第 6 部

プロセス間通信と名前サーバ

第 1 章

緒論

1.1 はじめに

最近のネットワーク技術の発達によって、多くの計算機同士が接続され、位置的に離れたユーザ間の情報交換や、資源の共有などが可能な計算機環境が構築されている [4, 3, 79, 44]。このような分散資源を有効に利用するためには、さまざまに分散した資源を特定することが必要であり、そのために名前付け、および位置付けを行なう方法を確立しなければならない。これらを実施するものとして、名前サーバが考えられている。

特に広域環境においては、管理母体の異なる組織がインタネットによって結合されているため、情報の集中管理は困難である。従って、分散管理された情報を基に位置付けを行なうような名前サーバが考えられる。

名前サーバの本来の機能は、計算機上の特定の資源に名前付けをし、その名前と実体を示す識別子の対応をとることにある。さらに、その資源に関する付加的情報をも管理する、インフォメーションサーバの機能を持たせることも試みられている [23]。

一方分散環境を考えた場合、より巨大で複雑な問題を解くための手法として、開放型システムが考えられる [20]。開放型システムでは、プログラムはその実行の各仮定において、その時点で利用できるサービスを最大限に利用しながら計算を進めることになる。この時、どのようなサービスがどの時点で可能であるかを前もって知ることはできない。したがってこのようなシステムでは、その時点で利用可能なサービスを知ること、および効率的処理のためにサービスを提供するサーバの中で最適なものを選択することが必要となる。さらに環境が広域になるにしたがって、ユーザが利用可能なサーバの数も増えてくるため、最適なサーバを選択することがより重要な意味を持ってくる。

つまり広域開放型分散環境においては、環境の中から利用可能なサーバの位置を知り、その中で最適なものを選択する手段が提供されるべきである。サーバの位置を知るということは、サービスの名前とそのサービスを

提供するサーバ群との対応づけを行なうことであり、そのために名前サーバを用いることができる。サーバの選択を行なうためには、最適であることを判断するための情報を得る必要がある。しかし分散環境であるために、得られる情報は常に過去のものであり、その情報を基に予測を行なわなければならない。このような選択は常にサーバの候補を得ることに続いて行なわれるので、これを名前サーバで一括して行なうことは合理的である。

本研究では、まず広域環境における名前サーバに必要な特性について考察を行なう。そしてその特性の中でも、特にサービスを提供するような計算機資源の取り扱いに注目し、複数の候補から最適なサーバを選択するアルゴリズムについて、待ち行列理論およびシミュレーションにより比較検討を行なう。その後、このような機能を含んだ、名前サーバの設計および実装を行なうことによって、効率良くサーバを利用することのできる分散アプリケーションの開発を支援する環境を提供する。

1.2 目的

本研究の目的は、広域開放型分散環境におけるサービスの利用にあたって、サーバの位置の取得および最適サーバの選択の方法を提案することである。

1.3 本論文の構成

本論文は 6 つの章から構成され、第 2章では、名前サーバの機能について、既存の名前サーバの概要と問題点を示すと共に、広域環境の特徴に関して述べる。第 3章では、サーバ選択アルゴリズムについて考察および評価を行ない、第 4章で選択機能を採り入れたインタネット名前サーバの設計を行ない、第 5章でその実装例を示す。そして第 6章で、本研究によって実装された名前サーバに関する考察および今後の課題をあげ、最後に第 7章で結論を述べる。

第 2 章

背景

本章では、まず名前サーバの意義について議論し、既存の名前サーバに関して概要を述べる。次に広域開放型分散環境について、その特徴と問題点を述べ、最後に本研究の目的について説明を行なう。

2.1 名前サーバ

名前サーバを特徴づける要因としては、その機能および扱うオブジェクトの種類、名前づけの方法、さらにインタネット環境への対応という点を考えることができる。

2.1.1 名前サーバの機能

名前サーバとは、ある資源の名前と、その位置に関する情報の対応づけをするものである。ここでは、この様に名前のついた資源をオブジェクトと呼ぶ。

計算機上には各種の資源があり、これらを参照することによって様々な処理を行う。計算機は通常これらをビット列として識別するが、これは人間が理解するのは困難なものである。そこで、理解しやすい文字列などを用いて名前をつけ、その名前と、計算機のための識別子の間の対応づけを行う。

このような機能を持つ名前サーバを構築するにあたって、実際に名前サーバの扱う対象、および名前付けの規則を明確にしておく必要がある。

2.1.2 名前サーバの扱うオブジェクト

名前サーバが管理すべき情報は、原則として全ての計算機資源である。主なものとしては、計算機そのもの、ユーザ、ファイル、プロセス等が考

えられる。これらに名前づけを行うことによって、参照や位置づけ、資源の共有といった操作を行うことが可能となる。

本論文では、名前サーバによって扱われるこれらの個々の資源をオブジェクトと呼ぶ。名前サーバの扱う名前には、単一のオブジェクトとグループリスト、および直接には対象を示さず、ある条件を満たす対象を意味する抽象的な名前というものが考えられる。

単一のオブジェクト

全ての計算機資源はオブジェクトであり、それを特定できる何らかの名前が必要である。従って、名前サーバはある名前が与えられた時に、それに相当するオブジェクトを示す識別子を返す機能を持たなければならない。

さらに、オブジェクトが与えられた時に、その名前が必要となることもある。従って、オブジェクトを示す識別子から、そのオブジェクトにユニークにバインドされるべき名前を返す機能も持っているべきである。

グループリスト

計算機上のあるオブジェクトを扱う場合に、個々のオブジェクトだけでなく、同種の複数オブジェクトなど、集合に対して処理を行なうことも多い。そのような場合に、それらを個々に参照するのではなく、グループに対して名前をつけて全体として参照を行なえることが望ましい。

さらに、グループに所属するメンバのリストそのものを名前サーバによって管理することによって、管理を一元化することができ、一貫性を保つのが容易になる。

抽象的な名前

計算機上の資源の中でも、「現在ログインしているユーザ」、「もっとも空いている計算機」など、あらかじめ定まっている特定の資源ではなく、適当な条件を満たす対象、という抽象的な概念によって対象を指定したい場合がある。このような場合には、従来、グループリストを名前サーバから得て、クライアント側で要求を満たすオブジェクトを選別する方法がとられていた。

この方法では、各種の要求に柔軟に対応できる反面、同様の機構がアプリケーション毎に必要となるという欠点がある。そこで、良く使用される選択条件は、あらかじめ名前サーバ側で用意しておけば、クライアントは名前によって要求するだけで、要求を満たすオブジェクトを見つけることができるようになる。

2.1.3　名前付け規則

環境の中で名前によってオブジェクトを示すためには、その名前の付け方が問題となる。一般に名前付けの方法には、絶対型、相対型、階層型の 3 種類が考えられる。

絶対型ネーミング

全てのオブジェクトを対象として、ユニークな名前をつける。例えばシリアルナンバをつけるなどがこの方法であり、どの立場からでも同一の名前によって識別することができる (図 2.1)。

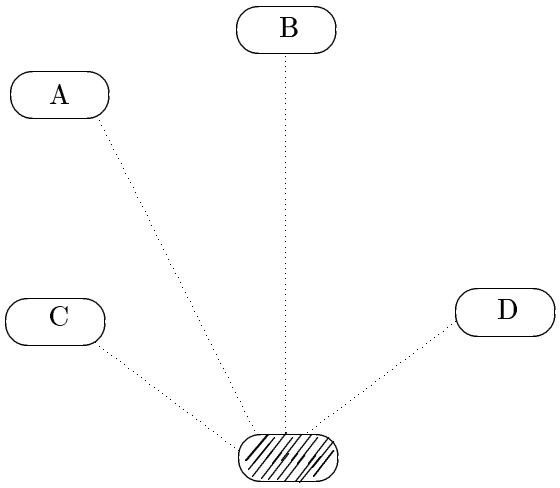


図 2.1: 絶対型ネーミングの例

オーソリティは 1 つで集中管理され、名前空間はフラットなものとなる。

相対型ネーミング

オブジェクトを指定するために、クライアントからそのオブジェクトへの関係で示す方法である。これは例えば uucp メイルシステムなどで用いられている方法であり、特定のオブジェクトの名前がクライアント毎に存在することになる (図 2.2)。

この方式では、そのクライアントにとって、ある名前がユニークに特定のオブジェクトを示せれば良いので、名前全体の管理は分散管理するこ

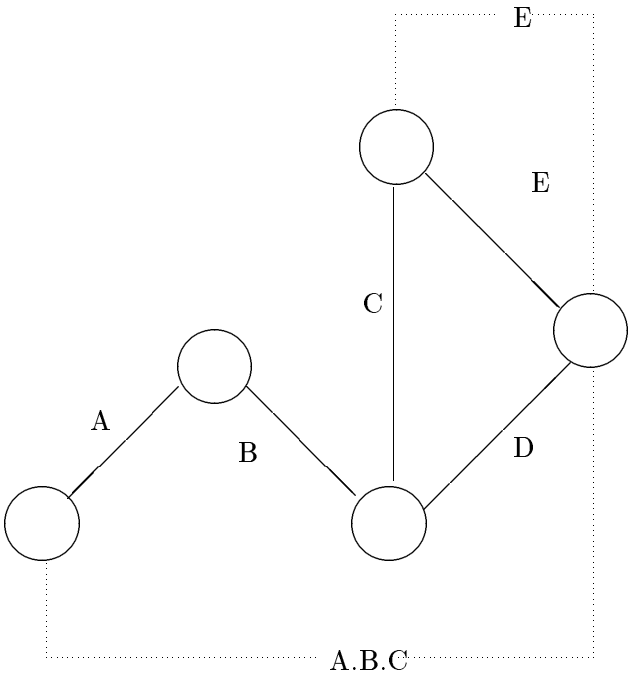


図 2.2: 相対型ネーミングの例

とができるが、クライアントを動作させる環境毎に、名前が変化することになる。

階層型ネーミング

名前そのものに階層構造を持たせたもので、あるレベルでは、そのすぐ下に位置する名前のユニーク性のみを保証する方法である。UNIX のファイルシステムなどにも用いられている方法であり、どの位置からでも同一のオブジェクトを指せる絶対型の指定と、クライアントの位置によって示すオブジェクトが変化する相対型の指定のどちらも可能である (図 2.3)。

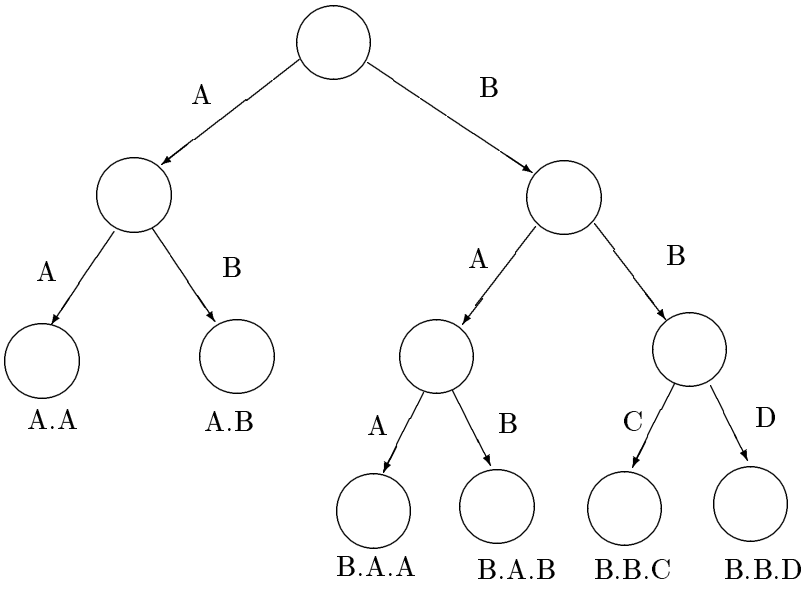


図 2.3: 階層型ネーミングの例

階層型のネーミングは、さらに固定階層と可変階層の 2 種類に分類される。情報は各階層毎に管理すれば良いので、分散管理される。

2.1.4　インタネットにおける問題点

名前サーバをインタネット環境で使用するためには、名前付けの問題以外にも、信頼性、一貫性、インタフェース、セキュリティなど、いくつか考慮すべき問題がある。

信頼性

名前サーバは、一般に分散環境における通信機能を支える、基本的な機能であるため、システムの信頼性、データの信頼性が重要な意味を持つ。

名前サービスが、あるホストのダウン、あるいは通信のエラーによって機能できなくなるようなことのないように、何らかの対応をとる必要がある。さらに、情報そのものに関しては、その情報の新しさ、情報源の信頼性などで判断することができるので、名前サーバは、このような信頼性を明らかにしながらクライアントにデータを提供すべきであると考えられる。

一貫性

名前サーバは、データベースを持ち、そのデータをクライアントに提供する、いわばデータベースシステムであるということができる。さらに、データに対する要求が広範囲から発生する環境であり、効率及び信頼性を保つために、分散管理が行なわれている。従って、分散データベースとして共有データの一貫性を保つ機能が要求される。

分散データベースとして名前サーバをとらえた時には、次のような機能が必要となる。

- 全ての名前サーバの情報の更新、削除を行ない、名前サーバ同士の情報の一貫性を保つ。
- 名前割り当ての際に、正当性を保ち、オーソリティが分散している場合には登録の衝突を回避する。

一般に一貫性を保証するためには、順序制御を行うことが必要十分条件と考えられている [17]。しかしこのような一貫性の保証を問題にする場合、名前サーバは確かにデータベースシステムではあるが、以下の点で、汎用データベースとは性格が異なる。

1. 名前サーバが機能しないと、システムの動作に支障をきたす
   名前サーバの管理するデータは、個々のユーザと同時に、システム内の各種の基本アプリケーションなどによっても参照されることが多い。従って、汎用データベースに比較して、即答性が第 1 に求められる。
2. リードとライトの同期制御の必要性が低い
   通常のデータベース管理では、リードとライト、ライトとライトの同期制御を必要とする。しかし、名前サーバの場合は、状況の変化を告げるライト要求が来た時点で、それ以前のデータは無効となるため、

先に来ていたリード要求のためにライト要求を延期しても無効データを読ませるだけになってしまう。

以上のことから、一般に名前サーバでは最終的なデータの一貫性がとれていれば、ある期間データに矛盾が生じることを避けることよりも、即答性を向上させることが重要であることがわかる。

インタフェース

名前サーバを参照する方法としては、反復的に行なう方法と再帰的に行なう方法が考えられる。

1. **反復的な方法**
   要求を受けた名前サーバが、求めるデータを持っていない場合、そのデータに関する情報を持つと考えられる、他の名前サーバのアドレスをクライアントに返す方法である。クライアントは、その結果にしたがって、他の名前サーバに問い合わせを行なう。

2. **再帰的な方法**
   要求を受けた名前サーバが、求めるデータを持っていない場合、他の名前サーバを検索し、データを提供する。

対話的な方法では、ユーザインタフェースが複雑になるという欠点がある。一方、再帰的な方法を用いると、名前サーバ自身がデータを検索しなければならないため、名前サーバにかかる負荷が増大するという問題があげられる。

セキュリティ

インタネット名前サーバは、多くのコミュニティ間で情報交換を行なうが、名前サーバの持つ情報の中には、ある対象に対して知られたくないという性質のものも含まれることになる。従って、名前サーバは保持する情報のアクセスコントロールを行なって、権利のないアクセスや変更からデータを守る機能が必要となってくる。さらに、ユーザ確認のためのパスワードや通信データなどを暗号化する機能も必要である。

2.2　既存の名前サーバ

現在、分散環境における名前サーバについての研究としては、以下のものがあげられる。

1. 論理的な階層名前空間を分散名前サーバによって管理し、メッセージ配達機能を持つ、Xerox の Grapevine[62]
2. Grapevine と同様の思想の基に、名前サーバ部分を独立、充実させた、Xerox の Clearinghouse[52]
3. DARPA インタネット・ネーミングコンベンションにしたがって、カリフォルニア大学バークレイ校で開発された BIND(Berkeley Internet Name Domain) サーバ [?]

