修 士 論 文

ネーミングサービス混在環境における
ユーザ情報の一元管理に関する研究

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

坂下 幸徳

2003 年 3 月

修 士 論 文

ネーミングサービス混在環境における
ユーザ情報の一元管理に関する研究

指導教官　敷田 幹文 助教授

審査委員主査　敷田 幹文 助教授
審査委員　松澤 照男 教授
審査委員　篠田 陽一 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

110048 坂下 幸徳

提出年月: 2003 年 2 月

概 要

様々なシステムの混在環境において, ユーザ情報を一元管理することは管理コストの削減, 管理者の負担の軽減につながる. しかし, 現在一般的に利用されている NIS や Domain Controller 等のクライアントとサーバから構成される 2 層構造のネーミングサービスでは, 情報の格納形式やネットワークアクセスのアーキテクチャが異なるため，ユーザ情報を一元化することは困難である．また，ユーザ情報はコンピュータ上で利用されるネーミングサービス以外にも，人事部のような部署においては住所禄のような利用方法がある. しかし，これらの情報とネーミングサービスで管理されているユーザ情報はそれぞれが独立し, 別データベースで管理されており, 一元管理されていない．このような状態の要因の一つとして，従来のネーミングサービスはネーミングサービス独自の情報管理を行っており，他のシステムからは利用しにくい状況であるためである．そのため，他のシステムからの利用が困難になり，その結果システム全体から見た場合の管理性を低下させ，管理にかかるコストも増加し問題となっている.

本論文では，情報の一元管理を行う手法として従来方式のクライアントとサーバの間に新たに情報を変換するサーバを追加した 3 層構造ネーミングサービスを提案し，この提案方式に基づき実装した評価システムを用い評価実験を行い，その管理性・性能・運用方法において有効的なネーミングサービスを示した.

目 次

図 目 次

表 目 次

第1章　はじめに

本章では，研究背景，研究目的を述べ，最後に本論文の構成を述べる．

1.1　研究背景

近年，コンピュータの普及に伴い，様々なシステムが混在している環境が増えてきている．また，その混在環境は大規模化が進んでいる．そのような環境において，様々なシステムで利用されているユーザ情報・ホスト情報等の管理情報の分散は管理コストや管理者の負担を増大させる．管理コストの削減や管理者の負担軽減が重要視されている現状では，これらの管理情報は一元管理する傾向にある．

これらの管理情報を管理する従来のネーミングサービスはそれぞれのシステムにおいて，独自のアクセス方式と情報の格納方法を用いているため，これらを統合させることは非常に困難である．

また，ユーザ情報はコンピュータ上で利用されるネーミングサービス以外にも，人事部のような部署においては住所禄等のような別の利用がある. しかし，これらの情報とネーミングサービスで管理されているユーザ情報はそれぞれが独立し, 別データベースで管理されており, 全体としての管理性を悪くしている．このような状態の要因の一つとして, 従来のネーミングサービスはネーミングサービス独自の情報管理を行っており，他のシステムからは利用しにくい状況であるためである．

このような状況の中で，近年 LDAP(Lightweight Directory Access Protocol) [1, 2, 3] に対応した Directory サーバを用い，多くのシステムが LDAP に対応させることで管理情報を一元管理する動きが活発になっている. しかし，これは従来利用していた情報を移行することが困難であるだけでなく，クライアント全てが LDAP に対応している必要があり，対応していないクライアントは，管理することが出来ない．また，対応しているソフトウェアも，ある値を取得するのに全クライアントが同一の属性値に対して問い合わせしなければ，情報を重複させる結果となる．例えば，あるクライアントが電子メールアドレスを問い合わせる際に ‘mail-address’ で問い合わせるのに対して，別のクライアントでは ‘E-MAIL’ で問い合わせを行う場合にはそれぞれの属性値を用意し同じ値を重複させて格納しておく必要が生じる．このような状況では情報の整合性を保つのが困難になり情報の管理性が低下する．Directory サーバを利用した管理方法では，クライアントの LDAP ソフトウェアに依存してしまい，大規模な複数のシステムの混在環境において管理性が低下し利用が困難な状況にある．

1.2 研究目的

本研究の目的は複数のシステムが混在する大規模システム環境において情報を一元管理しシステム全体の管理性を向上させるネーミングサービスを提案・構築し，その有効性を検討することである．ネーミングサービスにおいてユーザ情報を一元管理することによって，ユーザ・管理者の負担を軽減することが出来ると考えられる．

現在，利用されている NIS (Network Information Service) [4], Domain Controller [5] 等のネーミングサービスは，独自の形式でユーザ情報をしている．格納されている情報に対するアクセス方式も様々な実装がされており，統一性がない．このようなネーミングサービスでは，様々なシステムが混在している環境において，システム毎にユーザを管理することになり，情報の一元管理が困難となる．

本研究では，様々なシステムが混在する環境において，それまで別々のネーミングサービスによって管理されていたユーザ情報を一元化し，その情報に対してクライアントからは，特別な変更を加えることなく利用できる方式として 3 層構造ネーミングサービスを提案・実装しユーザ情報の一元管理に関する有効性を示す．

1.3 本論文の構成

本論文の構成は，2 章で従来のネーミングサービスとその問題点について，3 章で提案方式について説明し，4 章で提案方式に基づく評価システムの実装について述べる．5 章では評価システムを用いた応答性能とキャッシュヒット率からの評価実験の結果を述べ，6 章では提案方式について管理性，性能，運用方法の観点から評価を行う．7 章ではまとめと今後の課題を述べる．

第2章　大規模システムにおける情報管理

本章では，大規模システムにおける情報管理の現状と従来方式のネーミングサービスについて説明し，ユーザ情報を一元管理するために行われてきた方法と問題点について述べる．

2.1　情報管理

現在，システムが大規模化するに従いシステムで利用される情報が増加傾向にある．システムが利用する情報は，ユーザ ID やパスワード，コンピュータ名，IP アドレス等の従来コンピュータが利用してきた情報に加え，電話番号や住所等の実世界での情報も，コンピュータで利用されるようになってきた．しかし，これらの情報の管理は各々のシステムが独自に用意したレポジトリで管理されている．他のシステムから利用されることを考慮されていない．そのため，同一人物の情報であっても複数のレポジトリに分散する結果となり，システム全体から見た場合の管理性を著しく低下させ管理にかかるコストも増加する結果となっている．

コンピュータで利用されるネーミングサービスだけを取り上げてみても UNIX と Windows で異なる管理を行っている．代表的なものに UNIX では NIS， Windows は Domain Controller がある．NIS や Domain Controller は，情報を格納しているデータベースを管理し，クライアントはこれらのデータベースに格納された情報を取り出し利用する．例えば NIS の場合，ユーザ情報を格納している/etc/passwd ファイルは NIS によって管理される．NIS サーバが passwd ファイルの情報を管理し，NIS クライアントはユーザ情報を取得するのに，NIS サーバに問い合わせを行い情報を取得する．このように NIS や Domain Controller で情報を集中管理することで，ユーザは自分のマシン以外のマシンからログインした場合においても，常に自分のマシンで設定していた環境と同じように利用することが可能となる．

NIS, Domain Controller について以下で紹介する．

- NIS
  NIS の構成図を図 2.1 に示す．NIS は Sun Microsystems によって開発され，現在 UNIX 系の OS においてユーザ情報やホスト情報を管理するのに広く利用されている．NIS はクライアントサーバモデルで構築されており，NIS サーバと NIS クライアントは RPC (Remote Procedure Call) を用い通信を行う．NIS サーバのプロセスは

ypserv, NIS クライアントのプロセスは ypbind となっており，このプロセスが RPC で通信を行うことで，情報を共有する．NIS では情報をマップファイルと呼ばれる DBM (Database Management) ファイルで管理されており，高速な検索が可能となっている．NIS サーバには，Master Server と Slave Server の 2 種類が存在し，Master Server はマップファイルの管理と配布を行い Master Server のマップファイルが更新されると更新されたマップファイルが全ての Slave Server に配布される．NIS クライアントは Master Server, Slave Server のどちらに問い合わせを行っても良い．管理者は Master Server のマップファイルを管理するだけで良い．NIS クライアントは資源にアクセスする際，必要な情報をその都度 NIS サーバに問い合わせを行いアクセスを行う．

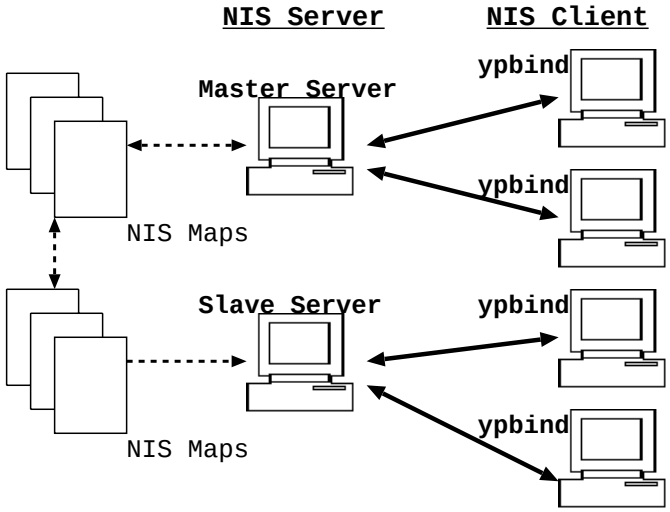

図 2.1: NIS 構成

- Domain Controller
  Domain Controller の構成図を図 2.2 に示す．Domain Controller は Microsoft Corporation によって開発され，現在 Windows においてユーザ情報を管理するのに広く利用されている．Domain Controller も NIS と同様クライアントサーバモデルで構築されており，Domain Controller とそのクライアントは NetBEUI を用いて通信を行う．Domain Controller はユーザ管理を行い，ホスト名の管理は WINS (Windows Internet Name Service) が行う．Domain Controller は情報を SAM (Security Account Manager) と呼ばれるデータベースファイルで管理を行っている．また Domain Controller は Primary Domain Controller と Backup Domain Controller の 2 種類が存在する．Primary Domain Controller は情報の管理と配布を行う．Backup Domain Controller は Primary Domain Controller が管理している情報のバックアップを行う．Domain Controller のクライアントはいずれの Domain Controller に問い合わせを行っても良い．管理者は Primary Domain Controller の情報を管理するだけで良い．WINS は IP アドレスと

