた北陸先端科学技術大学院大学 堀口 進教授に深く感謝致します。

北陸先端科学技術大学院大学 白川 英二 助教授 と 土本 晃久 助手には、サブテーマで熱心に御指導いただき、深く感謝申し上げます。

北陸先端科学技術大学院大学 林 亮子 助手には、貴重なコメントをいただき、深く感謝申し上げます。

北陸先端科学技術大学院大学 福士 将 助手には、有益な御助言をいただき、深く感謝申し上げます。

最後に、日頃よりお世話になったマルチメディア統合システム講座の皆様に厚くお礼申し上げます。

参考文献

[1] 高木 浩光, 松岡 聡, 中田 秀基, 関口 智嗣, 佐藤 三久, 長嶋 雲兵, “Java による大域的並列計算環境 Ninflet”, 並列処理シンポジウム JSPP’98 論文集, pp. 135-142, 1998.

[2] Dominique Bénech, François Jocteur-Monrozier, Anne-Isabelle Riviére , Supervision of the CORBA Environment with SUMO: a WBEM/CIM-Based Management Framework, International Symposium on Distributed Objects and Applications, Antwerp, Belgium, 2000.

[3] IONA - CORBA Products -, http://www.iona.com/products/orbhome.htm

[4] CORBA サービス仕様, http://www.omgj.org/technology/documents/spec/services.html

[5] HORB Home Page, http://horb.etl.go.jp/horb/

[6] OMG’s CORBA Website, http://www.corba.org/

[7] オブジェクト指向研究会, 分散オブジェクトモデリングガイド, ピアソン・エデュケーション, 2001.

[8] Djamel Belaid, Nicolas Provenzano, Cantal Taconet, “Dynamic Management of CORBA Trader Federation”, 4th USENIX Conference on Object-Oriented Technologies and Systems, 1998.

[9] Top500 Supercomputer Sties, http://www.top500.org