ここでは、それぞれの名前付け規則、基本機能、管理するデータ、構成、その他を簡単にまとめ、問題となる点について述べる。

2.2.1 Grapevine

Grapevine は、インタネット上でのメッセージ配達を主な目的とし、さらに資源の位置づけ、オーセンティケーション、アクセスコントロールを行う、Xerox の Research Internet 上の複数の計算機による分散システムである。

このシステムは、メッセージ配達をするメッセージサーバと、データベース管理を行うレジストレーションサーバによって成り立っている。このうちメッセージサーバは、ある指定された 2 点間のメッセージ転送を行なうもので、メッセージが確実に相手に到着するか、届かない場合にはその理由とともに送り返すことを保証するものである。レジストレーションサーバは、データベースを管理し、名前とオブジェクトの対応づけを行なうものであり、名前サーバの中心的な機能を担当する。ここではレジストレーションサーバを中心に述べる。

Grapevine は、以下の事柄を設計目標としてあげている。

1. ひとたび、Grapevine がクライアントからメッセージ配達の要求を受け付けたら、そのメッセージが受け取り先に確実に到着するか、そうでなければ、原因を示すものと共に必ず送り返される、ということをクライアントに対して保証すること。
2. Grapevine サーバの存在する計算機の 1 つがダウンしても、クライアントに対するサービスに支障をきたさないこと。

名前付け規則

インタネット全体は論理的にいくつかに区分けされ、それを registry と呼ぶ。オブジェクトの名前としては、registry 名とその registry 内でのオブジェクト名からなる、2 階層の名前が採用されている。

identifier.registry_name

したがって、データベースも registry 単位で分散管理されている。

管理するデータ

individual と、group の 2 つのタイプのエントリがある。グループタイプのエントリには、アクセスコントロールリスト、デリバリリストなどがあり、他のエントリである名前の集合である。

Indevidual のエントリは、次のものを含んでいる。

- Authenticator: password
- index sites の list
- connect site: インタネットアドレス
- アクセスコントロール関係の情報: friends リスト、owners リスト

また、それぞれの要素は、タイムスタンプを含んでいる。

構成

このシステムは、複数の Grapevine 専用計算機によって実現されており、メッセージサーバと、レジストレーションサーバの 2 つの部分に分かれている (図 2.4)。

registry 内のデータは、必ず複数のレジストレーションサーバによって複製が保持されており、どのレジストレーションサーバによって管理されているデータベースに対しても変更を行うことができる。

クライアントは、各マシン上で動いている Grapevine ユーザパッケージと呼ばれる、インタフェースプログラムを通して、レジストレーションサーバと更新することができる。パッケージとレジストレーションサーバ、またレジストレーションサーバ間は、どちらもインタネットプロトコルにしたがって交信する。

GV という名前の registry があり、これはそれぞれのレジストレーションデータベースの分散と複製の情報を管理する registry である。この registry 内の名前は総て “∗.gv” という形であり、これは、registry 自身につけられている名前である。 “reg.gv” は、“reg” という registry を含んでいるレジストレーションサーバをメンバとするグループの名前であり、同様に、“gv.gv” は、全てのレジストレーションサーバをメンバとするグループ名である。この様にサーバ自身をも名前をつけて管理する。

この GV registry は、全ての registry 上にその複製が保持されているため、クライアントは、ある情報を得るために適切なレジストレーションサーバの位置を、どのレジストレーションサーバからでも得ることができる。

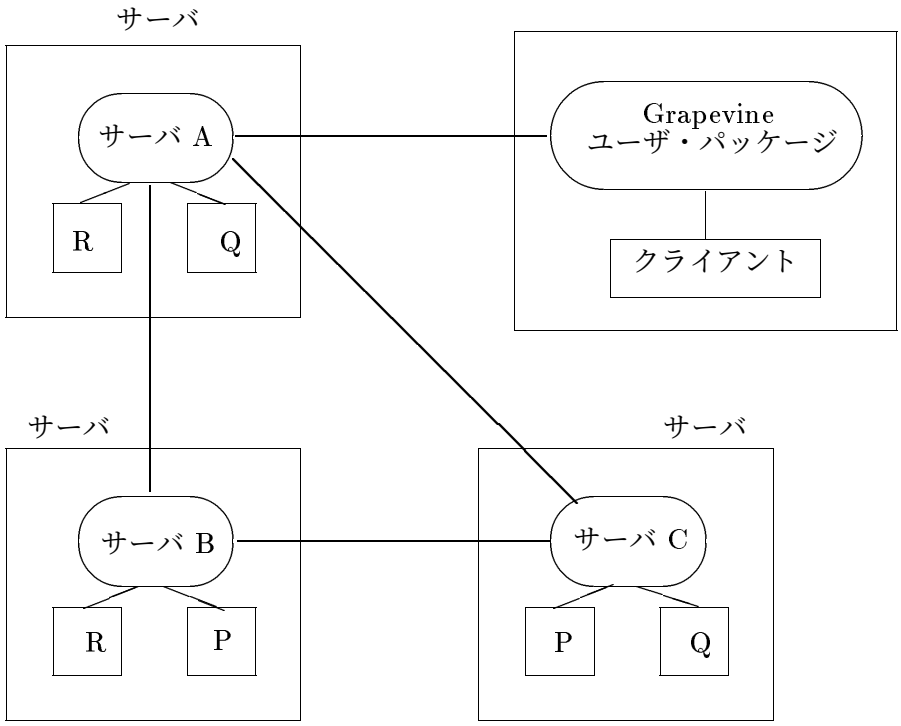


図 2.4: Grapevine のシステム構成

基本的機能

メッセージサーバ

Accept message:	{sender, passwrod, recipients, message-body} → ok
Message Polling:	{individual} → {empty, noempty}
Retrieve message:	{name, password} → sequence of messages → ok

レジストレーションサーバ

Authenticate:	{individual, password} → {authentic, bogus}
Membership:	{name, group} → {in, out}
Resource Location:	{group} → members
	{individual} → connect site
	{individual} → ordered list of inbox sites
Registration DB Update:	

セキュリティ

Owners list と Friends list からなるアクセスコントロールリストによって、エントリごとに指定されている。

Owners list に名前のあるものは、グループメンバ、オーナ、フレンドの各リストへ、加入及び削除を行うことができる。また、Friends list に名前のあるものは、グループメンバに対して、加入、削除を行うことができる。

即ち、そのエントリの内容を変更できるのが Friend であり、さらにそのエントリに対するアクセスコントロールの指定も変更できるのが Owner である。

検索アルゴリズム

例えば、クライアントが “f.r” というエントリについての情報を得るには、ユーザパッケージを使って、どこかのレジストレーションサーバと通信をして、

1. “r.gv” のメンバリストを調べる。
2. そのリスト内のどれかのレジストレーションサーバのアドレスを調べる。
3. そのサーバと通信を試みる。
   (a) コネクションが張れた場合、“f.r” の情報を要求し、情報、あるいは “name not found” を得る。
   (b) コネクションが張れなかった場合、2. に戻る。

これが基本アルゴリズムであるが、実際には次のように行われている。

1. 初めに通信したレジストレーションサーバに、“f.r” の情報を要求する。
   (a) そのサーバが “r.gv” のメンバであって、情報が存在した場合
   → 要求した情報が返される。

(b) そのサーバが “r.gv” のメンバであって、情報が存在しなかった場合
→ name not found が返される。

(c) そのサーバが “r.gv” のメンバでなかった場合
→ wrong server が返される。
この場合には、基本アルゴリズムにしたがってやり直す。

ユーザパッケージはレジストレーションサーバとの通信を、次のような方法で始める。

ローカルネットワーク上に name lookup server があり、それは、インタネット上にあるいくつかのレジストレーションサーバの名前とそのアドレスをマッピングする機能を持つ。ユーザパッケージはその name lookup server とブロードキャストプロトコルによって通信することができる。これによって、ユーザパッケージは特殊なパケットを送り、アクセス可能なレジストレーションサーバと通信を始めることができる。また、name lookup server がダウンした場合も、そのユーザパッケージが直接つながっているローカルネットワーク上にあるレジストレーションサーバとの通信ができるように、ユーザパッケージがローカルネットワークに特殊なパケットをブロードキャストすると、アクセス可能なレジストレーションサーバが返事を返すようになっている。

データ変更アルゴリズム — タイムスタンプ方式

各エントリには、そのエントリのバージョンナンバとしてのタイムスタンプが含まれていて、最後にエントリの内容に変更のあった時刻を示している。また、各アイテムにも個々にタイムスタンプがつけられていて、そのアイテムの変更時間を示している。このタイムスタンプには、変更要求を受け付けた時間と、受け付けたサーバのインタネットアドレスが含まれている。

データ変更の要求を受けたレジストレーションサーバは、

1. ローカルのデータベースを更新する。

2. 更新されたエントリ全体を含む変更メッセージを、メッセージサーバによって、関係するレジストレーションサーバ全て、すなわち “*.gv” リストのメンバに送る。

変更メッセージを受け取ったレジストレーションサーバは、送られてきたエントリとサーバが持つエントリを比較して、マージ操作を行う。つま

り、2 つのバージョンのエントリの全てのアイテムを、どちらか新しいタイムスタンプを持つものに変更し、新しいバージョンのエントリを作る。

グループタイプのエントリには 3 つのリストがあるが、それぞれが Active list と Deleted list を持ち、それによって、異なる順序で変更メッセージが到着した場合にも一貫性を保つようになっている。つまり、操作のログを持っていることに等しい。

変更操作の行われている比較的短期間の一貫性は、この方法によって保たれるが、変更メッセージが破壊された場合の一貫性の保証問題が残る。そのために日に一度等、定期的に全てのデータベースを比較し、マージ操作を行う。さらに、データ登録そのものの衝突については、システムの外部で、人間によって集中化が行われるように提案しており、システム自体での対処はなされていない。

キャッシュ

各メッセージサーバは、受取人とその inbox の対応づけを行う、inbox サイトキャッシュと、アクセス不可能なサーバのリストである、ダウンサーバリストを持っている。

inbox の変更によって無効となったキャッシュの情報は、間違えて送られたメッセージサーバから返される、cache flush notification message によって、修正される。

問題点

この Grapevine には、次に述べる Clearinghouse と同様の問題点を持つ。そのため、問題点に関しては次節にまとめる。

2.2.2　Clearinghouse

Clearinghouse は、単にメッセージ配達をサポートするための住所録を目標としたものではなく、インタネット上に分散されたオブジェクトの名前づけと位置づけをサポートするための分散システムである。

名前付け規則

Clearinghouse では、全体を論理的に区分けし、それぞれをオーガニゼーションと呼ぶ。さらにオーガニゼーション内にもいくつかの論理的な区分けがなされており、それらをドメインと呼ぶ。これらの区分けには、物理的なネットワークとの直接の関連はない。

Clearinghouse の管理するオブジェクトは、そのオブジェクトの所属するオーガニゼーションとドメインを用いて、3 階層のネーミングで名前づけされている。

identifier @ domain_name @ organization_name

また、別名をつけることができるが、これらはドメイン単位で管理されている。

管理されるデータ

Clearinghouse は様々な種類のオブジェクトを管理するが、これらは単位 (individual) と集合 (group) に大別できる。individual としては、計算機や、ファイルサーバ、プリンタサーバ、メイルサーバ、クリアリングハウスサーバなどの各種サーバ、ユーザなどがある。group はそれらの集合であり、アクセスコントロールリスト、ディストリビューションリストなどがある。

データ構造としては、1 つのエントリについて、いくつもの属性を持つことができる。データタイプは individual と group の 2 つで、0 と 1 で表される。データタイプが 1、すなわち group タイプの場合、属性値は名前のリストとなる。また、属性値自身がエントリ名の 1 つであっても良い。

名前 → (⟨ 族姓名, 属性タイプ, 属性値, ⟩, ⟨ , , ⟩, ...)