NetBIOS 名の解決に利用され，通常クライアントがネットワークに接続した際にブロードキャストを送信し動的に WINS に NetBIOS 名が登録され管理される．UNIX マシン等の NetBIOS 名を持たないマシンを管理する場合には WINS に静的に情報を登録しておく必要がある．

Domain Controller クライアントはユーザがネットワークにログオンする際に認証要求を Domain Controller に送信する．Domain Controller はそのログオン要求が有効かどうかを判断したあと，ユーザのシステム識別子 (SID) と所属するグループの SID を取り出し，それらを ‘セキュリティトークン’ としてクライアントへ返す．クライアントは資源にアクセスする際に，この ‘セキュリティトークン’ と資源のアクセス制限をチェックすることにより，資源にたいしてアクセスを行う．つまり，Domain Controller クライアントは，資源を利用する度に Domain Controller への問い合わせを行わず問い合わせはログオン時の 1 回となる．ただし WINS を利用したホスト名の解決は，資源へのアクセスの都度行われる．

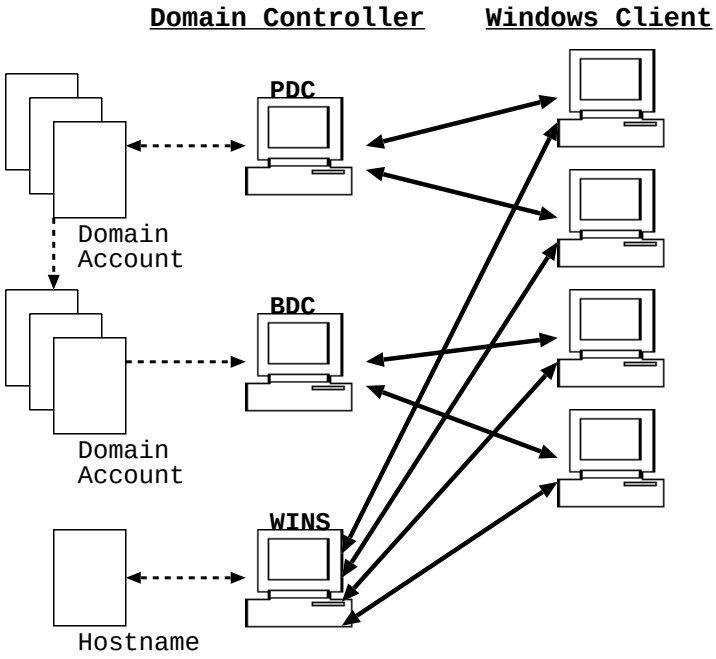

図 2.2: Domain Controller 構成

また，ユーザ管理には NIS や Domain Controller 以外にダイアルアップ接続で利用されるユーザ情報を管理する RADIUS (Remote Authentication Dial-In User Services) や RDBMS (Relational Database Management System) を利用した Web サービスのユーザ管理など，様々なシステムがある．

2.2 大規模分散システムの集中運用管理

従来のネーミングサービスが混在した環境におけるシステム構成を図 2.3 に示す．UNIX のみの環境，Windows のみの環境においては NIS, Domain Controller で情報を管理することは可能である．しかし，各ネーミングサービスはシステム構成は似ているが，ネットワークや情報格納のアーキテクチャの違いにより UNIX, Windows の混在環境においては，UNIX で管理している情報と Windows で管理している情報を一元管理することが困難になっている．このため，小規模な環境では UNIX, Windows と別々に管理を行うことは可能だが，大規模な環境では管理者の負担は増大し，別々に管理を行うことが出来なくなる．

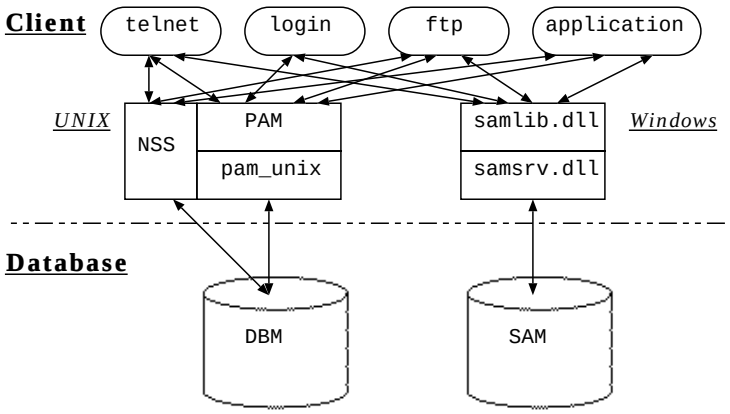

図 2.3: 従来のネーミングサービス

また，大規模な環境になるに従い管理する情報も増加する．そのため情報を格納しておくデータベースにもネーミングサービスで最適なものが必要となる．ネーミングサービスで情報を格納するデータベースにも様々なものがあり，NIS で用いられている DBM ファイル, Domain Controller で用いられている SAM 以外にも，近年ユーザ情報を格納するのに注目されているものに Directory サーバがある．主要な Directory サーバ製品は ITU-T (International Telecommunication Union-Telecommunication sector) で規格化され国際標準となっている X.500 ディレクトリサービスに準拠しており，インターネット向けに軽量化したプロトコルである LDAP でアクセスすることが出来る．Directory サーバは RDBMS と同様に様々な種類の情報を格納することが出来る．Directory サーバと RDBMS の特徴についての比較を表 2.1 に示す．RDBMS は各システムによってネットワークアクセス方式が異なるが，Directory サーバは LDAP という共通のプロトコルでアクセスすることが出来る．Directory サーバは RDBMS のような表構造の情報格納ではなくツリー構造で情報格納しており，一番先頭ノードをルートノード，節にあたるノードをコンテナ，最下位のノードをリーフと呼ぶ．コンテナにはリーフ及びコンテナを格納することが出来る．Directory サーバは，RDBMS に比べトランザクション機能が無く登録処理には時間が多くかかってしまうが，その分検索処理を高速化になるように設計されている．即ち Directory

サーバは，更新の頻度の少なく検索処理が主であるような情報の格納に適している．このような Directory サーバの特徴は，ネーミングサービスに適しており様々なシステムが LDAP に対応することで，情報の管理性を向上させることが出来る．

表 2.1: Directory サーバと RDBMS の比較

	Directory サーバ	RDBMS
ネットワークアクセス方式	LDAP	システムにより異なる
情報格納方法	ツリー構造	表構造
トランザクション機能	×	○
検索処理速度	○	△
登録処理速度	×	○

Directory サーバを利用したネーミングサービスには，Solaris の NativeLDAP や Windows の Active Directory 等があるが，これらのシステムにおいて利用している Directory サーバは格納されたツリー構成や属性名の違いにより統合することは困難である．

2.3 ユーザ情報の一元管理

現在，複数のシステムが混在する環境が増加傾向にある．そのため，これらで利用するユーザ情報等の情報を一元管理する必要性が唱えられ，研究が盛んになってきた．代表的な研究に田中の研究 [6]，倉前の研究 [7, 8] がある．

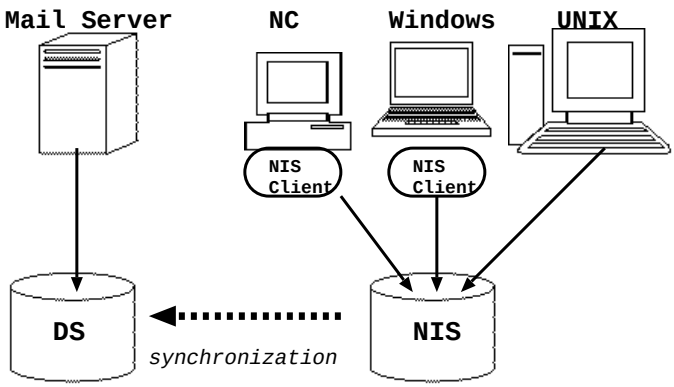

図 2.4: NIS を用いたアカウント統合

田中の研究のシステム構成を図 2.4 に示す．この研究では UNIX, Windows, Network Computer (以下 NC と略す), Mail Server の 4 つのシステムのユーザ情報を一元管理した

システムを提案・運用している．Windows, NC に別途 NIS を参照するためのソフトウェアをインストールすることで UNIX, Windows, NC のユーザ情報を NIS で一元管理している．しかしこのシステムで用いた Mail Server はユーザ情報を Directory サーバで管理され NIS のユーザ情報を取得することが出来ない．そのため，NIS と Directory サーバの間で情報を同期させ一元管理を行っている．

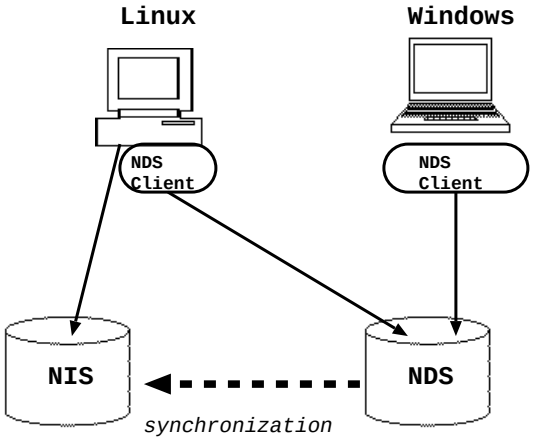

図 2.5: NDS を用いたアカウント統合

倉前の研究のシステム構成を図 2.5 に示す．この研究では Linux，Windows の 2 つのシステムのユーザ情報を一元管理したシステムを提案・運用している．ユーザ情報は Directory サーバ製品の Novell Directory Services (以下 NDS と略す) に格納し，Linux, Windows それぞれに NDS を利用するためのソフトウェアを別途インストールし構築している．しかし，Linux において NDS 環境は NIS 環境よりも 100 倍以上も応答性能が低下してしまい実用に耐えられない．そのため，Linux においてユーザ認証のみ NDS を参照し，その他の情報の問い合わせは NIS を参照するようにバイパスシステムを構築している．また NDS に格納されたユーザ情報を NIS 形式に変換し NIS と同期を取ることによって，ユーザ情報を一元管理している．

2.4 Meta-Directory による情報統合

前節までの他に複数の情報の一元管理を行う方法として Meta-Directory を利用した方法がある．Meta-Dirctory の代表的なものに Novell DirXML [9]，iPlanet Meta-Directory [10] がある．Meta-Directory を用いた情報の統合環境を図 2.6 に示す．この方法では，Meta-Directory を中心に複数データベース間で同期を取る．Meta-Directory はそれぞれの Database 間の属性のマップを管理する．このマップに従い，情報に変更が生じた場合に全てのデータベースに自動的に反映させる．例えば Account DB の ‘mail’ という属性と Mail Server の ‘mail-address’ という属性に同じ値が入ることを意味する属性のマップ

が Meta-Directory に管理されていた場合，Mail Server でユーザ A の ‘mail-address’ が変更されると自動的に Account DB に登録されたユーザ A の ‘mail’ の値が変更される．

即ち Meta-Direcory は全体のデータベースにおいて情報を一元化する方式であり，クライアントは今まで接続していたデータベースを変更する必要がない．

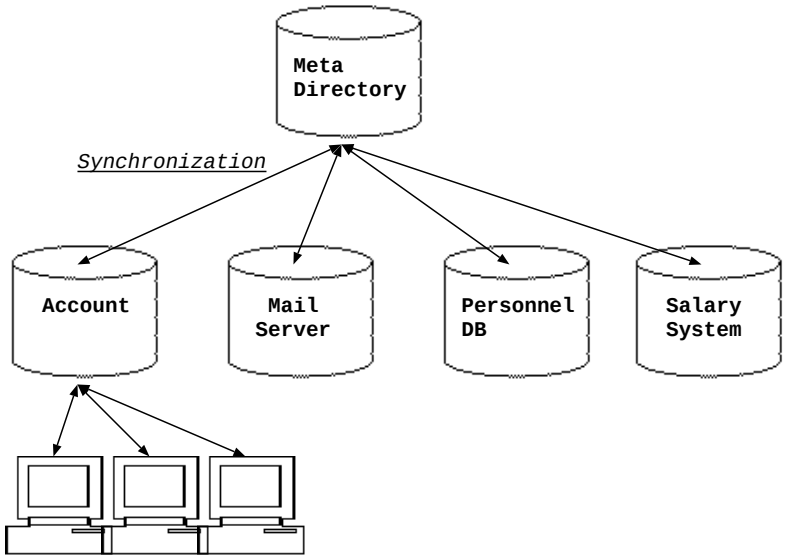

図 2.6: Meta-Directory による情報統合環境

2.5 従来方式の問題点

前節までに述べた従来方式には幾つかの問題点が存在する．

1. クライアントにソフトウェアをインストール
   両研究共にデータベースを参照するためのソフトウェアをクライアントにインストールする必要がある．これはクライアントの種別，バージョンを限定してしまうと言うだけでなく，大規模システムにおいてクライアントへのインストール作業と運用時の管理性の困難さが問題となる．
2. データベース間の同期が必要
   両研究及び Meta-Directory は複数データベースが混在し同期を必要とする．このことはデータベース間で情報の整合性を保つのが困難になるだけでなく，情報の反映にタイムラグを生じる結果となり問題となる．
3. 既存システムに依存した情報格納
   両研究及び Meta-Directory は，データベースに特定システムに依存した方法で情報を格納している．このことは，従来利用していた複数のシステムからの情報移行を困難にするまた，クライアントが特定の属性値を用いて問い合わせを行う場合，データ

ベースには，同じ値に対して複数の属性を持つ必要が生じ，結果として情報が重複し整合性を保つのが困難になる．

4. 情報量増加時のスケーラビリティが低い
   両研究共に NIS を利用しているため，情報が増加した場合の水平方向のスケーラビリティが低い．NIS は, クライアントからの問い合わせのキー毎に DBM ファイルを生成し管理をしているため，例えば NIS で管理される ‘passwd.byname’ ファイル一つに全ユーザの情報が登録されることになり．水平方向のスケーラビリティが低い．

このような問題点は管理性を悪化させる原因となり, 複数のシステムが混在する大規模システムには適しているとは言えない．

第3章　3 層構造ネーミングサービスによる一元管理方式

2 章で示した従来方式のネーミングサービスにおける問題点を解決するために，3 層構造ネーミングサービスを提案する．本章では，提案方式の構成要素となる Client Layer, Naming Server Layer, Database Layer について述べた後，Naming Server Layer で必要となる機能であるフォーマット変換機構，キャッシュ機構について述べる．

3.1　3 層構造ネーミングサービス

本研究では，従来方式の問題点を解決するために，3 層構造ネーミングサービスを提案する．NIS や Domain Controller といった従来のネーミングサービスは，クライアントとサーバの 2 層構造で構築されているが，本研究では クライアントとサーバの間に新たに情報の変換を行うサーバを追加し，3 層構造のネーミングサービスにする．これにより，クライアントに特別なソフトウェアをインストールすることなく，一元管理されたデータベースにアクセスすることを可能にする．

構成図を図 3.1 に示す．提案する 3 層構造ネーミングサービスは，Client Layer，Naming Server Layer，Database Layer から構成する．3 層構造の中間層にあたる Naming Server Layer で様々なクライアント種別の差異を吸収することにより管理性の向上を計る．

3.2　Client Layer

この層は従来利用されているネーミングサービスのクライアントを使用する．クライアントに対して特別なソフトウェアをインストールすることは移植性及び管理性を悪くするため，従来のネーミングサービスで利用されているクライアントには変更を加えない．

3.3　Naming Server Layer

Naming Server Layer の構成図を図 3.2 に示す．この層の目的は, 様々なクライアント種別の差異を吸収することと，情報量増加時のスケーラビリティを向上させることが目的である．また，3 層構造ネーミングサービスでは，この Naming Server Layer が追加され

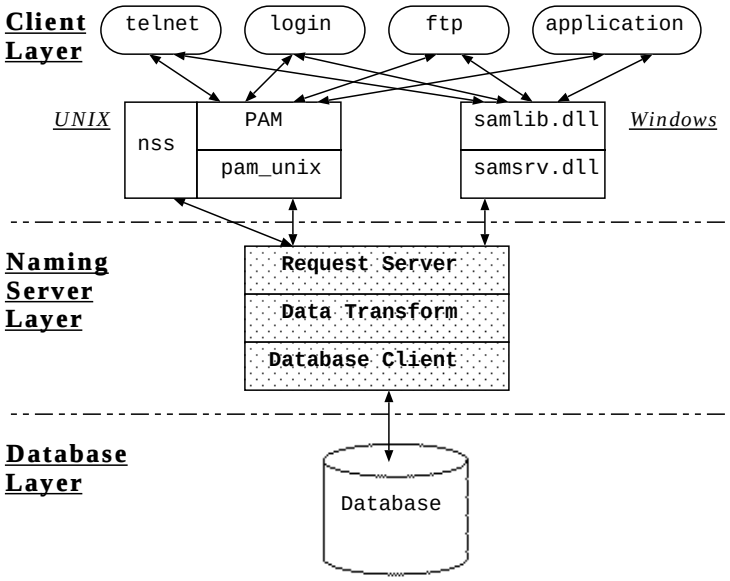

図 3.1: システム構成

ることによりクライアントへの応答性能の低下が考えられるが，Naming Server Layer にキャッシュ機構を設置することにより，性能低下を防ぐ．これらを実現するために，この層は以下の 3 つのモジュールから構成する．

- Request Server Module
  様々なクライアントに対するリクエストの送受信を行うモジュールである．このモジュールをクライアント種別に応じて用意することで，クライアントに特別なソフトウェアをインストールすることなく提案するネーミングサービスを利用出来るようにする．このモジュールで受け取ったクライアントからのリクエスト要求は Data Transform Module へ渡される．また，データベースから取得した情報は Data Transform Module から渡され，Request Server Module から各クライアントのリクエストに対する応答を行う．