基本的な機能

Clearinghouse が提供する機能は、データベースとしての基本機能であり、登録、削除、変更、検索が行なえる。細部は 2 種類のデータタイプ、individual および group に関してそれぞれ以下のように定まっている。

1. **individual に関するもの**
   オブジェクトの位置付け
   オブジェクトの位置の登録、削除、変更
   オブジェクトの名前の登録、削除、変更
   属性名、属性値の登録、削除、変更
   名前の受渡し
   オブジェクトのタイプによる検索

2. **group に関するもの**
   グループメンバの取得
   グループのメンバシップのテスト

グループ名の登録、削除、変更
グループメンバの追加、削除
グループ名の受渡し
グループのタイプによる検索

構成

Clearinghouse は複数の Clearinghouse サーバによって構成されているが、クライアントに対しては 1 つのシステムとして振舞う。全体のデータはドメイン毎に分散管理され、あるドメインのデータはいくつかのサーバによって複製が保持されている (図 2.5)。

ドメイン内の全てのオブジェクトについての情報を持つものをドメイン Clearinghouse、オーガニゼーション内の全てのドメイン Clearinghouse 情報と、他のオーガニゼーション Clearinghouse の情報を持つものをオーガニゼーション Clearinghouse と呼ぶ。物理的には、全ての Clearinghouse サーバがオーガニゼーション Clearinghouse となっていて、Clearinghouse サーバは、いくつかのドメインのドメイン Clearinghouse であり、同時にオーガニゼーション Clearinghouse でもある。

Clearinghouse サーバ自身もオブジェクトであり、オーガニゼーション “O” 内のドメイン Clearinghouse は、次のような名前を持つ。

identifier@O@ClearinghouseServers

また、ドメイン “D@O” のドメイン Clearinghouse は、

D@O@ClearinghouseServers

という名前のグループのメンバとして登録される。

また、オーガニゼーション Clearinghouse についても同様であり、

identifier@ClearinghouseServers@ClearinghouseServers

という名前を持ち、

O@ClearinghouseServers@ClearinghouseServers

という名前のグループのメンバとして登録される。

クライアントは、各計算機上の “stub clearinghouse” と呼ばれるインタフェースを通して Clearinghouse と交信を行なう。stub は、少なくとも 1 つの Clearinghouse サーバのアドレスを持つ。さらにそのアドレスが無効になった場合に備えて、ブロードキャストなどの方法を用いて少なくとも 1 つの Clearinghouse サーバのアドレスを知ることができる。

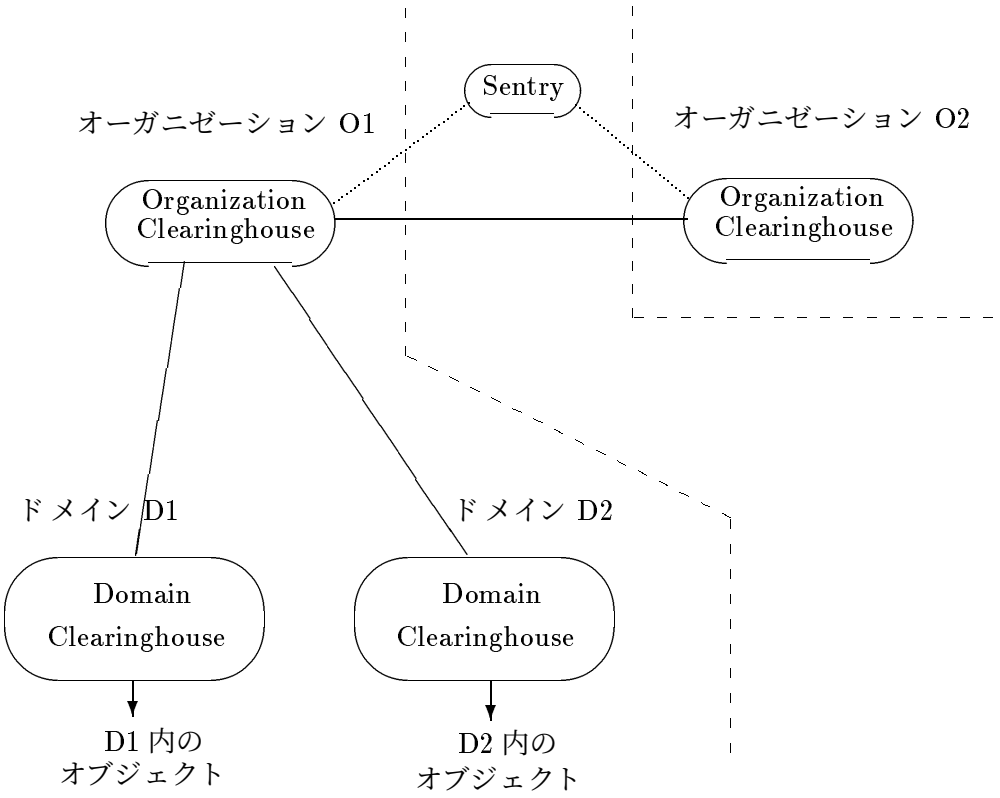


図 2.5: Clearinghouse のシステム構成

検索アルゴリズム

クライアントが “A@B@C” についての情報を要求する場合、

1. stub がいずれかの Clearinghouse サーバに連絡をとる。
2. そのサーバが “B@C” のドメイン Clearinghouse である場合
   (a) 求める属性の値を返す。
   (b) stub はクライアントに値を返す。
   (c) 終了
3. そのサーバが “B@C” 以外の “*@C” のドメイン Clearinghouse である場合
   (a) ドメイン “B@C” のドメイン Clearinghouse の名前とアドレスを探し、値を stub へ返す。
   (b) 5. へ
4. そのサーバが “*@C” 以外の “*@*” のドメイン Clearinghouse である場合
   (a) オーガニゼーション C のオーガニゼーション Clearinghouse の名前とアドレスを探し、値を stub へ返す。
   (b) stub はそれらのうちの 1 つと連絡をとる。
   (c) 2. へ戻る。
5. “A@B@C” を検索し、求める属性の値を返す。
6. stub はクライアントに値を返す。

データ更新アルゴリズム

基本的には、前節の Grapevine における更新アルゴリズムと同様であり、タイムスタンプとリクエストのログを残す。

Clearinghouse は、メッセージ配達の機能を持たないので、変更メッセージ伝達は、一般のメイルシステムを使用している。

1. stub は、変更したいエントリのあるドメイン Clearinghouse を探し、要求を出す。
2. ドメイン Clearinghouse は権利の確認を行ない、正しい要求者であった場合に受諾する。ここでクライアントは開放される。

3. ドメイン Clearinghouse はデータベースを変更し、タイムスタンプをつけて変更メッセージを全てのドメイン Clearinghouse サーバにメイルで送る。変更要求は、タイムスタンプと共に格納しておく。

4. メッセージを受け取った Clearinghouse サーバは、タイムスタンプを比較し各自のデータベースを正しくする。

メッセージが届かなかった場合に備えて、タイムスタンプと共に格納してある要求を使って、定期的に全てのドメイン Clearinghouse のデータベースを比較し、一貫性を保つ。

セキュリティ

相手の確認には、ログイン時と同じパスワードが使用される。エントリ毎にアクセスコントロールリストを持っている。リストはグループタイプのエントリであり、以下のような構造をしている。

$$\{\langle \text{set of names, set of operations} \rangle, \langle , \rangle, \ldots\}$$

また、アクセスコントロールリスト自体のアクセスコントロールを実現するために、Super System Administrator の名前とパスワードが、アクセスコントロールリストに含まれている。

さらに、互いに信用することのできないオーガニゼーション間で、ある程度の情報を交換したいという場合に対応するため、sentry を設けている。これは、両オーガニゼーションの中間に位置し、オーガニゼーションをまたぐ情報交換は、全てこの sentry を通して行なうものである。このことによって、サーバは sentry からの要求は、信用することのできないオーガニゼーションからのものであることを知り、相応の対処をすることができる。この方法では、信用できないオーガニゼーションには、本当の Clearinghouse サーバのアドレスの代わりに sentry のアドレスを知らせておくことになる。

問題点

Clearinghouse, Grapevine とも、論理的分割で名前を管理する点では非常に優れている。また、データ構造にも柔軟性があり、全てのオブジェクトを同じ方法で名前管理ができることも優れた点といえる。

しかし、どちらもデータベースの更新アルゴリズムに問題がある。ここで行なわれている方法では、定期的にシステムを停止し、その間に膨大なファイル転送、比較という作業が必要となる。システムを停止させるのは

避けるべきことであり、管理する全データの転送、比較という作業は、速度も面からも好ましい方法とはいえない。

また、Clearinghouse では、3 階層に固定された階層ネーミングを採用しているが、registry 間の関係を示しているオーガニゼーションが、さらに階層構造を持っていると考えた方が良い場合もあり得る。従ってそのレベルを固定階層にすることは、システムの構築に関しては利点ともなり得るが、柔軟性に欠けるという問題がある。もちろん、registry 間の関係を示す方法のない Grapevine では、この問題はより顕著に現われる。

2.2.3　BIND

BIND(Berkeley Internet Name Domain) サーバは、分散された環境に存在するオブジェクト、リソースに名前付けする標準的な方法を提唱し、それらのオブジェクトについての情報を保存し検索するための機能を提供する。BIND サーバでは、管理上の分類を示すいくつかのドメインによって区切られた、階層構造を持つ名前空間を、集団で管理する [39, 40]。

オブジェクトを、その物理的な位置と独立に参照できるようにすることによって、よりトランスペアレントな分散環境を提供することを目的としている。

名前付け規則

名前空間は木構造で構成され、それぞれのノードはラベルを持っている。各ノードのラベルは 63 文字以内の文字列である。ドメインは、ルートノードから、そのドメインまでのラベルを右から左へ “.” で連結したものである。名前の最後は “.” で終る。ルートノードは、長さ 0 のヌル文字列 “” で表される。

各ドメインはその下にサブドメインを持つことができ、サブドメイン名は、そのドメインの中でのみユニークである。また、階層の深さは無制限であり、どのサブドメインもさらにその下にサブドメインを持つことができる。

構成

以下のコンポーネントで構成されている。

- **リゾルバ (resolver)**
  ユーザインタフェースとして、各アプリケーションにリンクされる。ユーザから見ると、システム全体は単独の関数あるいは OS の 1 つの

機能と見なすことができる。名前空間は単一の木構造なので、ユーザは何処からでも必要な情報を得ることができる。

また、リゾルバにとっては、名前サーバは不特定多数存在するが、その情報は本質的には変化しないものと考えることができる。

- **名前サーバ**
  ドメインの木構造に関する情報を保持する。名前サーバにとっては、システム全体はゾーンと呼ばれるローカル情報の集合と見なすことができる。名前サーバは定期的に、オーソリティの情報をコピーし、またリゾルバからの参照に対して、並列に検索及び応答を行なう。

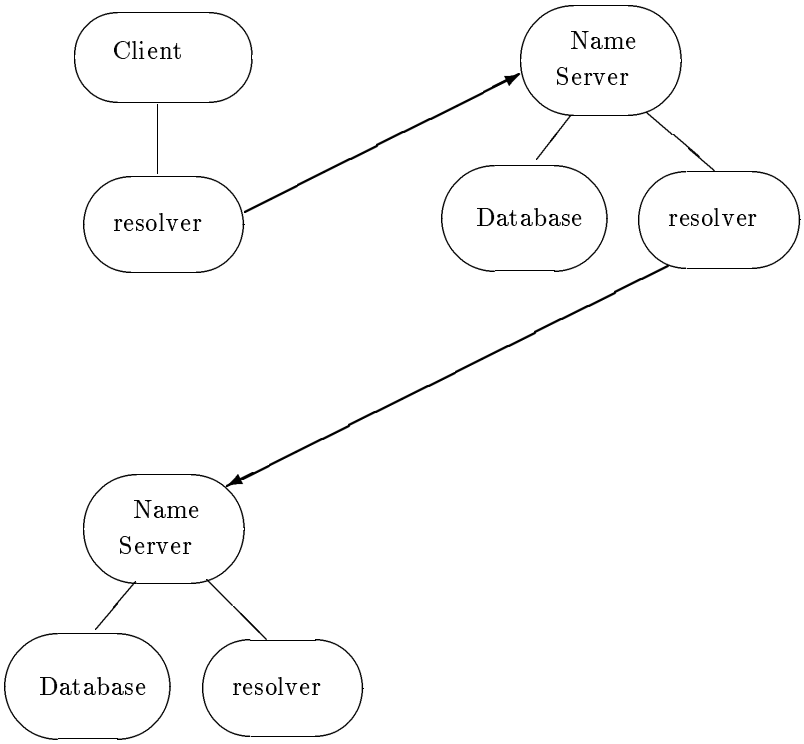


図 2.6: BIND サーバのシステム構成

BIND サーバの使い方には、設定によって、マスタ、キャッシング、リモート、スレーブなどの種類が存在する。

1. **マスタサーバ**
   1 つのドメインに対して、オーソリティとなる持つサーバが最低 1 つ存在する。このサーバはそのドメイン内の全ての情報を管理する。通常、各ドメインは 2 以上のサーバを持ち、そのうちの 1 つがプライマリ・マスタサーバ、残りがセカンダリ・マスタサーバとなる。

   - **プライマリ・マスタサーバ**
     各ドメインにただ 1 つだけ存在し、管理データはディスク上のファイルから読み込む。自身の代理として動作すべき、セカンダリ・マスタサーバを指定することができる。
   - **セカンダリ・マスタサーバ**
     プライマリ・マスタサーバと同様に振舞うが、ブート時にはプライマリ・マスタサーバから情報を得、定期的に一貫性のチェックを行なう。

2. **キャッシング・サーバ**
   どのドメインのオーソリティでもなく、キャッシュのみを行なうサーバである。

3. **リモートサーバ**
   ある計算機上にサーバをおかずに、他の計算機上のサーバを直接参照することができる。この場合には、その計算機上にはクライアントのみ存在すれば良いが、キャッシュを持たないので、全ての参照をネットワークを通じて行なう。

4. **スレーブサーバ**
   自分自身で解決できない要求を、あらかじめ定められた特定の名前サーバへと転送する。他のドメインと接続できないホスト上のサーバに対して設定する。また、特定のサーバにキャッシュを集中化させるために用いられることもある。マスタサーバが同時にスレーブサーバとなることもできる。

管理するデータ

BIND サーバは、集団で 1 つのグローバルデータベースを管理している。管理するデータは、ホストアドレス、メイルボックス、サーバなどで、これらのデータは、resource records(RR) に格納される。RR は、ドメインネーム、クラス、タイプ、データ、データの有効期間などの各フィールドで構成される (表 2.1)。

表 2.1: BIND サーバの管理するデータ

種類	レコード名	内容
TYPE	A	ホストアドレス
	CNAME	別名
	HINFO	計算機の種類および OS
	MX	メイルの送り先
	NS	名前サーバ
	PTR	他の名前空間へのポインタ
	SOA	ゾーンのオーソリティに関する情報
CLASS	IN	インタネットシステム
	CH	カオスシステム
TTL		リソースレコードの有効時間