- Data Transform Module
  データベースに格納された情報をそれぞれのクライアントのシステムに応じてフォーマットを変換処理するモジュールである．データベースに格納されている既存の情報から生成可能である情報に関しては，クライアント種別毎にフォーマットルールを設け，情報の自動生成を行う．フォーマットルールを記述した設定ファイルは，Request Server Module の各サーバ種別毎に用意する．このフォーマット変換機構については 3.5 節で述べる．また，応答性能を向上させるために必要なキャッシュ機構もこのモジュールに設置する．キャッシュ機構については 3.6 節で述べる．

- Database Client Module

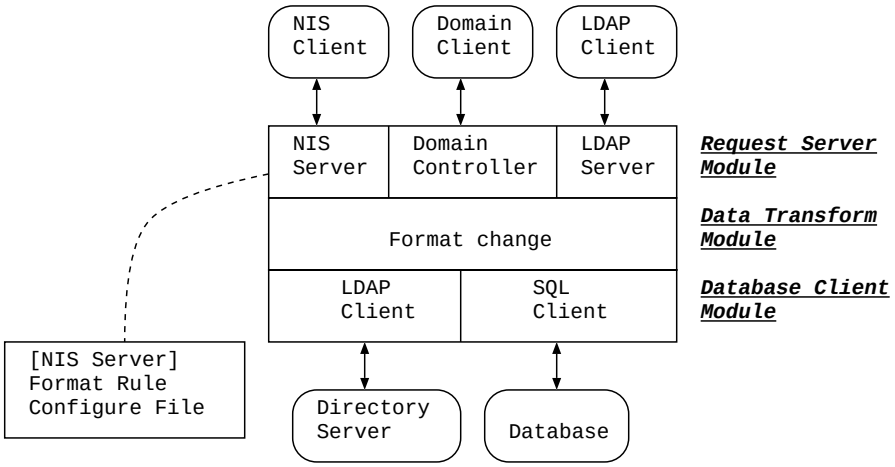

図 3.2: Naming Server Layer 構成

データベースへの応答を行うモジュールである．このモジュールを LDAP や SQL といったデータベースへのアクセス方式毎に用意することでデータベースを自由に選択することが出来る．また，複数のデータベースに対してアクセスする機能を有することで管理するユーザ数が増加し情報量が増加した場合，複数データベースに分割し情報を格納することが可能になり，水平方向のスケーラビリティの向上を図ることが出来る．

3.4 Database Layer

この層は特定のネーミングサービスに依存した情報格納方式を取らず，あらゆる値を柔軟に格納出来るデータベースを利用する．利用するデータベースには，ユーザ情報の増加にも十分耐えうるスケーラビリティを有したものが好ましい．本研究では，ユーザ情報管理に適したデータベースの Directory サーバを対象とする．

3.5 フォーマット変換機構

フォーマット変換は Request Server Module のクライアント種別毎のモジュールの設定ファイルに記述されたルールに従い，変換を行う機構である．フォーマット変換機構を有することで，以下の利点が考えられる．

- データベースに格納する情報を減少
- 情報の整合性を保つことが容易
- クライアントへ同一フォーマットの情報が提供可能

設定ファイルに記述するルールの例を図 3.3 に示す．記述するルールは，情報が格納されているホスト名，データベースに格納されたツリーの先頭ノード，対象情報を示した属性名を指定することにより格納された情報を取得するだけでなく，取得した値に対して四則演算を設定することが出来る．

データベースに格納されている情報を図 3.4 に示す．図 3.3 の ‘uid’ に関するフォーマット変換ルールに従い変換を行うと，ユーザ ‘sakasita’ の ‘uid’ は，データベースより取得した ‘1’ にルールに記述された ‘1000’ を加算しクライアントに ‘1001’ を返す．

nis_mod.conf

```
uid: %hoge.jaist.uid + 1000%
gecos: %hoge.jaist.fn% %hoge.jaist.sn%
```

図 3.3: フォーマット変換設定ファイル例

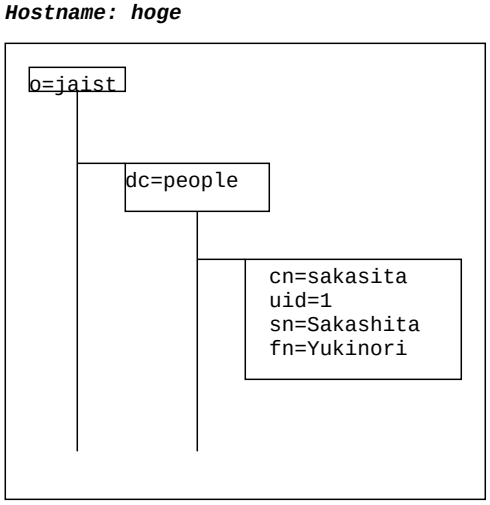

図 3.4: データベースに格納されたデータ

次にフォーマット変換処理の流れを図 3.5 に示す．Naming Server Layer にはあらかじめ読み込んだフォーマット変換ファイルに，‘gecos’ の変換ルールが記述されている．

1. クライアントからユーザ ‘sakasita’ の ‘gecos’ 情報を要求する
2. Namig Server Layer は設定された ‘gecos’ の情報からデータベース ‘hoge’ のツリーの先頭ノード ‘jaist’ 配下の情報に対して属性値 ‘fn’, ‘sn’ を要求する
3. データベースでは，指定されたツリーノード配下の情報よりクライアントから指定されたユーザの属性値 ‘fn’, ‘sn’ を検索し Naming Server Layer へ返す
4. Namig Server Layer では，データベースより取得した属性値を用いて，フォーマット変換を行いクライアントへ返す

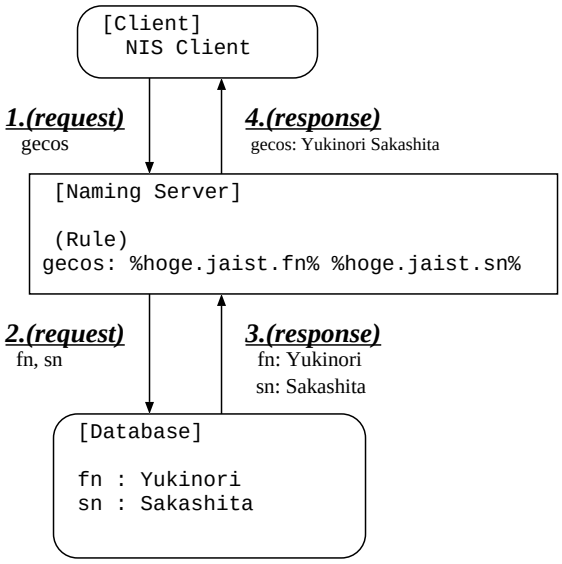

図 3.5: フォーマット変換処理

3.6 キャッシュ機構

キャッシュ機構は，ネーミングサービスを 3 層構造にすることによって考えられるネットワークによる遅延をより少なくするための機構である．ネーミングサービスの情報は，情報の変更が少なくアクセスする情報も集中化していると考えられるために中間層である Naming Server Layer にキャッシュ機構を有することは，ネットワークによる遅延に対して有効であると考える．図 3.6 にキャッシュ機構がある場合とない場合の各サーバへのアクセスを示す．キャッシュ機構がない場合にはクライアントから Naming Server Layer, Naming Server Layer からデータベースへとアクセスが発生し，要求と応答を考えると合計 4 回のネットワークアクセスが発生する．これが Namig Server Layer にキャッシュ機構がある場合では，クライアントから Naming Server Layer にアクセスした後，Naming Server Layer 上に対象の情報がキャッシュされていれば，その情報をクライアントへ返す．これにより合計 2 回のネットワークアクセスで情報をクライアントへ返すことが出来る．これは，従来の 2 層構造のネーミングサービスと同等のネットワークアクセスの回数となる．

Naming Server Layer のキャッシュ機構の構成を図 3.7 に示す．キャッシュリストから情報を検索する際の KEY として，データベースから情報を取得した際に POSIX 1003.1 で提供されている関数を参考にし値を登録する．例えば，ユーザ情報の場合，getpwnam 関数では引数にユーザ名の文字列を指定しネーミングサービスからユーザ情報を取得し．getpwuid 関数では引数にユーザ ID の数字を指定しユーザ情報を取得する．このような場合，データベースからユーザ情報を取得した際にクライアントからの問い合わせの KEY としてユーザ名とユーザ ID の 2 種類が考えれれる．そのため，この 2 つの属性値をキャッシュリストの検索 KEY として登録する． 2 つの登録した検索 KEY からは, 同一の情報へ

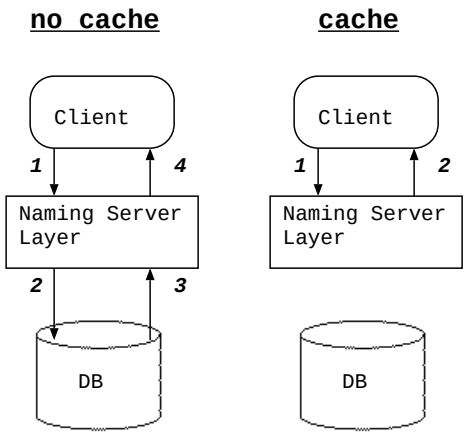

図 3.6: キャッシュ機構のデータアクセス

のリンクを設定する．ユーザ情報以外の情報においても 2 つ以上の検索 KEY が必要となるため，メモリ使用量を抑えることが出来る．

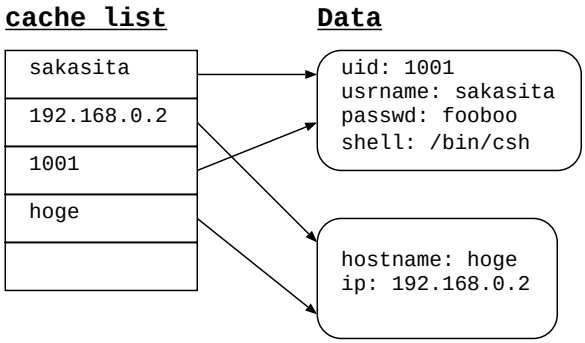

図 3.7: キャッシュリストとデータ

第4章　評価システム

本章では 3 章で提案した 3 層構造ネーミングサービスを実装した評価システムについて述べた後，新たに追加された Naming Server Layer についての実装と，評価システムで利用した Directory サーバに格納した情報について述べる．

4.1　評価システムの概要

提案する方式に基づき評価システムの実装を行った．実装したシステムを図 4.1 に示す．Client Layer には，Solaris 8 (10/01) の NIS クライアント (ypbind) 及び，Windows 2000 Professional を用いた．Naming Server Layer は，Request Server Module に NIS Server モジュール, Domain Controller モジュール, Database Client Module に LDAP Client モジュールを用意した．Database Layer には，オープンソースの Directory サーバである OpenLDAP 2.0.25 [11] と商用の Directory サーバである iDS (iPlanet Directory Server 5.1) [12] をそれぞれ利用した．評価システムで管理される情報はユーザ情報及びホスト名・ IP アドレスとした．

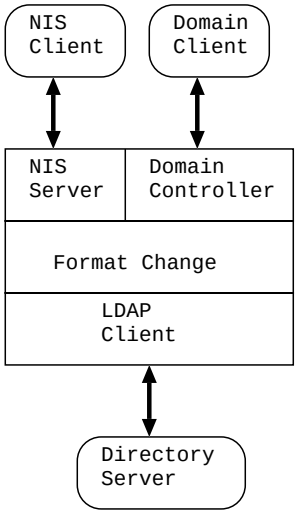

図 4.1: 評価システム

4.2 実装環境

評価システムは表 4.1 に示す環境で開発を行った．実装に利用した NIS サーバ，Domain Controller が C 言語で記述されており，LDAP 関連の C ライブラリが OS にバンドルされているために，親和性を考慮し Naming Server Layer の開発は C 言語で行った．Naming Server Layer の NIS モジュールの開発には，Solaris 8 Source Foundation, Domain Controller モジュールの開発には Samba 2.2.4-ja-1.0 [13] をそれぞれ利用した．LDAP 関連のライブラリは特定の Directory サーバに性能が依存がしないように OS に付属のライブラリを利用した．

表 4.1: コンパイル環境

マシン	Sun Ultra 5 (CPU: UltraSPARC-IIi 400 MHz, Memory: 256 Mbytes)
OS	Japanese Solaris 8 10/01 + Recommend Patch
コンパイラ	Sun WorkShop Compilers 5.0

4.3 Naming Server Layer の実装

実装した Naming Server Layer の構成図を図 4.2 に示す．Request Server Module は ypserv, smbd, nmbd から，Data Trans Module は nis_mod, smb_mod, infotrans から，Database Client Module は ldaputil からそれぞれ構成される．

Naming Server Layer は各クライアントからのリクエストを受信するサーバプロセスと，フォーマット変換やデータベースへのリクエストを行うサーバプロセスが存在する．前者をサブプロセス，後者をメインプロセスと呼ぶ．Naming Server Layer のプロセスについて図 4.3 に示す．メインプロセスとサブプロセスは IPC (Interprocess Communication) の Doors [14] で情報のやり取りを行う．メインプロセスを Doors のサーバ，サブプロセスを Doors のクライアントとして実装した．Solaris ではクライアントが Doors のサーバ手続きを呼び出すたびに，サーバプロセス内のスレッドが処理を行い，必要に応じてサーバプロセス内に新しいスレッドを生成される．スレッド管理は，Doors ライブラリが自動的に行う．従来の ypserv 等のプロセスではローカルのデータファイルを利用し情報を取得しており，データファイルを検索中に他の処理は行われない．評価システムで利用した Doors は同期手続き呼び出しであるため，従来の ypserv 等のプロセスがローカルのデータファイルを利用するのと同様の応答を実装し易い．これらの利点より IPC に Doors を用いた．

このような構成で実装された評価システムの Naming Server Layer は，以下の手順に従い処理される．

1. クライアントから，それぞれのネーミングサービスで利用されるプロトコルでリクエ

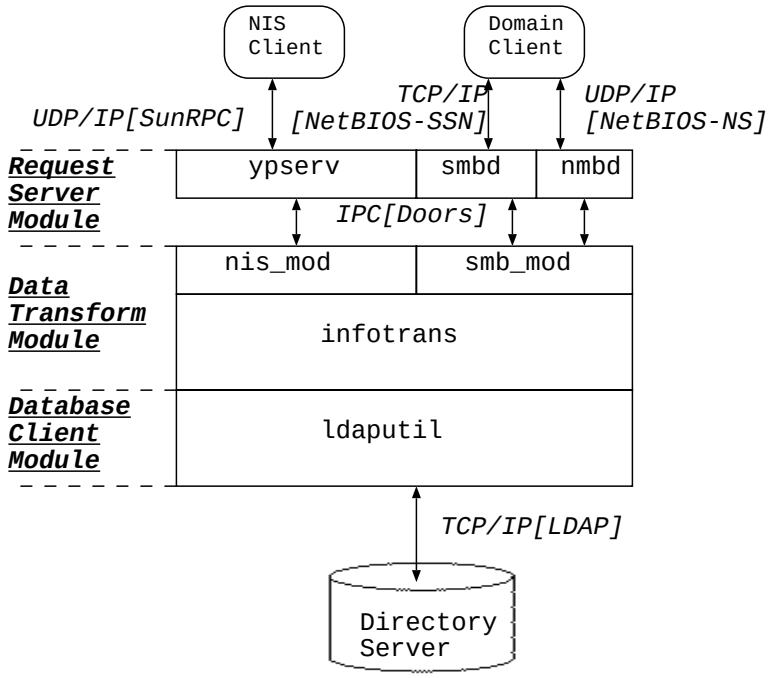

図 4.2: Naming Server Layer 構成

ストを送信する

2. 改良したサブプロセスはクライアントからのリクエストを受信し，リクエスト内容を IPC を用い Naming Server Layer のメインプロセスへリクエストを送信する
3. サブプロセスからのリクエストを受信すると，リクエストを共通形式に変換し infotrans に受け渡す
4. infotrans では受け取った要求からキャッシュリストを検索し，対象情報が存在しない場合に ldaputil にデータベースの検索要求を出す. 対象情報が存在する場合には，対象情報を nis_mod (smb_mod) に返す
5. ldaputil では infotrans より受け取った検索要求をもとに，Directory サーバへアクセスを行い情報を取得する
6. データベースから得られた情報は，infotrans へ返されキャッシュリストに情報を追加すると共に nis_mod (smb_mod) へわたす.
7. nis_mod (smb_mod) はデータベースから得られた情報をもとに各クライアントのフォーマットに変換する
8. 変換された情報を IPC でサブプロセスへ情報を返す
9. サブプロセスからクライアントへ情報を返す

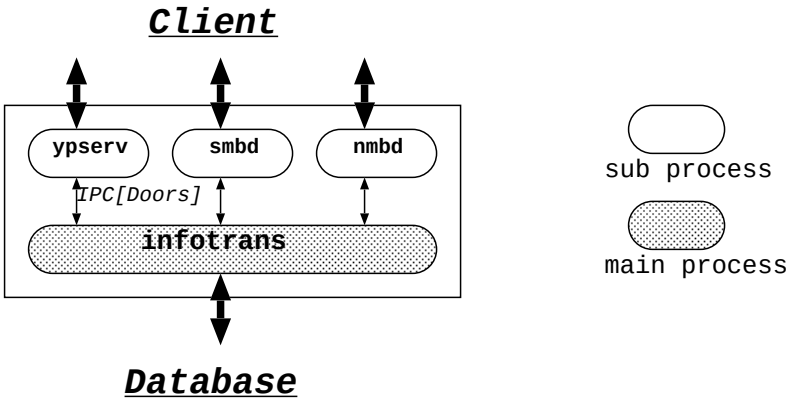

図 4.3: Naming Server Layer のプロセス

4.3.1 Request Server Module

サブプロセス (ypserv, smbd, nmbd) は，既存のプログラムを改良し利用する．提案方式の ypserv の構成を図 4.4 に示す．従来の ypserv は，クライアントからリクエストを受信後，ypdispatch 関数，ypmatch 関数を経て ypset_current_map 関数で情報の格納されたローカルのデータファイルを開き dbm_fetch 関数で対象の情報を検索し，結果をクライアントへ返す.

評価システムでは従来の ypserv を改良し実装を行った．改良点は ypmatch 関数から initDoors 関数を呼び出し，Doors クライアントの設定を行う． その後 transYpmatch 関数で設定された Doors クライアントを用いてメインプロセスへリクエスト内容を送信する．従来の ypserv で利用していたローカルのデータファイルへのアクセスは行わない.

4.3.2 Data Transform Module

構成を図 4.5 に示す．このモジュールは各クライアントに依存した処理と共通処理の 2 つに分割し実装を行った．nis_mod (smb_mod) が前者にあたり infotrans が後者にあたる．

クライアントに依存した処理では，サブプロセスとの通信とクライアント種別に応じたフォーマット変換が主である．まず始めに getnisparam 関数でサブプロセスから Doors 利用しクライアントからのリクエストを受ける．受け取ったリクエストは，共通処理の関数 getresponse 関数に渡す．この時，getresponse 関数へはリクエスト種別と検索キーを渡す．例えばクライアントからのリクエストが，ユーザ ‘hoge1’ のユーザ情報を取得するものだった場合，リクエスト種別には ‘passwd.byname’, 検索キーには ‘hoge1’ が getresponse 関数へ渡される．共通処理を経て得られた情報は，transPasswd 関数によってフォーマット変換を行い getnisparam 関数からサブプロセスへ返す.