検索アルゴリズム

名前サーバは、要求を発生するリゾルバの所属するドメインから検索をはじめる。

1. 再帰的な検索を行なう場合は 5. へ。
2. 自分の保持するデータベースから検索の対象となる名前の所属する、すぐ上のドメインを探す。見つからない場合は 4. へ。
3. ラベル同士の比較を行なう。
   (a) ラベルが一致した場合、検索タイプが CNAME なら 6. へ。そうでなければ、置換を行なって 1. へ。
   (b) 一致しなかった場合で、自分がオーソリティでなければ、オーソリティの名前サーバのアドレスを返答領域にコピーして 4. へ。
   (c) “*” との照合が許されるタイプの場合は、一致するものがなく、かつそのゾーンに “*” がない時には、CNAME を展開した結果であれば単に exit し、そうでなければエラーを返す。“*” が含まれていた場合は、そのエントリを返答領域にコピーして 6. へ。
4. キャッシュを検索して見つかった場合は、その内容をオーソリティの名前サーバのアドレスと共に、返答領域にコピーする。検索の結果にかかわらず、6. へ。

5. リゾルバルーチンを用いて、名前サーバへの問い合わせを行なう。各問い合わせは非再帰の指定をするが、返答の内容によって次の名前サーバへの問い合わせを行ない、最終的に得られた結果を返す。

6. 自分自身のデータベースのみの中から、有用な情報を返答領域に追加して終了する。

データ更新アルゴリズム

情報の変更は、各ゾーンのプライマリ・マスタサーバでのみ行なわれる。セカンダリサーバのデータの更新は、各ゾーンの SOA のリソースレコードの内容を基に行なわれる。

セカンダリサーバは、プライマリサーバから情報をもらうと、一定の間隔毎に、プライマリサーバとの間でシリアル番号を比較し、新しくなっていればそのゾーンの内容全てを更新する。また、比較に失敗すると一定間隔毎に、再試行し、ある期間以上プライマリサーバへのアクセスができない状態が続くと、自分が持っているコピーを破棄する。

比較の間隔、再試行の間隔、破棄するまでの期間は、各ゾーン毎に、それぞれ独立に設定することができる。また、セカンダリサーバ自身にも、プライマリサーバと同様の転送機能を付加することによって、プライマリサーバがダウンしている状態でも、他のセカンダリサーバからデータを受け取って処理を行なうことも可能となっている。

データ管理

プライマリサーバにおける、オーソライズドデータは、ディスク上にファイルの形で保管され、名前サーバは立ちあげ時にこのファイルの内容をメモリ中に読み込んで動作する。

セカンダリサーバは、自分がマスタサーバとなっているゾーンの情報を、プライマリサーバから受け取り、メモリ中に保持すると共にファイルに書き出す。立ちあげ時には、ファイル中の情報を読み込んだ後すぐに、プライマリサーバとの間でシリアル番号の比較を行なう。

いずれの場合もキャッシュはメモリ中にのみ保持される。

インタフェース (resolver)

BIND サーバとの間の通信や、データの転送などの作業を行なうのがリゾルバである。リゾルバは設定ファイルから、最低 1 つの名前サーバのアドレスを知る。

現在リゾルバは、サブルーチンの集合として提供されている。

セキュリティ

アクセスコントロールの機能は提供されていない。

問題点

BIND においては、オブジェクトのグループとして、メイルボックスの静的グループ以外はサポートされていない。しかし、計算機の名前やアドレスなど、名前サーバの扱うべき他の種類のオブジェクトについても、グループ化は必要である。

さらに、動的な情報の更新ができないため、静的に定まっている情報以外を管理することができない。

2.3 広域開放型分散環境

複数の計算機がネットワークで結合された分散環境では、アプリケーションを構築する際に、既存のサーバを利用することが考えられる [96]。このように、その時点で利用できる資源を活用して処理を行なうような環境を開放型分散環境と呼ぶ。さらにこの分散の範囲を、組織間など広域に広げたものが広域開放型分散環境である。

2.3.1 リクエスト・リプライプロトコル

分散環境におけるアプリケーションの構築方法として、最も一般的なものが、リクエスト・リプライプロトコルである。

リクエスト・リプライプロトコルとは、ある処理の単位に関して、要求を出す側と、実際に処理を行ない応答を返す側にわけて考えるものである。要求を出す側をクライアント、処理を行なう側をサーバと呼ぶ。また、サーバが提供する処理をサービスと呼ぶ。

2.3.2 複数サーバの存在

環境が広域化してくるに連れ、利用可能な環境の中に、同じサービスを提供するサーバが複数存在するような状況が生じてくる。このような場合に、どのサーバに処理を依頼するかという問題は、そのサーバの利用状況などによって、結果を得るまでの時間に影響を与える。

一般に、複数のサーバから処理を依頼するサーバを選択する方法としては、静的に指定する方法とマルチキャストによる方法がある。

- **静的に指定する方法**
  もっとも多くの場合に用いられている方法であり、プログラム中、あるいはファイル中でサーバの存在するホストを指定する。

  この方式の欠点は、サーバの存在するホストのダウンに対応できないこと、特定のサーバに負荷が集中する可能性があることなどである。

- **マルチキャストによる検索**
  複数のサーバに要求を出し、もっとも早く答えたものを利用する方法がある。この方法は YP[68] 等で用いられている。

  欠点としては、マルチキャストによってネットワークの混雑すること、結果的に不要となる要求によって、各サーバの処理量が増えることなどがあげられる。

2.3.3　通信の距離

分散環境においてリクエスト・リプライプロトコルを用いる場合には、サービスの処理時間はサーバの計算時間だけではなく、ネットワークの状況にも影響されると考えられる。すなわち、通信にかかる時間が大きければ全体としての処理時間も増大する。

また、このように通信にかかる時間を考慮に入れると、クライアントとサーバの間で情報を共有することが困難になる。例えば、負荷の重いサーバを避けるなど、クライアントがサーバの状態によって処理を変更する場合を考える。この場合、実際に問題となるのは、サーバが処理を行なう時のサーバの負荷であるが、これは、選択の時点から考えると未来のことであり、知ることはできない。さらに選択の時点でクライアントが知り得るのは、情報の伝達遅延を考慮に入れると、既に過去の情報である。

このような伝達時間を通信の距離と呼ぶ。

2.3.4　広域分散環境

組織間、あるいは国際的なネットワークを接続した場合のインタネットなどの広域環境を考えた場合、その特徴としては以下の事柄があげられる。

- **バンド 幅が狭い**
  一般に広域の環境では、バンド 幅当たりの回線のコストが大きくなるために、ローカルエリアネットワークと比較して、低速のリンクが使われることが多い。そのために、転送するデータ量が増えると通信にかかる時間が非常に大きくなる可能性がある。

- **遅延時間が大きい**
  通信を行なう場合、データ量とは無関係に、電気信号や伝播が伝搬するために一定の時間が必要であるが、特に国際回線などで衛星通信を行なっている場合には、この値がかなり大きくなる。

- **中継点が多い**
  広域環境は、point-to-point リンクの集合として実現されることが多く、通信の相手と直接接続されていることが少ないため、一般に通信相手との間に数多くの中継点が存在する。これらの中継点では、中継を行なうためのオーバヘッドが存在するために、データ量が少なくても一定の遅延が生じる。さらに、データ量が増えると、混雑が生じ、遅延が著しく増大したり、データの消失なども発生する。

広域分散環境ではこれらのことをふまえ、データ量や通信相手など、通信にかかるコストを減少させるようにする必要がある。従って、サービスそのものはもちろんのこと、サービスを提供するために用いられる情報収集などの通信についても、その通信量および頻度を最低限に押えなければならない。

2.4 本研究の目的

広域開放型分散環境においては、環境全体に分散して存在している各種のサーバの中から、適当なものを選択し処理を行なわせることが、重要な問題となっている。

本研究では、あるサービスを提供するサーバの中から、もっとも適当なものを選択して答えるような選択アルゴリズムを提案し、このような機能を持った名前サーバの設計と実装をおこなうことによって、広域開放型分散環境を支援する。

第 3 章

サーバの選択

本章では、まず開放型環境におけるサービスについてモデル化を行ない、同じサービスを提供する複数のサーバから、最適なものを選択するアルゴリズムについて考察する。

3.1 サービスの性質

本節では、広域開放型分散環境におけるサービスについて問題となる応答時間の要因を分類し、何を目的として選択を行なうべきかを明確にする。

3.1.1 サービスの応答時間

同じサービスを提供するサーバが複数あった場合、通常最も速く応答が得られるようなサーバを選択することが適当であると考えられる。また要求を出す際に、ある時間以内に応答が得られないのなら最初から要求を出すことなくエラーを返して欲しいという要求もあり得る。これらの要求を満たすためには応答を得るまでの時間を予測することが必要である。この時間はそのサービスの計算時間及び通信時間の和として与えられる。このうち計算時間は、サービスの計算量・サーバの性能・サーバの負荷に依存しており、通信時間はサービスの通信量・ネットワークのスループット・ネットワークの輻輳に依存している。

これらは以下の 3 つのタイプに分類することができる。

1. **サービスに固有の量**
   実際にどの程度の量の計算が必要であってそのためにどの程度の量のデータを通信しなければならないかは、サービスごとに特定できる。すなわちこれらはサーバの種類や状態とは無関係にクライアントの要求によって定まる量である。

2. **サーバによって定まる量**
   サーバの性能はサービスの種類との関係はなく、サーバが存在するホ

ストに固有の一定量である。またネットワークのスループットはクライアントとサーバの間の経路が定まればそれに応じて定まる一定量である。

3. **時間と共に変化する量**
   サーバの負荷やネットワークの輻輳は他のソフトウェアや通信に大きく影響を受け、時間と共に変化する可能性のある量である。

各サーバについてこれらの量をなんらかの方法で知ることができれば、それに基づいて要求に対して応答を得るまでの時間をそれぞれのサーバに対して予測することができる。その結果、最も適当と思われるサーバを選択することができるのである。つまり、問題はこれらの量をいかにして得るかということになる。

3.1.2 選択における問題

サーバの候補からの選択にあたっては、一般に最も短い時間で応答を得られるサーバが、クライアントにとって最適なものと考えられる。従って、名前サーバはサーバの計算時間、ネットワークの通信時間などから最短時間で応答を返すと予想されるサーバを選択することになる。

しかし、サーバの計算時間はクライアントによらない値であるが、クライアントとサーバの通信時間はクライアントによって変化する。これを予測するためには、クライアントとサーバの間のネットワークの構成などを知らなければならない。さらに経路が複数ある場合など、クライアントがどの経路を選択するかという情報を得なければならないが、これは名前サーバにとって、非常に困難である。

そこで、名前サーバに含む選択アルゴリズムでは、サーバの情報のみを元にして選択を行なう。通信量が多い場合には、ネットワーク情報を加味することが必須であるが、これはクライアントが選択することとし、名前サーバは候補となるサーバのリストのみを返す。

そのため、まず理論的な最適値を求めいくつかのアルゴリズムについて、この最適値との比較を行なう。その後、実際にネットワークによる通信時間を考えた場合の状況の変化について考察する。

3.2 待ち行列のモデル

クライアントにとって利用可能なサーバが複数存在する場合に、待ち行列のできる位置によって図 3.1および図 3.2のような 2 つのモデルが考えられる。

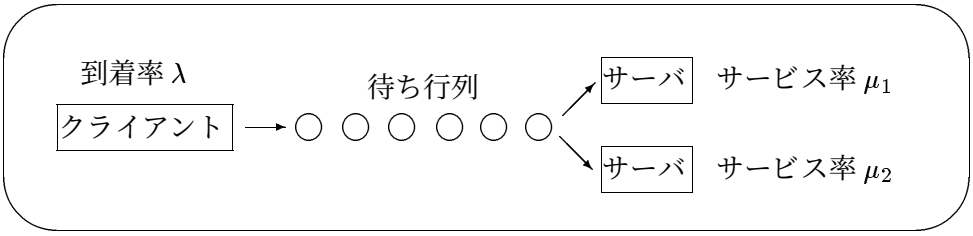


図 3.1: 全体として 1 つの待ち行列を作る場合

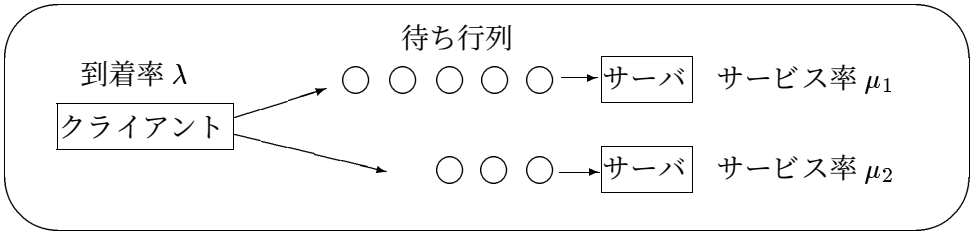


図 3.2: サーバごとに待ち行列を作る場合

図 3.1は要求をまとめて待ち行列にため、処理の終ったサーバに行列の先頭の要求を行なわせるものである。この場合には待ち行列が存在する限り、サーバがアイドルすることはないので平均待ち時間を最小とすることができる。ただし、この状態を実現するためには処理の終了を遅滞なく知る必要があり、待ち行列の位置と各サーバの距離が 0 とみなせる程度でなければならない。すなわちサーバ同士が近接している場合のモデルである。

これに対して図 3.2は要求が発生した時点でいずれかのサーバにその要求を依頼してしまい、サーバ側で待ち行列を作る場合である。これはサーバが分散している状態であり、広域分散環境における一般的なモデルである。この場合は要求が発生した時点で選択を行なう必要があるため、選択の方法が平均待ち時間に大きな影響を与える。

3.3　待ち行列理論による解析

実際の選択アルゴリズムの評価を行なうため、前節のそれぞれのモデルに対して待ち行列理論による解析を行ない、理想的な場合およびランダムに選択した場合の理論値を求める。要求の発生およびその処理はランダムであるとし、指数分布に従うものとする。

3.3.1 理想的な場合

理想的な場合とはサーバが近接していると考えられる状態であり、図3.1に示されたモデルが適用できる。

条件としてサーバが 2 つの場合を考え、要求の到着率 (単位時間あたりの要求発生回数) を λ、サービス率 (単位時間あたりのサービス処理回数) をそれぞれ μ_1, μ_2 とする。