共通処理では，Database Client Module への検索要求と取得した情報の管理が主である．getresponse 関数は，searchHash 関数を用いてキャッシュリストに検索対象の情報が存在す

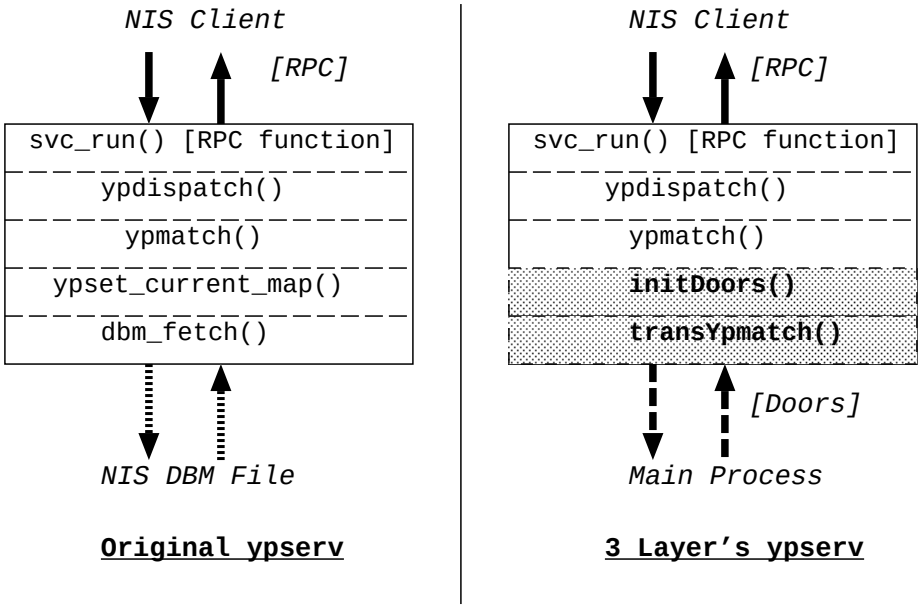

図 4.4: Request Server Module

るかを検索し，存在しなければ getldapdata 関数から Database Client Module の ldaputil に実装された initldap 関数にデータベースへの検索要求を出す．Database Client Module から取得した情報は，setHash 関数でキャッシュリストに登録される．キャッシュリストに登録する際に，取得した情報の先頭ノードにアクセス時間を追加する．このアクセス時間は，キャッシュリストからこの情報が呼び出される度に更新される．また，このアクセス時間が古い情報に対しては，キャッシュリストから削除を行う．キャッシュリストに登録後，getResValue 関数を通じて nis_mod の transPasswd 関数に情報を受け渡す,

データベースから取得した情報を管理するキャッシュリストは図 4.6 に示す構造体を 1 つのノードとした配列で表現したハッシュで構成している．ハッシュでは，図 4.7 に示すチェイン法を用いている．チェイン法では同じハッシュ値をもつ情報をリストで繋ぐことにより，キャッシュリストのサイズに関係なく情報を登録することが出来る．ハッシュ値を求める hash 関数は，与えられた文字列の ASCII コードの総和とキャッシュリストの配列数の剰余を返す単純な計算としている．

4.3.3 Database Client Module

Directory サーバからクライアントからの要求に応じた情報を検索し取得する処理の実装を行った．構成を図 4.8 に示す．Data Transform Module から呼び出される initldap 関数から Directory サーバへのコネクションオープン，検索，情報格納，コネクションクローズの処理を順次呼び出し処理を行う．コネクションには TCP/IP のプロトコルである LDAP を用いて Directory サーバへ接続する．Directory サーバへの認証は ldap_simple_bind_s 関

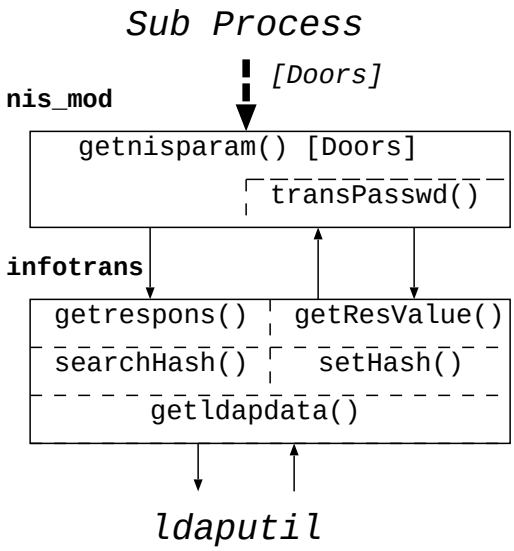

図 4.5: Data Transform Module

```
typedef struct cell {
    char key[MAXLEN]; /* SEARCH KEY */
    Resultdata *data;
    struct cell *next;
} CELL;

CELL hash_tbl[HASHSIZE];
```

図 4.6: CELL 構造体

数を用い匿名でバインドを行っている．認証を匿名で行うことにより，パスワード情報を持たないようなホスト名・IP アドレスといった情報を取得することが出来る．検索は，searchEntry 関数で検索リクエストを生成後，Directory サーバへ検索要求を出す，検索リクエストには クライアントから指定されたリクエスト種別により判別できるツリーの先頭ノードと検索キーを指定する．検索対象としては，指定されたツリーの先頭ノード配下のツリーを全て検索対象として設定した．検索によって取得された情報は，setResultdata 関数で図 4.9 で示される Resultdata 構造体をノードとしたリスト構造に格納し，コネクションを切断する．切断後，取得した情報を Data Transform Module に渡す．

4.4 Database Layer の実装

評価システムでは Database Layer に Directory サーバを利用し情報を格納した．Directory サーバの格納する情報種別を決定するスキーマには IETF (Internet Engineering Task Force)

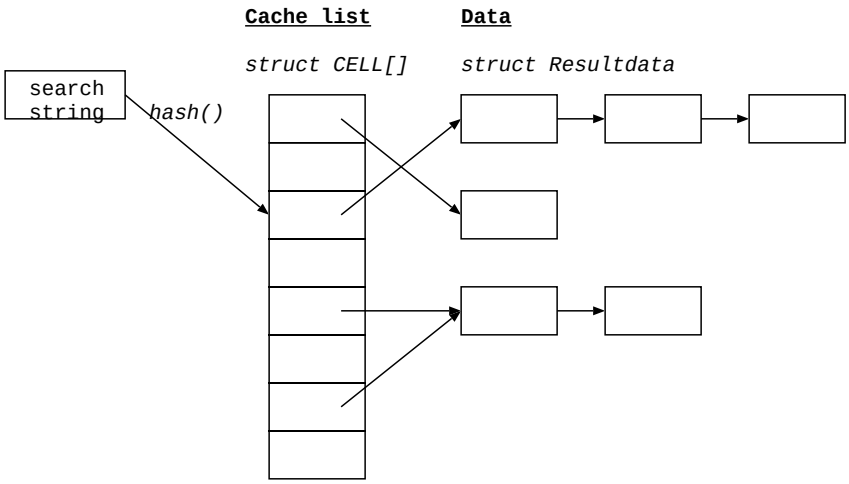

図 4.7: キャッシュリスト (チェイン法)

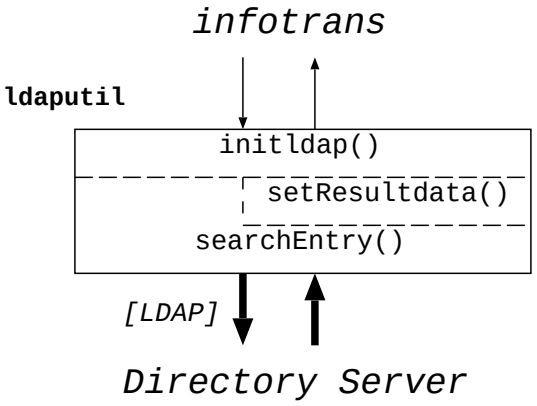

図 4.8: Database Client Module

で管理されている RFC (Request for Comments) で定めれらたスキーマ以外に，独自のスキーマを追加し設定した．評価システムで設定したスキーマを表 4.2 と表 4.3 に示す．ユーザ情報のパスワードには，NIS, Domain Controller といったシステム毎のパスワードではなく，暗号方式や表記法毎にパスワードを格納する．

情報は図 4.10 に示すツリー構造に格納する．全てのルートになるノードには ‘o=jaist’ を設定する．これは Directory サーバにおいては「jaist という組織」を意味する．ルートノードの直下には ‘dc=people’ と ‘dc=host’ と 2 つにツリーを分類した．‘dc=people’ は，ユーザ個人に関する情報を示したツリー, ‘dc=host’ は，コンピュータに関する情報を示したツリーとした．また本評価システムでは実装していないがグループに関する情報は, ‘dc=people’, ‘dc=host’ と同レベルの階層に ‘dc=group’ を作り管理するの良いと考える．

グループは個々のユーザやコンピュータ等の資源の情報をまとめて扱う必要があり，場

```
struct _resultdata {
    char key[MAXLEN]; /* Attribute type */
    char value[MAXLEN];
    struct _resultdata *next;
};
typedef struct _resultdata Resultdata;
```

図 4.9: Resultdata 構造体

表 4.2: ユーザ属性

属性名	説明	定義
cn	共通ネーム (ユーザ名)	RFC2377
sn	名前	RFC1274
fn	苗字	独自
uid	ユーザ ID	RFC1274
uidNumber	ユーザ ID (NIS)	RFC2307
gidNumber	グループ ID	RFC2307
homeDirectory	ホームディレクトリ	RFC2307
loginShell	ログインシェル	RFC2307
crypt-passwd	DES 暗号によるパスワード (ASCII 表記)	独自
des-passwd	DES 暗号によるパスワード (16 進数表記)	独自
md4-passwd	MD4 暗号によるパスワード (16 進数表記)	独自
dc-acct-ctrl	Windows アカウントフラグ	独自
acct-lct	アカウント情報最終更新時刻	独自

合によってはユーザと資源がグループのメンバーになるというような構成があるためである.

表 4.3: ホスト属性

属性名	説明	定義
cn	共通ネーム (ホスト名)	RFC2377
ipHostNumber	IP アドレス	RFC2307

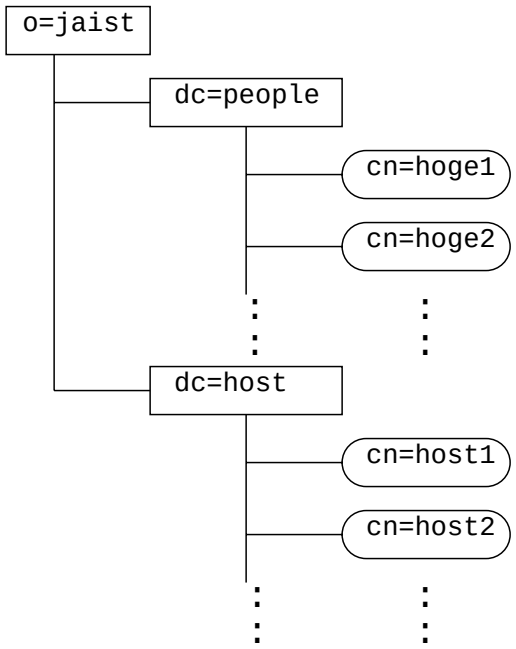

図 4.10: ツリー構成

第5章　評価実験

4 章で実装した評価システムを用い，提案方式の性能について行った評価実験について実験方法と結果を述べる．

5.1　概要

提案する 3 層構造ネーミングサービスの評価システムを用い，評価実験を行った．評価実験は以下の 2 つの観点から行った．

- 応答性能
- キャッシュヒット率

応答性能は，従来の 2 層構造ネーミングサービスを 3 層構造にすることによってクライアントへの応答性能の低下が考えられる．そのため，この応答性能を測定することにより 3 層構造にする場合に応答性能のボトルネックになる要因を調査することを目的としている．

キャッシュヒット率は，評価システムにおいて Naming Server Layer に構築したキャッシュ機構の有効性を調査することを目的としている．

5.2　応答性能

提案方式で追加された Naming Server Layer の応答性能について複数の観点から測定を行い，従来方式に比べ応答性能の低下の要因を調査した結果について述べる．

5.2.1　測定環境

評価システムの評価環境を図 5.1 に示す．Sun Blade 150, Sun Enterprise 3000 は OS に Solaris 8 (10/01), IBM Thinkpad R31 には Windows 2000 Professional を利用した．2 層構造ネーミングサービスとして NIS を対象とした．NIS クライアントには Sun Blade 150, NIS サーバには Sun Enterprise 3000 をそれぞれ用いた．3 層構造ネーミングサービスでは Client Layer に Sun Blade 150 と IBM ThinkPad R31, Naming Server Layer に Sun Blade

150, Database Layer に Sun Enterprise 3000 をそれぞれ利用した．それぞれのマシン間は，Layer 2 Swiching HUB を用い 100 Mbps のネットワークで接続した．NIS クライアント，Client Layer のマシンはネーミングサービスの設定ファイルである nsswitch.conf の記述を図 5.2 のように設定し，NIS 及び評価システムを利用出来るように設定した．

応答性能の測定にあたり，測定用プログラムを作成した．測定用プログラムでは UNIX においてユーザ情報を取得する getpwnam 関数を用いた．NIS クライアント，Client Layer のマシンからネーミングサービスにたいしてリクエストを 1000 回行い 1 ユーザあたりの平均の応答速度を求めた．

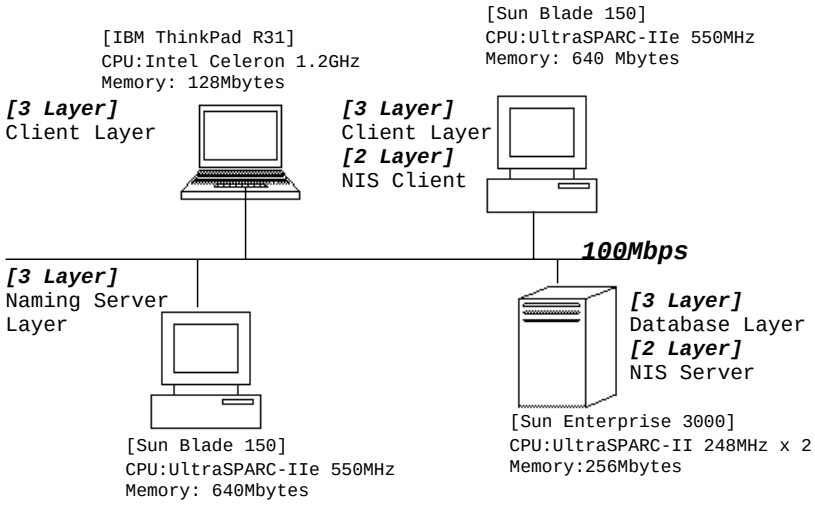

図 5.1: 評価環境

```
passwd: files nis
group : files
hosts : files nis
netmasks: files
ethers: files
automount: file
```

図 5.2: nsswitch.conf

5.2.2 クライアントからの応答性能

2 層構造ネーミングサービスの NIS と 3 層構造ネーミングサービスの評価システムを用いてクライアントからの応答速度を測定した．測定は，データベースに登録するユーザ

を 1 ユーザから 500 ユーザまで変化させ応答速度を測定を行った．尚，この測定に用いた評価システムでは，キャッシュ機構の処理を外している．

測定結果を図 5.3 に示す．この結果より，全てのシステムにおいて登録しているユーザ数に関係なく一定の応答速度となった．また, 500 ユーザ登録時に評価システムでは NIS の 8.87 ミリ秒に比べ Database Layer に OpenLDAP を使用した場合 1.77 倍の 15.68 ミリ秒，iDS を使用した場合 1.89 倍の 16.66 ミリ秒の応答速度となった．

iDS では OpenLDAP に比べ，Directory サーバ自体の管理に必要な情報が多く存在し，ツリーデータが格納されている DBM ファイルの総サイズを比較しても，iDS の方が遥かに大きい．そのため，必要最低限の情報のみを管理している OpenLDAP の方が iDS に比べ応答速度が，速くなっていると考えられる．

次節からの測定は OpenLDAP を用い，登録ユーザ数を 500 ユーザと固定で測定を行った．

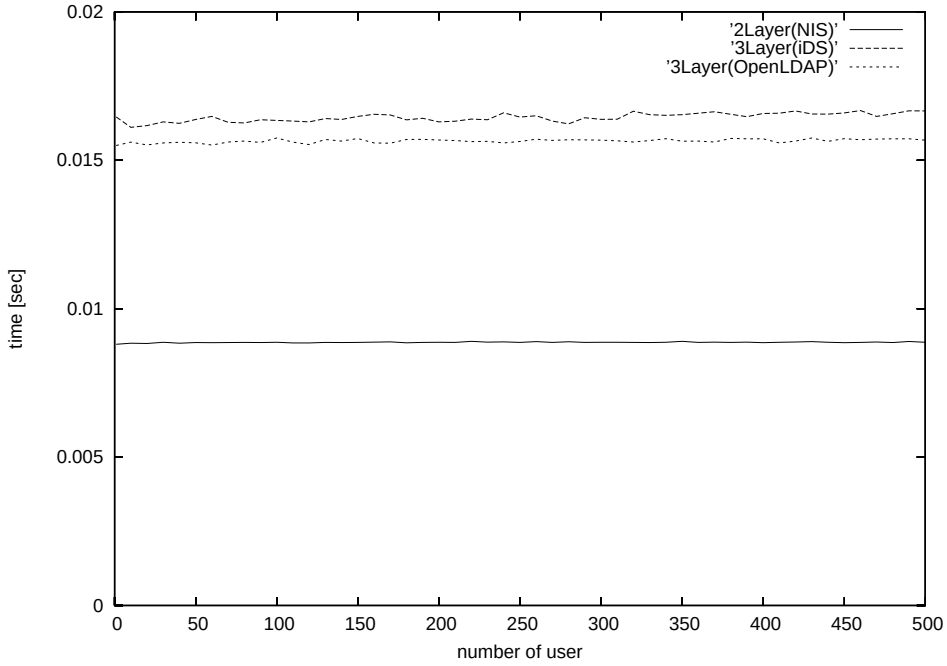

図 5.3: 1 ユーザあたりの応答時間

5.2.3 フォーマット変換機構による遅延

評価システムを用いて，Naming Server Layer でのフォーマット変換機構による応答速度の遅延を測定した．

測定は，Naming Server Layer にユーザ情報の ‘GECOS’ を生成するルールを設定し測定を行った．設定したルールは四則演算を含んだものを設定する．1 個の変換を行うルールを図 5.4 に示す．このようなルールで 10 個の変換ルールまでを設定し測定した．

```
gecos: %hoge.jaist.uid + 10%
```