ここで、マルコフ連鎖を考えると、各状態は以下のように表される。

0 待ち行列の長さは 0、両サーバはアイドル

1 待ち行列の長さは 0、サーバ 1 がサービス中

2 待ち行列の長さは 0、サーバ 2 がサービス中

3 待ち行列の長さは 0、両サーバがサービス中

4 待ち行列の長さは 1、両サーバがサービス中

$\vdots$

k 待ち行列の長さは $(k-3)$、両サーバがサービス中 $(3 \leq k \leq N)$

ここである時間あたりの要求到着確率を T_λ、サービス終了確率を T_1 および T_2 として、$T_1 \geq T_2$ つまり $\mu_1 \geq \mu_2$ とし、待ち行列の長さは最大 $N-3$ とする。

次に状態遷移を考えると、

$$
\begin{bmatrix}
\overline{T_\lambda} & T_1 & T_2 & 0 & \dots & \dots & \dots & \dots & \dots & 0 \\
T_\lambda & \overline{T_{1\lambda}} & 0 & T_2 & 0 & \dots & \dots & \dots & \dots & 0 \\
0 & 0 & \overline{T_{2\lambda}} & T_1 & 0 & \dots & \dots & \dots & \dots & 0 \\
\vdots & T_\lambda & T_\lambda & \overline{T_{12\lambda}} & T_{12} & 0 & \dots & \dots & \dots & 0 \\
\vdots & 0 & 0 & T_\lambda & \overline{T_{12\lambda}} & T_{12} & 0 & \dots & \dots & 0 \\
\vdots & \vdots & \vdots & \vdots & T_\lambda & \overline{T_{12\lambda}} & T_{12} & 0 & \dots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \overline{T_{12\lambda}} & T_{12} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & T_\lambda & T_{12}
\end{bmatrix}
\begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ P_N \end{bmatrix}
=
\begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ P_N \end{bmatrix}
\tag{3.1}
$$

ただし、

$$
\begin{aligned}
T_{12} &= T_1 + T_2 \\
T_{1\lambda} &= T_1 + T_\lambda \\
T_{2\lambda} &= T_2 + T_\lambda \\
T_{12\lambda} &= T_1 + T_2 + T_\lambda
\end{aligned}
$$

と表される。ここで (3.1) 式と、

$$\sum_{i=0}^{N} P_i = 1$$

を連立させて遷移確率方程式を解くことにより、各状態の存在確率が求められる。この結果とリトルの公式により、平均待ち時間 W は、

$$W = \frac{1}{\lambda}(P_1 + P_2 + \sum_{i=3}^{N}(i-1)P_i) \tag{3.2}$$

と求まる。

3.3.2　ランダムに選択した場合

まず、サーバが 1 つであった場合について考える。ここで、要求の到着率 (単位時間あたりの要求発生回数) を λ、サービス率 (単位時間あたりのサービス処理回数) を μ とする。

待ち行列理論 [93] により、待ち時間分布 (待ち時間が t 以内の確率)$W(t)$ は一般に、定常状態で待ち行列の長さが n である確率 p_n、及び、サービス時間が t 以上となる確率 $S(t)$、時間 t の間に少なくとも n 回の処理を行なう確率 $W_n(t)$ を用いて、

$$W(t) = p_0(1 - S(t)) + \sum_{n=1}^{\infty} p_n W_{n+1}(t) \tag{3.3}$$

となることが示されている。さらに $W_n(t)$ と、t 以内にちょうど n 回の要求に答えられる確率 $S_n(t)$ の間には、

$$\begin{aligned} W_n(t) &= 1 - \sum_{k=0}^{n-1} S_k(t) \\ &= \sum_{k=n}^{\infty} S_k(t) \end{aligned} \tag{3.4}$$

という関係があるので、結局 (3.3) 式は、

$$W(t) = p_0(1 - S(t)) + \sum_{n=1}^{\infty} p_n \sum_{k=n+1}^{\infty} S(t) \tag{3.5}$$

となる。ここで、ランダムサービスでは、サービス時間分布は指数分布となり、また、サービス人数分布はポアソン分布となるため、

$$S(t) = e^{-\mu t} \tag{3.6}$$

$$S_k(t) = \frac{(\mu t)^k}{k!} e^{-\mu t} \qquad (k = 0, 1, 2, \ldots) \tag{3.7}$$

を用いて計算すると、

$$W(t) = 1 - e^{-(1-\rho)\mu t} \quad (3.8)$$

ただし $\rho = \lambda/\mu$

さらに待ち時間分布の確率密度 $w(t)$ は、

$$\begin{aligned} w(t) &= \frac{dW(t)}{dt} \\ &= (1-\rho)\mu e^{-(1-\rho)\mu t} \end{aligned} \quad (3.9)$$

で表される。従ってサービス時間を含む平均待ち時間 W は、

$$\begin{aligned} W &= \int_0^\infty t w(t) dt \\ &= \int_0^\infty t(1-\rho)\mu e^{-(1-\rho)\mu t} dt \\ &= \frac{1}{(1-\rho)\mu} \end{aligned} \quad (3.10)$$

である。

次にサーバが複数あった場合は、各サーバに対する待ち時間の加重平均をとれば良い。ここでサーバの数を n、要求の到着率を λ、サーバ i のサービス率を μ_i、サーバ i が選択される確率を P_i とすると、全サーバへの要求に対する平均待ち時間 W は、

$$\begin{aligned} W &= \sum_{i=1}^{n} P_i W_i \\ &= \sum_{i=1}^{n} \frac{P_i}{(1-\rho_i)\mu_i} \\ &= \sum_{i=1}^{n} \frac{P_i}{\mu_i - P_i\lambda} \end{aligned} \quad (3.11)$$

となる。

ここで、P_i を決定するために μ_i を用いることが考えられる。すなわち、

$$P_i = \frac{\mu_i}{\sum_{k=1}^{n} \mu_k} \quad (3.12)$$

とおくことによって、サービス率に応じた振り分けを行なうことができる。このような選択方法を加重ランダムと呼ぶ。

最適な振り分け

この結果を元に、サーバが 2 つの場合について、W が最小となるような P_i を求める。まず、(3.11) 式で $n = 2$ とおき、さらに $P_1 + P_2 = 1$ を用いると、

$$\begin{aligned} W &= \frac{P_1}{\mu_1 - P_1\lambda} + \frac{1 - P_1}{\mu_2 - (1 - P_1)\lambda} \\ &= \frac{2\lambda^2 P_1^2 + (\mu_2 - \mu_1 - 2\lambda)P_1 + \mu_1}{\lambda^2(-P_1^2 + (\mu_2 - \mu_1 - 1)P_1 + \mu_1\mu_2 - \mu_1)} \qquad (3.13) \\ \frac{dW}{dP_1} &= \frac{\mu_1}{(\mu_1 - P_1\lambda)} - \frac{\mu_2}{(\mu_2 - (1 - P_1)\lambda)^2} \qquad (3.14) \\ \frac{d^2W}{dP_1^2} &= \frac{2\mu_1\lambda}{(\mu_1 - P_1\lambda)^3} + \frac{2\mu_2\lambda}{(\mu_2 - (1 - P_1)\lambda)^2} \qquad (3.15) \end{aligned}$$

ここで、$\frac{dw}{dP_1} = 0$ を満たす P_1 を求めると、

$$P_1 = \begin{cases} \frac{1}{2} & (\mu_1 = \mu_2) \\ \frac{\mu_1}{\lambda} - \frac{(\mu_1+\mu_2)\mu_1}{(\mu_1-\mu_2)\lambda}(1 \pm \sqrt{\frac{\mu_2}{\mu_1}})(1 - \frac{\lambda}{\mu_1+\mu_2}) & (\mu_1 \neq \mu_2) \end{cases} \qquad (3.16)$$

となる。ここで、第 2 次導関数が正の条件で極小点の評価をすると、

$$P_1 = \begin{cases} \frac{1}{2} & (\mu_1 = \mu_2) \\ \frac{\mu_1}{\lambda} - \frac{(\mu_1+\mu_2)\mu_1}{(\mu_1-\mu_2)\lambda}(1 - \sqrt{\frac{\mu_2}{\mu_1}})(1 - \frac{\lambda}{\mu_1+\mu_2}) & (\mu_1 \neq \mu_2) \end{cases} \qquad (3.17)$$

このような確率で 2 つのサーバに対して要求を振り分ければ、(3.11) 式で表される待ち時間の平均は最小となる。このような選択方法を最適加重ランダムと呼ぶ。

3.3.3　2 種類のモデルの比較

理想的な場合および加重ランダム、最適加重ランダムによる選択を行なった場合の理論値を図 3.3 に示す。

実際に選択を行なう場合には、この 2 つの曲線の間の値となるべきである。理想的な場合を示した単一待ち行列の曲線に近付くほど待ち時間は少なくなる。逆にあるアルゴリズムが、ランダムに選択した場合よりも悪い結果を生むような状態では、ランダムな選択を行なうべきである。

3.4　実際の選択方法

実際に分散された環境では、待ち行列はサーバごとにでき、要求の発生と同時に選択を行なう必要がある。

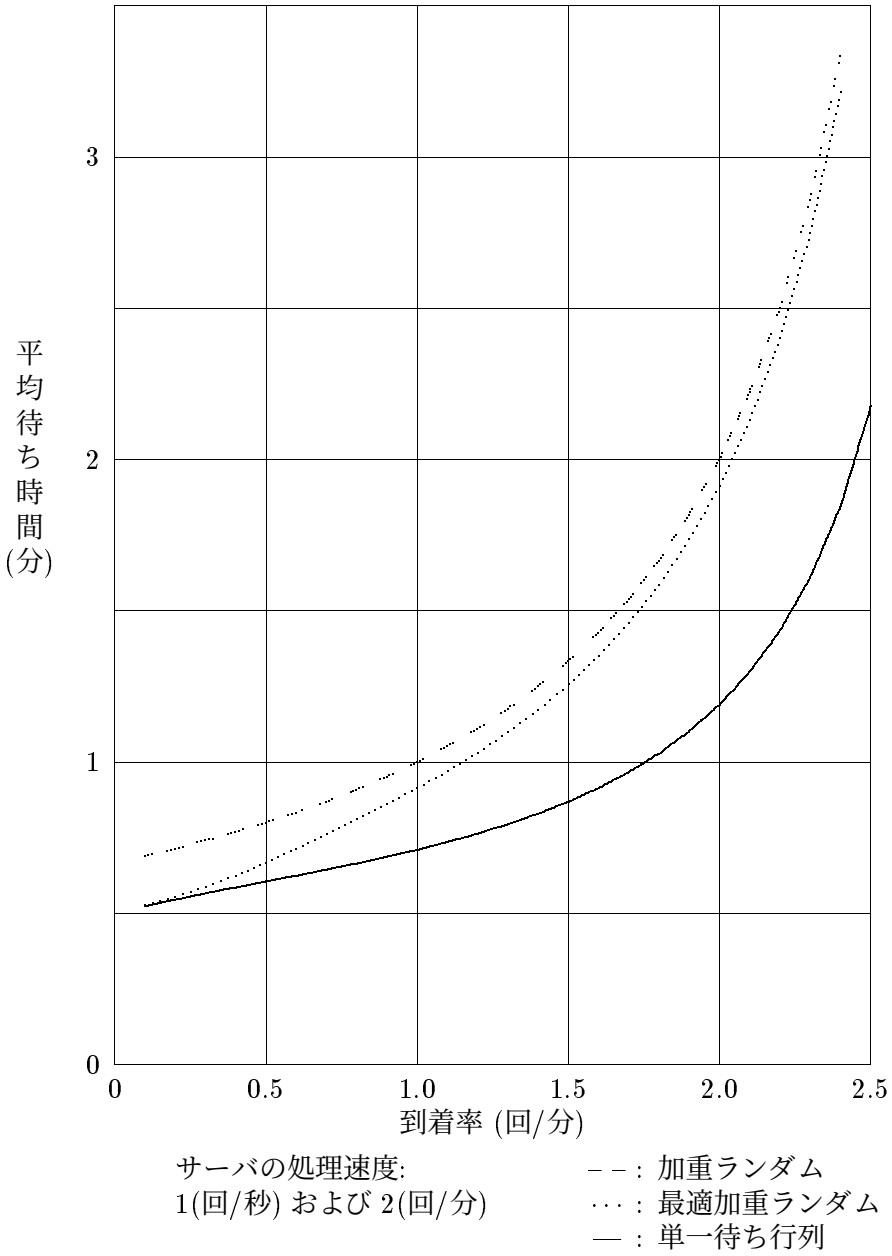


図 3.3: 理想的な場合およびランダムに選択した場合の理論値

3.4.1　基本的なアルゴリズム

複数の候補の中から、1 つを選び出す方法として、基本的なものには以下の 3 種類がある。

- **ランダム**
  複数のサーバのうち特定のものを、一定の確率で選択する。選択は純粋な確率事象であり、時刻、以前の状態などによらずに行なわれる。また、確率はサーバの性能を反映した重み付け確率とする。即ち、計算性能の高い計算機上のサーバほど、高い確率で選択される。

- **ラウンド ロビン**
  複数のサーバを順番に選択する。即ち、選択はそれ以前のサービスの履歴にしたがって、決定的に行なわれる。ただし、この際もサーバ毎の選択頻度は、そのサーバの位置する計算機の性能を反映した値とする。

- **最短待ち行列 (shortest queue)**
  その時点でサーバに到着し、処理されるのを待っている待ち行列の長さを調べ、もっとも短時間で処理の終りそうなサーバを選択する。つまり、待ち行列の長さにその処理自身を加えた値を、計算機の性能によって加重比較を行なう。

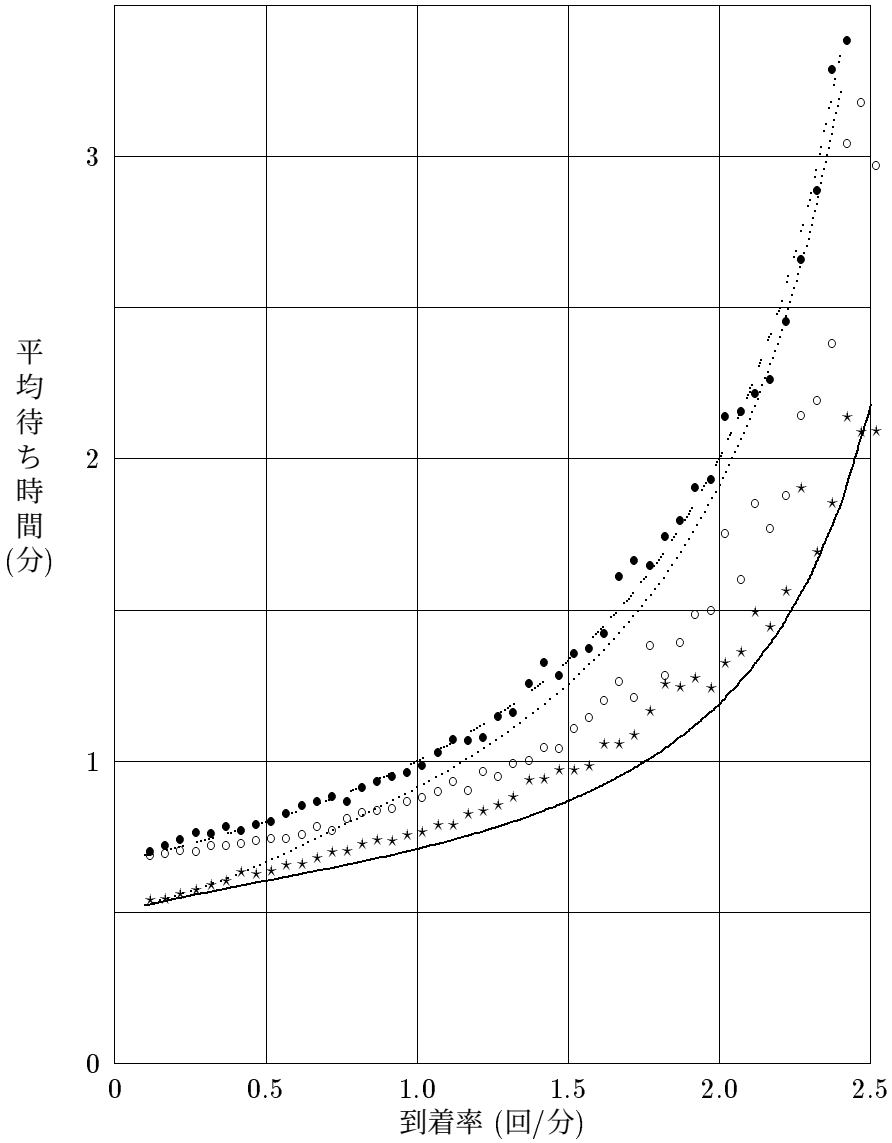
3.4.2　シミュレーションによる評価

シミュレーションにより第 3.4.1節で述べた 3 種類のアルゴリズムの評価、及び通信量の影響の評価を行なう。

ここではネットワークの通信による遅延時間がない状態で、2 つのサーバが存在すると仮定する。この 2 つのサーバは、あるサービスについて、平均処理時間がそれぞれ 1 分、30 秒であると仮定する。この場合、サービス率はそれぞれ、1(回/分), 2(回/分) となる。この場合のシミュレーション結果を図 3.4に示す。

図 3.4によると、要求の発生頻度が低い時には、最短待ち行列法では全ての要求を、速いサーバで処理するのに対し、ランダムあるいはラウンドロビンによって選択を行なう方法では、遅いサーバへも一定の割合で要求を送るため、平均待ち時間が増えている。

その点を除けば、ランダムに選択を行なった場合は理論値とほぼ等しく、ラウンド ロビンによる選択を行なうと若干待ち時間が少なくなるということがいえる。また最短待ち行列による方法は理想値とかなり近く、要



サーバの処理速度:
1(回/分) および 2(回/分)

● : ランダム
○ : ラウンドロビン
⋆ : 最短待ち行列
– – : 加重ランダム (理論値)
· · · : 最適加重ランダム (理論値)
— : 単一待ち行列 (理論値)

図 3.4: 通信時間が無視できる場合の比較

求の発生時点で待ち行列の長さとサービス率から待ち時間を予測し選択を行なうことによって、ほぼ最大の効率が得られることがわかる。

3.4.3　選択に必要な情報の収集

実際に分散されたネットワーク上では、名前サーバは、実際にサービスを処理するサーバの状態を予測して選択を行なわなければならない。選択に必要な情報としては、サービスを行なうサーバの位置、そのサーバの存在するホストの計算性能、サービスを行なう時点でのサーバの負荷などがある。

これらの情報のうち、サーバの位置及び、ホストの情報は、比較的静的に定まるものである。従って、これらの情報については、名前サーバ自身が管理を行ない、選択の際に利用することができる。

ところが、サーバの負荷の情報は時間と共に変化する量であり、これは名前サーバとの距離を考慮に入れなければならない。

距離による情報の遅れ

選択を行なう名前サーバと、情報源であるホストの間に距離が存在する時、その間の情報の伝達には一定の時間を要する。そのため選択の時点での情報は、サーバの位置するホストにとっては既に過去のものとなっている。

従って、最短待ち行列アルゴリズムを用いても、必ずしも空いているサーバを選択できるとは限らない。

情報更新の間隔

あるホストの負荷などの情報を収集する場合、その収集は常に行なわれているわけではなく、一定の間隔を持って行なわれる。そのため距離による情報の遅れに加えて、情報が更新されてからの時間による遅れが生じ、実際のキューの長さと異なる情報を得る可能性が高くなってしまう。

さらに、名前サーバが持つ情報がひとたび更新されると、次に更新されるまで同じ情報を持ち続けることになる。従って、この情報を基に選択を行なうと、次回の更新までの期間は状態に変化がないものと見なされてしまい、全て同じサーバを選択してしまう。つまり、情報の更新間隔と比較して、要求の発生間隔が短い場合には、負荷の集中が生じることとなり、待ち時間が増大する。距離による情報の遅れがないと仮定した場合の、情報更新の間隔による影響を図 3.5に示す。

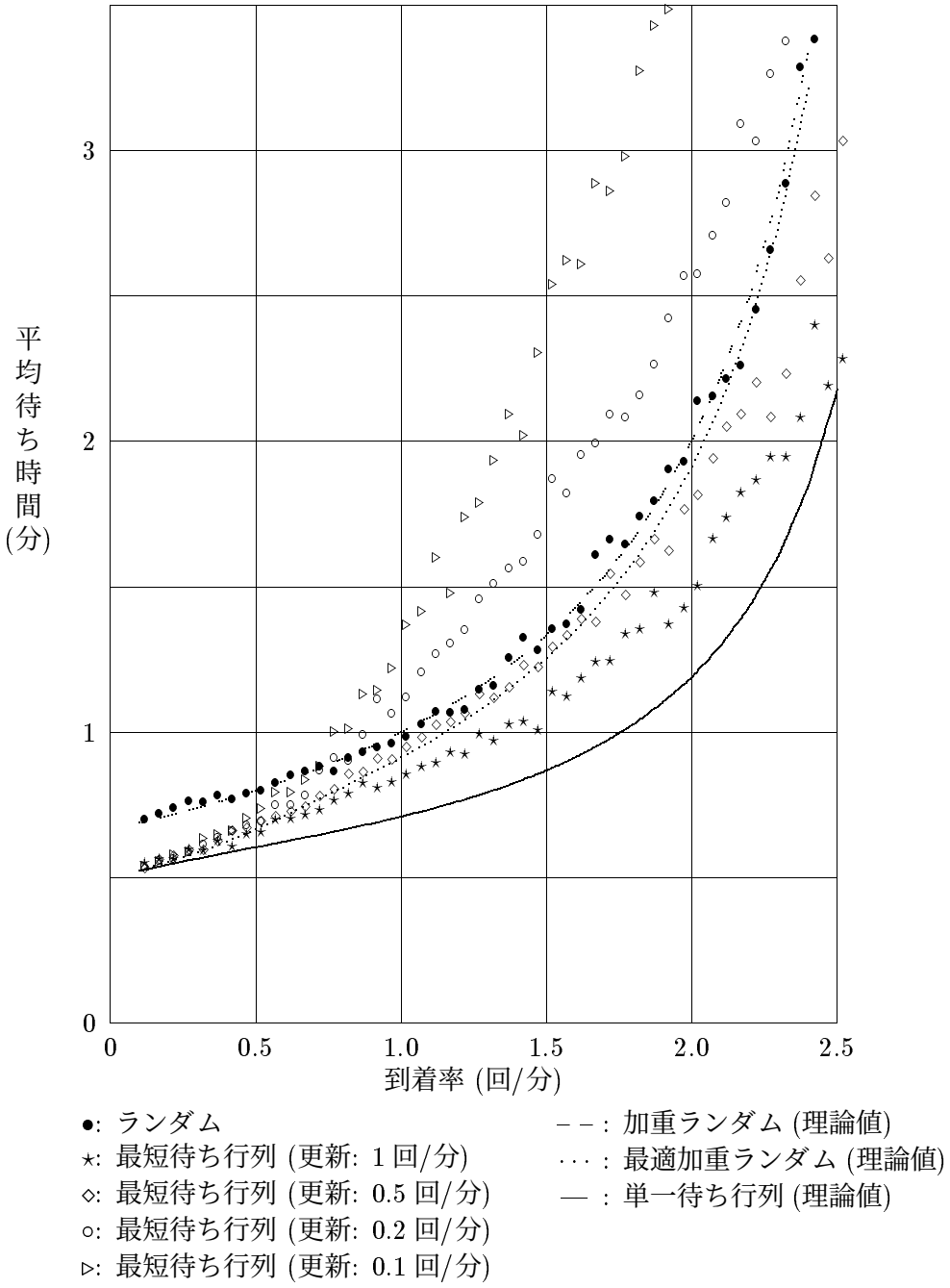


図 3.5: 情報更新の間隔による影響

情報の信頼性

前項で述べたように、収集された情報をそのまま使用することは、かえって負荷の集中を生じさせてしまう可能性があるため、得た情報の信頼性を考慮する必要がある。

情報更新の間隔による影響のシミュレーション結果である、図 3.5によれば、情報更新の間隔が要求到着の間隔に対して、約 4 倍程度になると、ランダムに選択を行なった方が短時間で応答を得ることができると考えられる。

そのためには、サーバからの情報として、待ち行列の長さとともに、要求の平均受け付け間隔を得れば良い。これらの情報によって、最短待ち行列の予測とランダムの 2 つのアルゴリズムを切替えて使用する。

さらにサーバ側でも、要求の平均受け付け間隔によって、情報の伝達間隔を変化させることが考えられる。すなわち、要求の発生頻度が高くなるに従って情報もきめ細かく送り、要求が稀にしか来ないような状況では、ある比較的長い間隔を空けて情報を伝達することにすれば、さまざまな状態において、常に最小のコストで選択を行なうことが可能となる。

3.4.4　ネットワークの通信時間

ここまで、ネットワークの通信時間が無視できる場合について考察してきたが、インタネット環境など特に低速なリンクを用いて大量の通信を行なわなければならない場合、この条件が満たされなくなる。このようなネットワークの例を図 3.6に、この条件の基でシミュレーションを行なった結果を図 3.7に示す。

このシミュレーションは、速いサーバに対するリンクが 64Kbps と特に遅く、逆に遅いサーバに対するリンクが 192Kbps と 3 倍の速度を持った場合について行なったため、最短待ち行列による選択によって遅いリンクの使用率が高まり、ランダムとほぼ同じ結果が出ている。また、得られる待ち行列情報の観測の時刻にもリンクの速度分だけの差が生じているため、最短待ち行列を選択することにも狂いが生じている。

3.5　最適な選択アルゴリズム

シミュレーション結果と理論値との比較により以下の結論が得られる。

- サービス要求の頻度が低く、サーバ側に待ち行列がほとんどできない場合には、最短待ち行列による選択は常に有効である。

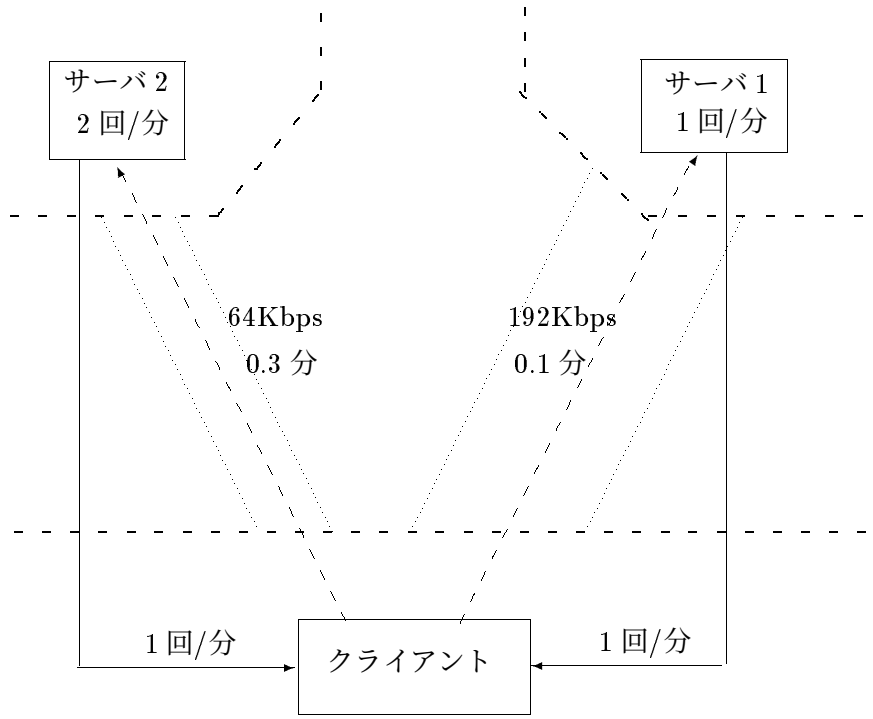
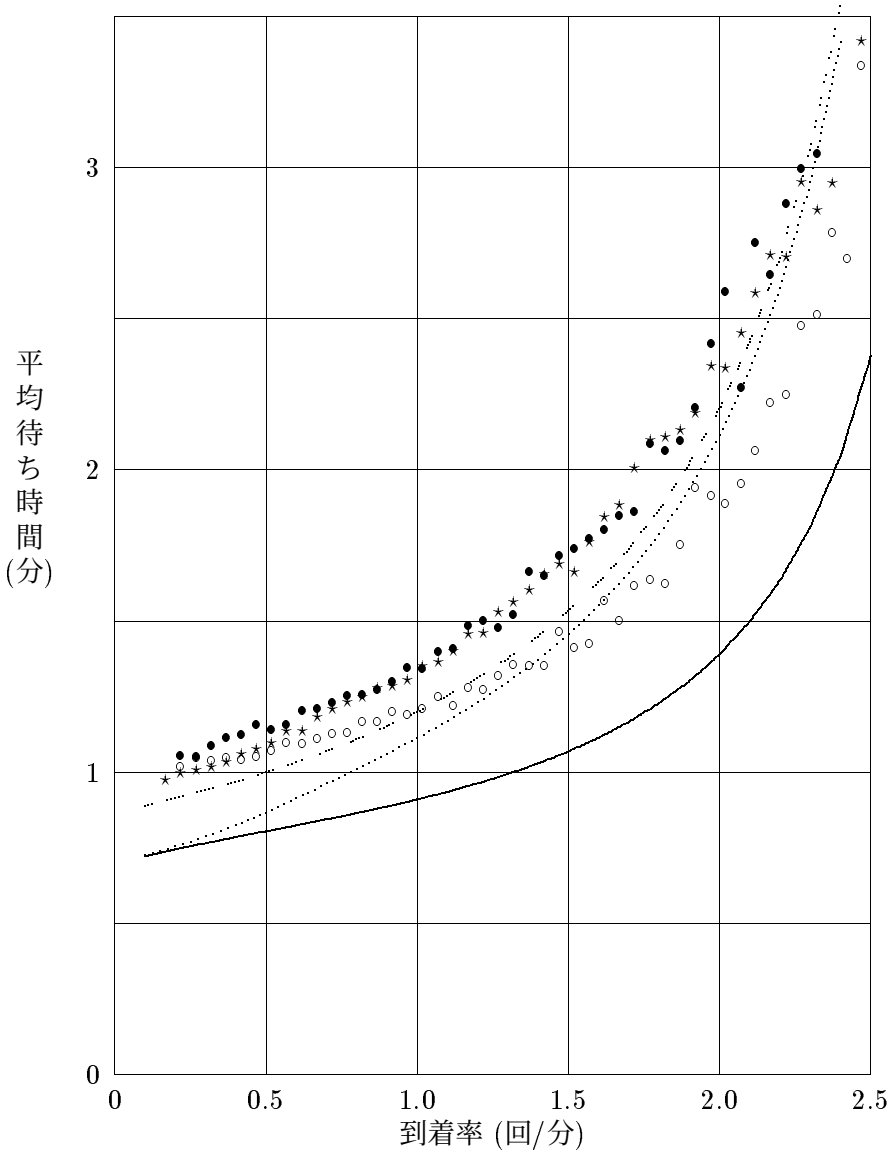


図 3.6: インタネット環境におけるサーバ選択の例



サーバの処理速度:
1(回/分) および 2(回/分)
通信時間:
0.3 分および 0.1 分
情報更新間隔:1 分

● : ランダム
○ : ラウンドロビン
⋆ : 最短待ち行列 (更新: 1 回/分)
– – : 加重ランダム (理論値)
· · · : 最適加重ランダム (理論値)
— : 単一待ち行列 (理論値)

図 3.7: 通信時間が無視できない場合

- サービス要求の頻度がある程度高い場合は、情報の更新間隔が平均待ち時間に大きく影響を与える。最短待ち行列の有効性を保つためには、サービスの受け付け間隔と同程度の間隔で情報の更新を行なう必要がある。情報更新の間隔がサービス間隔の 4 倍程度以上になると、要求の集中が起こるため、加重ランダムによる選択を行なった方が平均待ち時間は短くなる。

従って、サーバの存在するホストからの情報の更新は、原則的に変化のある都度行なうべきであることがわかる。しかし、変化の激しい時にあまり大量の情報を流すことは、ネットワークをより混雑させることになるため一定の制限を設けその制限間隔以内にはならないようにすべきである。また、これにともなって選択を行なう名前サーバ側でも情報の信頼性を評価し、4 回程度選択に使用した情報は抹消し、それ以降更新があるまでは加重ランダムによる選択を行なうという方法が有効である。

第 4 章

設計

本章では、論理的なネットワークをインタネットによって接続し、広域のネットワークコミュニティを形成する場合の、名前空間に対する基本概念と、それを実現する名前サーバの設計に関して述べる。

4.1 分散環境における名前サーバ

分散環境においては、他のホスト、あるいは他のネットワークなど、オーソリティの異なる環境全体を統一的に扱わなくてはならない。

そのためには、情報を集中させて管理することは困難であり、分散されたデータベースを仮想的に 1 つの体系に関係付けるような機構が必要である。

4.1.1 要求される機能

分散された環境において、各々のドメインがローカルに存在する資源の管理を行う。

分散環境においては資源全体の数は膨大なものとなり、これらを集中的に管理することは、非現実的である。さらに、ドメインの数が増えてくるにしたがって、環境全体における情報の変更の頻度は、相対的に高くなる。そのため、情報の変更があった場合、その情報を直接管理している部分の変更のみを行うことによって、その結果が環境全体に反映されるような構造が望ましい。

各種のオブジェクトに対して、統一的な名前表現によるアクセスの方法を提供する。

計算機上で扱われる資源には、ホストやユーザなどの他にも、プリンタやファイル、各種サーバなどのものがある。これらのものが、その対象によって異なる名前空間を持つことによる混乱は大きく、それを避けるためにも、統一的な名前表現が必要とされる。

1 対 1 のみならず、1 対多、多対 1、多対多などの対応づけを行い、抽象的な要求に対しても、適切な情報を答える。

ホストやユーザなど、単一の個体を指示する場合には、1 対 1、あるいは別名を含む多対 1 の対応が行われることが多い。しかし、管理者の集合、特定の種類の計算機などの、ある条件を満たす対象を検索したい場合には、これだけでは不十分である。このような要求を、ここでは抽象的な要求と呼ぶ。抽象的な要求としては、さらに、最も負荷の軽い計算機、最も速く応答の得られるサーバなどが考えられるが、こういった要求に答えるためには、複数の候補の中から最適なものを選択する必要もある。特に、あることを実行するサーバを知りたい場合には、最も早く応答が得られるものが望ましい。

4.1.2 考慮すべき事柄

- **分散環境への対応**
  原則としては、各ドメインのサーバは、直接そのドメイン自身に所属するオブジェクトのみの情報を管理し、サブドメイン内のオブジェクトの情報は持たない。しかし、必要に応じてサブドメインあるいは他のドメインのサーバへのポインタを持ち、検索の範囲をそれらの間に広げられるような機能を持たす。

- **統一的な名前表現**
  全てのオブジェクトは、

  オブジェクト名@ドメイン名

  という名前によって表現される。ドメイン名は不定長の階層構造を持つ。また、各オブジェクトは、正式な名前表現を 1 つ持ち、さらに別名を持つことができる。

- **抽象的な要求**
  複数の結果が得られるような要求に対しては、結果をリストとして返すことができるようにする。さらに、それらの候補の中から適当な

ものを選択するような機能をつける。特に、何等かのサービスを提供するサーバの場合、そのサーバの存在するホストに関する情報などから、最も短時間で応答の得られるサーバを選択する。

4.2　オブジェクトのグループ

名前サーバが扱うオブジェクトをグループ化して管理する場合、グループ化の方法として、静的にあらかじめ定まるものと、問い合わせを受けた時点で動的にリストを作成するものの 2 種類が考えられる。

4.2.1　静的なグループ

既に登録されている情報を、ある規則にしたがって分類したものであり、例えばメイルアドレスのグループや、特定の種類の計算機などがこれに相当する。BIND におけるグループは、メイルアドレスのグループのみであるが、名前サーバの扱うすべてのオブジェクトについてグループ化を行なう必要がある。

4.2.2　動的なグループ

現在ログインしているユーザ、負荷があるレベル以下のホストなど、ある条件を満たすオブジェクトの集合を示す。

4.2.3　サービス型オブジェクト

名前サーバが管理するオブジェクトのうち、特定のサービスを提供するサーバを、サービス型のオブジェクトと呼ぶ。

同様のサービスを提供するサーバが複数あった場合、その中から適当なものを選択する必要がある。この選択は、クライアントが行なう方法と、名前サーバが行なう方法の 2 通りが考えられる。

- **クライアントによる選択**
  名前サーバにかける負担が少なく、候補となるサーバのリストから、クライアント毎に適当な選択アルゴリズムを変更することができる。

- **名前サーバによる選択**
  選択のためのデータを転送する必要がなく、複数のクライアントに対して選択結果のキャッシュを利用することができる。クライアントが単純に構成できる。

それぞれの方法に利点・欠点があるため、クライアントによって任意の方法を選択することができるようにすることが望ましい。そのために名前サーバは、要求によって、候補となるサーバのリストあるいはその中から選択した最適サーバの、いずれでも返すことができるようにする。

4.3 構成

実装にあたっては、BIND サーバをベースとし、これに第 4章で述べたような拡張機能を付加する形で行なった。拡張された点としては、ホストおよびユーザ以外の、一般の計算機上の資源に関する記述を許すようにしたこと、各オブジェクトに対して、そのグループを定義し管理する機能を設けたこと、特にサービス型のオブジェクトに対し、最短時間で応答を返すサーバを予測し、その結果を返答する機能を付加したことなどがあげられる。

システム全体の構造を図 4.2に示す。ここで、サービスの選択は、名前サーバの機能の 1 つであるが、第 4章で述べた選択アルゴリズムの開発を容易に行なうために、実際には名前サーバとは別プロセスとして実装を行なった。

図 4.1において、システム全体は大きく以下の 4 つの要素から構成される。

- **名前サーバ**
  主に静的な情報の管理を行ない、リゾルバからの問い合わせに対して、適切な返答を行なう。また、自分自身が情報のオーソリティでない場合には、オーソリティを示す情報も付加する。

  さらに、情報が既知のものでなく、再帰的な検索が指定されている場合には、名前サーバ自身に含まれるリゾルバルーチンを用いて、他の名前サーバへの問い合わせを行なって情報を得る。

- **サービスマネージャ**
  各種のサービスを提供するサーバについて、その待ち行列の長さ、サービスの受け付け頻度などの動的な情報を管理し、最適サーバの選択を行なう。

- **リゾルバ**
  クライアントプログラムとのユーザインタフェースで、名前サーバへの問い合わせを行なう関数群である。

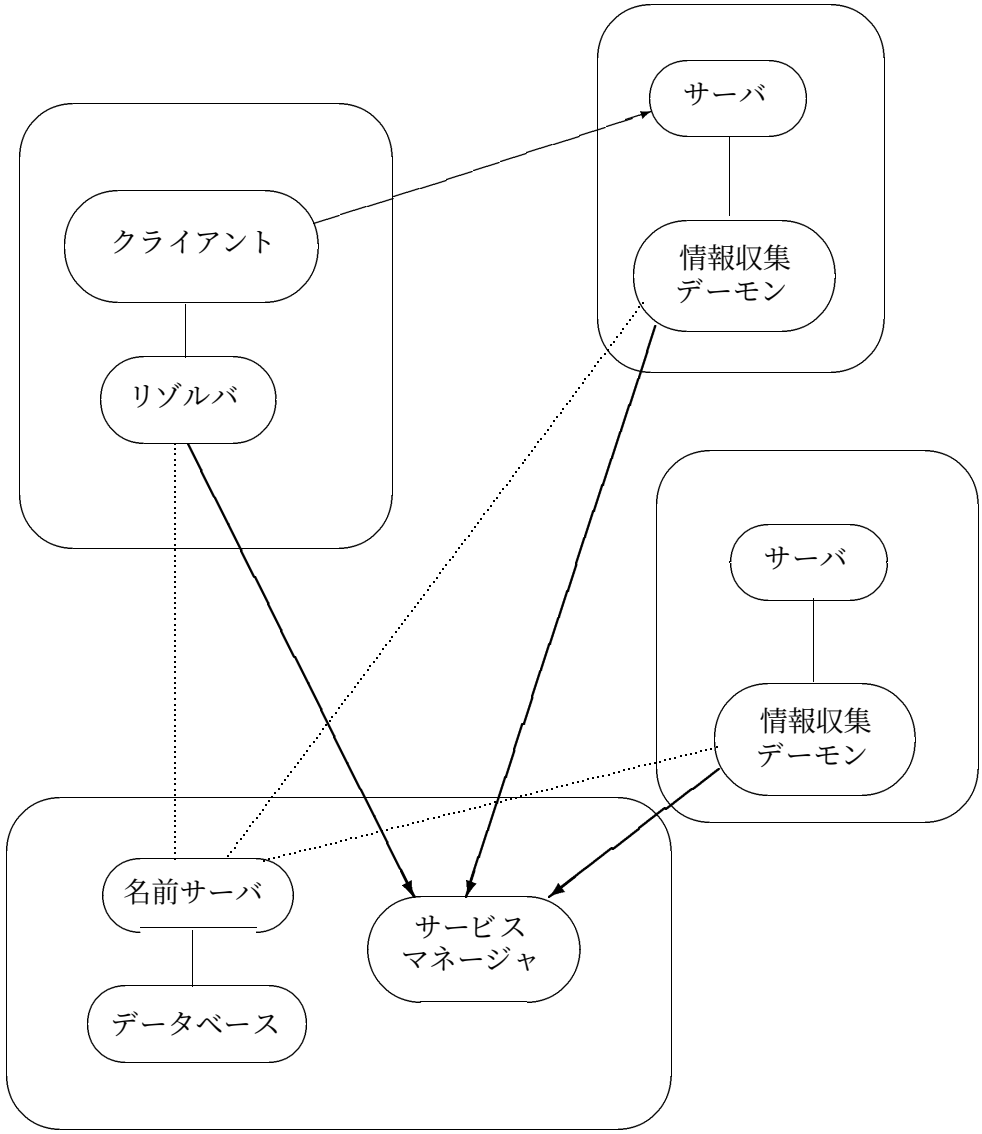


図 4.1: 名前サーバシステムの基本構成

- **情報収集デーモン**
  各ホストに 1 つ常駐し、そのホストの負荷情報、サーバ毎の処理頻度などの情報を、名前サーバを通じてサービスマネージャに伝達する。情報伝達の間隔は、処理頻度によって動的に調整を行なう。

この実装において、リゾルバは、まず名前サーバに選択マネージャのアドレスを問い合わせ、その後に選択マネージャに最適サーバの問い合わせを行なう、という機能を持つ。このようなリゾルバを、名前サーバ自身の一部として含めることにより、問い合わせを受けた名前サーバからサービスマネージャへの参照を行なう (図 4.2)。

4.4 データフォーマット

本節では、各構成要素の保持する情報を示す。

4.4.1 名前サーバ自身の保持する情報

名前サーバが直接管理するオブジェクトは、原則としてホスト名、ユーザ名、サービス名のみとした。それぞれのエントリは個々のオブジェクト、またはそのグループを示す。

1. **オブジェクト型のエントリ**
   名前ごとに複数の情報を格納する。情報の種類および数は、オブジェクト毎に定める。

   現在扱っているエントリを表 4.1に示す。

2. **グループ型のエントリ**
   グループ型のエントリの内容は、グループまたはオブジェクト名となる。グループ型エントリに直接オブジェクトの情報が格納されることはない。

   また、グループ中に、タイプの異なるオブジェクトを混在させることはできない。

4.4.2 サービスマネージャの保持する情報

サービスマネージャは、機能的には名前サーバの一部として動作し、何らかのサービスを提供するサーバに関する情報、およびそれに付随するホスト情報などを取り扱う。

現在、サービスマネージャによって扱われるエントリを表 4.2に示す。

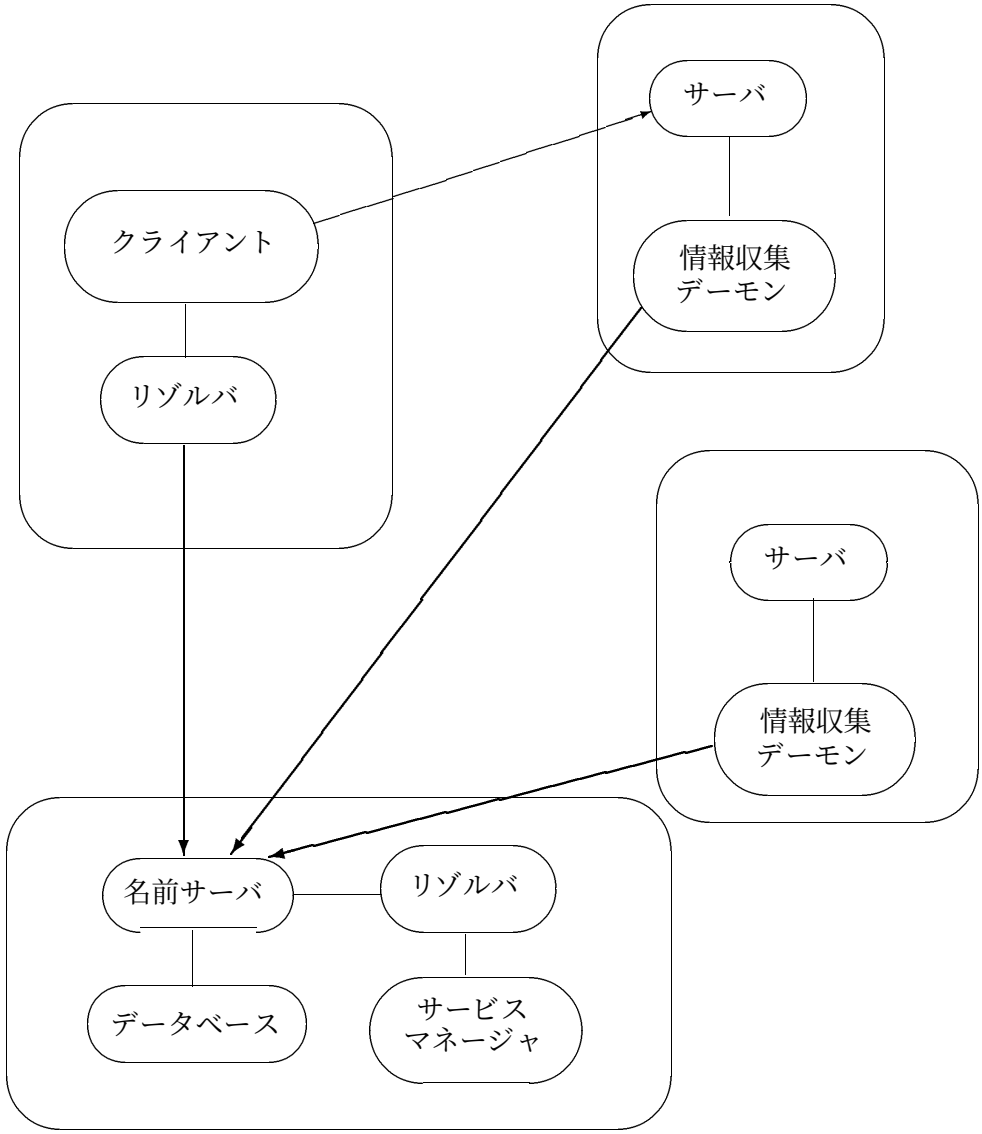


図 4.2: リゾルバ部分を組み込んだ名前サーバシステムの構成

表 4.1: オブジェクト型エントリに含まれる情報

タイプ	内容	格納形式
ユーザ	本名	文字列
	UID	整数
	電話番号	文字列
ホスト	IP アドレス	バイナリ形式
	機種	文字列
	計算速度 (MIPS)	実数
	負荷	(サービスマネージャのアドレス)
サービス	サーバのアドレスとポート	(サービスマネージャのアドレス)

表 4.2: サービスマネージャのエントリに含まれる情報

タイプ	形態	内容
サービス	グループ	サービスを提供するサーバのリスト
サーバ	オブジェクト	ホストおよびポート
ホスト	オブジェクト	負荷
		サーバの処理頻度

4.5　通信プロトコル

リゾルバと、名前サーバおよびサービスマネージャとの間の通信に用いられるメッセージフォーマットを図 4.3に示す。

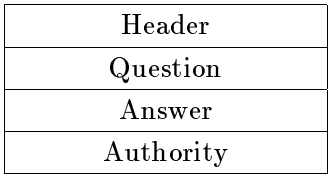


図 4.3: メッセージフォーマット

- **Header**
  すべてのメッセージに必ず存在し、メッセージの種類、エラー情報、ヘッダを除く各エントリの要素数などを含む。

- **Question**
  問い合わせを行なう名前、およびタイプが格納される。このエントリの数は、ヘッダ中に含まれ、その値は問い合わせメッセージでは 1 であり、返答メッセージでは 0 である。複数の問い合わせを同時に行なうことはできない。

- **Answer** このエントリは問い合わせに対する返答に利用され、その長さは任意である。エントリ内には、問い合わされた名前、タイプ、結果長、および結果が格納される。タイプがグループリストを示す場合には、返答の内容は特にグループリストの問い合わせを行なわない限り、各々のオブジェクトの値となる。

- **Authority** 名前サーバのオーソリティ以外が、キャッシュを参照して返答を返す場合に、オーソリティとなる名前サーバを格納する。フォーマットは Answer で用いたものと同じである。

第 5 章

名前サーバの実装

本章では、第 4章での設計に基づいた名前サーバの実装について述べる。

5.1 条件

Sun Workstation (SunOS 4.0) および Sony NEWS Workstation (NewsOS 3.3) 上でプロトタイプの実装を行なった。

選択を行なうサービスとしては、文書清書を行なう latex[34] を対象として実験を行なった。このサービスはデータ量によって処理時間が異なるが、サンプルのデータ量は約 50Kbyte から 200Kbyte 程度であり、処理時間は 100Kbyte のデータに対して計測を行ない、これを標準値として使用した。

5.2 各構成要素の動作

第 4章で述べた各構成要素の動作について説明する。

5.2.1 名前サーバ

名前サーバの部分は BIND (version 4.8.2) がベースになっている。サービスマネージャとは別プログラムであり、サービス型のオブジェクトに関する処理はすべてサービスマネージャが行なう。そのために、名前サーバにサービス名によってマネージャを問い合わせることができる。

5.2.2 サービスマネージャ

要求されたサービスについてリストを保持し、さらにサーバからの情報を利用してリスト中から最適なサーバを選択する機能を持つ。サービスリストの管理は、以下のようなファイルによって静的に行なう。

```
#
# service       hostname                        standard service time
#
tex             zushi.mt.cs.keio.ac.jp          40
tex             kogwy.cc.keio.ac.jp             100
tex             jp-gate.wide.ad.jp              50
```

5.2.3　リゾルバ

クライアントと名前サーバの間のユーザインタフェースであるが、サービスマネージャとのインタフェースも持つ。リゾルバは反復モードでは、以下のような手順で最適サーバを得る。

1. 名前サーバにサービスマネージャの位置を問い合わせる。
2. サービスマネージャから最適サーバの名前を得る。
3. 名前サーバに選択されたサーバの位置を問い合わせる。

このリゾルバを名前サーバ自身に組み込むことにより、名前サーバに対する一度の問い合わせで結果を得る再帰モードを実現することが可能である。

5.2.4　情報収集デーモン

サーバの存在するホストで動作し、定期的に負荷情報をサービスマネージャに送る。

サーバに対する処理は現実にはシリアライズされることは少なく、一般に並列に処理される。そのため実際の待ち行列はサーバごとに発生するわけではなく、CPU に対する各サーバおよび他のプロセスの待ち行列として生じる。今回の実装環境である UNIX では、これを load average と定義し、過去 1 分、5 分、15 分の CPU に対するプロセスの平均待ち行列長を保持している。

そこで情報収集デーモンは、過去 1 分間の load average を用いてサーバの負荷情報とする。情報伝達間隔は原則として一定であるが、前回伝達した情報と比較して load average が 1 以上変化した場合には、その時点で情報伝達を行なう。

第 6 章

考察及び今後の課題

本章では、第 5章において実装された名前サーバについて、
その評価および今後考慮すべきことがらについて検討する。

6.1 選択の効果

広域環境において複数のサーバが利用可能である場合、それらの間の選択を動的に行なうことの評価は、サービスの種類によって異なると考えられる。

特にサービスの通信量に比較して、計算量が多い場合には、サーバの位置する計算機の計算速度の差が、全体としての処理時間に大きく影響するため、ある程度のオーバヘッドを見込んでも、負荷の軽い計算機を探すことの意味がある。実際に従来の環境で、ユーザが処理を依頼する場合にも、あらかじめ他のコマンドなどを用いて負荷を測定した後に、サービスを依頼するということが、良く行なわれていた。

本研究では、このような機能を名前サーバに実装することによって、各アプリケーションが用意に選択を行なうことのできるような環境を提供した。

しかし、逆に計算量に比較して通信量が多い場合には、一般的に言ってクライアントに「近い」サーバに処理を依頼するのが効果的である。実際にユーザが直接サーバを指定する場合にも、その瞬間のネットワークの負荷を考慮することは稀であり、ユーザの知識の中から最も近いと思われるサーバを選択してしまうことが多い。

この理由として考えられることは、一つにはサーバ毎の処理速度の差に比較して、ネットワークの通信時間の差が非常に大きいということがあげられる。すなわち、サーバ固有の計算速度の差は、大体において 10 倍以内であり、負荷によっては、実際の計算時間の逆転が起こる可能性が十分ある。しかし、ネットワークの場合、10Mbps と 64kbps など 100 倍から 1000 倍程度の差があることも多く、このような場合にはネットワークの負

荷を考慮することなく、通信時間の少ないものを決定できる。

従って、通信時間の少ないサーバを選択する必要がある場合には、これらをいくつかのグループ毎に大きく分割し、グループ内でのみ比較を行なうことによって、効率を向上させることができると考えられる。

6.2　ネットワーク情報

実際にサーバが広く分散された環境においては、クライアントとサーバの間の通信時間が問題となることも多い。特に計算量に比較して、通信量が多くなるようなアプリケーションではもちろんであるが、そうでなくてもクライアントとサーバの間で通信を行なっている限り、これに要する時間を無視することはできない。

しかし、通信時間はその経路に大きく依存し、クライアント毎に異なるため、名前サーバで一括して管理することはできない。さらにクライアント自身でこのような情報を得る場合にも、どのようにして得るかという問題が残る。

その方法としては、情報が必要になった時点で各サーバに対して通信を行なってみて、その時間を計測する方法と、あらかじめ別のプロトコルなどを用いて、各サーバへの通信所要時間を得ておく方法が考えられる。

前者の方法では、通信時間計測によるオーバヘッドが大きいため、特に大量の通信を行なう場合以外は現実的であるとは考えられない。一方後者の方法では、その情報がどの程度の間隔で更新されるかということも問題となってくる。

しかしここで、全節で述べたように、ネットワークの通信時間の差が非常に大きいことを考慮に入れると、遅延時間に対する情報は、十分実用に絶える程度の信頼性を持っていると考えることができる。

ただし、クライアントとサーバ間の経路が変化すると、これらの情報も影響を受けるため、経路情報を何らかの意味で反映させることが必要である。また、あるホストとの間の通信時間がどの程度であるかという情報は、ネットワークアプリケーションの多くにとって有用な情報であるため、その計算機自身の保有するネットワークデータベースの一部として実現されることが望ましい。

現在このようなプロトコルとして OSPF[43] が提案されている。現状ではまだワークステーション上では動作していないが、このプロトコルでは各ネットワークに対する遅延時間のパラメータも伝達されるので、クライアント側のリゾルバもこの情報を利用することによって、選択に通信時間を反映させることができる。

第 7 章

結論

計算機ネットワークが発達し、広域な環境での分散アプリケーションが考えられるようになってきた。このような分散アプリケーションでは、既存のサーバを利用することによって、処理の分散化が行なわれることが多い。

あるサービスを提供するサーバなど、計算機上の資源の位置を知るための手段として名前サーバがあるが、さらにその中から、ある条件を満たす資源の集合が必要となる場合も多い。特に、環境が広域になるにつれ、同じサービスを提供するサーバが複数利用可能である場合が生じるが、このような場合にはどのサーバを選択するかという問題が重要な意味を持ってくる。

本研究では、計算機上の個々の資源、およびそのグループを扱うことのできるような名前サーバの設計および実装を行ない、さらに一般にサービス型のオブジェクトについては、最短時間で応答を返すと予測されるようなサーバを答える機能を提供した。

予測に際しては、サーバからの情報を基に、最も待ち行列の少ないと思われるものを選択する。その際に、情報の信頼性を考慮し、信頼性が低い場合にはランダムな選択に切替えることによって、要求の集中を回避するようにした。

このような名前サーバを用いることによって、特に計算量の多いサービスについて、最適なサーバの選択を行なうことが容易となった。この点において、効率的な分散アプリケーションの構築を支援するような環境を提供した。

現在この名前サーバシステムは、Sun workstation および Sony NEWS workstation 上でプロトタイプが稼働中である。