図 5.4: GECOS 生成ルール

測定結果を表 5.1 に示す．10 個の変換を行った場合，変換を行わなかった場合に比べ 0.54 ミリ秒の遅延が発生する．1 個の変換では平均 0.05 ミリ秒とわずかな遅延となる．フォーマット変換機構では，あらかじめ設定ファイルにより設定されたルールは Naming Server Layer を起動時にメモリ上に読み込んでおく．情報は，Database Layer より取得されメモリ上にある情報を用い変換を行う．そのため，全てメモリ上に展開されている情報での処理が可能となり，フォーマット変換機構での遅延は小さいものとなる．

表 5.1: フォーマット変換機構による遅延

変換数	応答時間 (ミリ秒)
0	15.68
1	15.74
10	16.22

5.2.4 アクセス回数による遅延

前節までは Naming Server Layer から 1 回の Directory サーバへのアクセスで指定したユーザの必要な情報を全て取得していた．この測定では必要な情報が発生する度に Directory サーバへアクセスを行い Directory サーバへのアクセスで生じる応答速度の遅延について測定を行った．

測定は，フォーマット変換機構でルールに記述された属性値の問い合わせに対して，メモリ上の情報を利用せずに新たに Directory サーバへアクセスし情報を取得するように評価システムを改良し行った．

測定結果を図 5.5 に示す．

アクセス回数が増加した場合の応答速度の低下はアクセス回数に比例しており，図 5.5 の近似式 (5.1) で示される．

$$f(x) = 0.00723x + 0.00845 \tag{5.1}$$

即ち 1 回のアクセスにつき応答速度が 7.23 ミリ秒低下する．

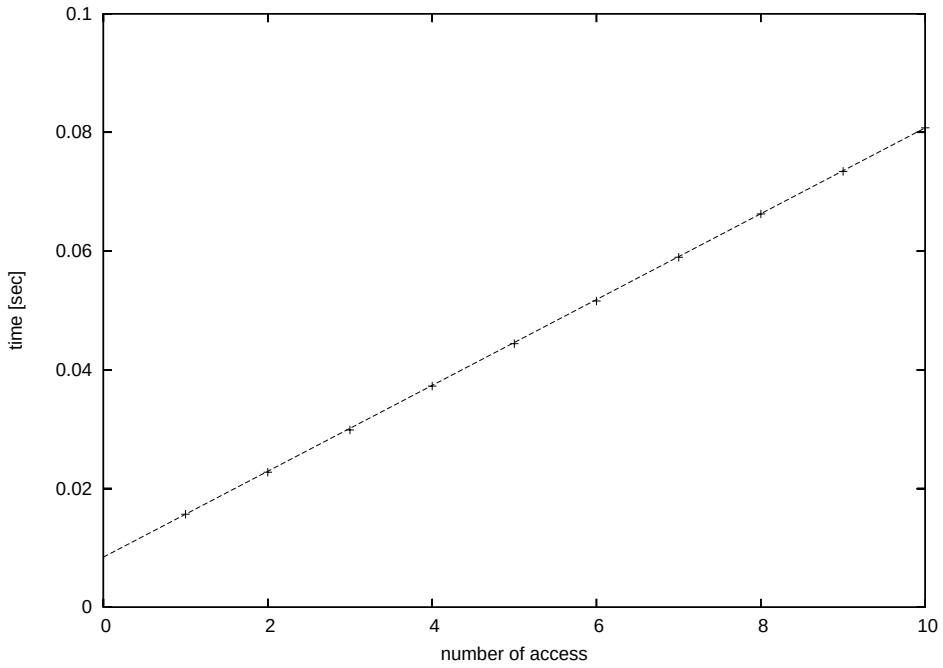

図 5.5: データベースアクセス回数による遅延

5.2.5 キャッシュ機構を利用した応答性能

3 層構造ネーミングサービスにおいて Naming Server Layer にキャッシュ機構が存在する場合の応答速度について測定を行った.

測定を開始する前に対象ユーザの情報を全てあらかじめキャッシュリストに登録しておく．この時，キャッシュリストの削除は行わない．即ち，この測定ではキャッシュのヒット率は 100 % となる.

測定結果を図 5.6 に示す．キャッシュにヒットした場合の応答速度は 2 層構造ネーミングサービスの NIS とほぼ同じ速度となった．また，近似式 (5.1) は Directory サーバへのアクセスが 0 回の時に，キャッシュにヒットした際の応答速度を示す．この場合の応答速度は 8.45 ミリ秒となり，この測定での応答速度とほぼ変わらない速度を示した.

5.2.6 検索指定の速度

評価システムを用いて Naming Server Layer から Database Layer への検索処理のリクエストに関して測定を行った.

Directory サーバへの検索処理のリクエストを行う関数として Solaris 8 が提供する LDAP ライブラリが提供する ldap_search_s 関数がある．この関数では，取得する属性を指定し必要な属性のみを取得する方法と，指定せずにエントリーの全ての属性を取得する方法が提供されている．この属性を指定した場合のリクエストと，指定しない場合のリクエスト

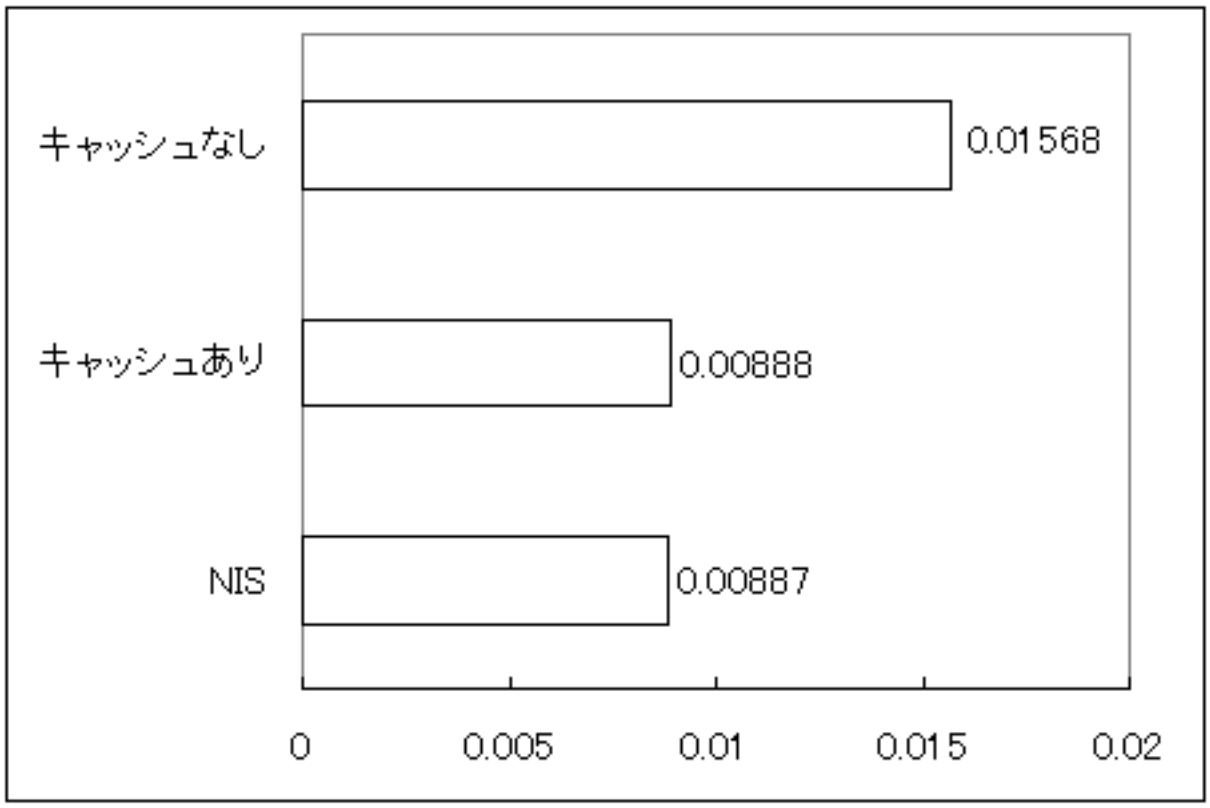

図 5.6: キャッシュ機構の応答性能

についてパケットサイズと応答速度を測定した．

NIS のユーザ情報と RADIUS のユーザ情報を Database Layer に格納した環境において属性を指定した検索処理の場合と指定しなかった検索処理の場合のパケットサイズを比較した．Directory サーバに NIS で必要なユーザ情報の属性 8 個を登録した環境，RADIUS で必要なユーザ情報の属性 55 個登録した環境をそれぞれ設定した．RADIUS のユーザ情報には Sun Directory Services 3.1 [15] のスキーマを参考にし属性を設定した．

図 5.7 に結果を示す．

属性を指定しないリクエストパケットは NIS のユーザ情報を格納した場合，RADIUS のユーザ情報を格納した場合共に 100 Bytes である．属性を指定するリクエストパケットは NIS のユーザ情報属性を指定した場合 194 Bytes，RADIUS のユーザ情報属性を指定した場合 682 Bytes となる．属性を指定しない場合のレスポンスパケットは NIS のユーザ情報を格納した場合 352 Bytes, RADIUS のユーザ情報を格納した場合 1213 Bytes である．RADIUS のユーザ情報が格納されている場合に NIS のユーザ情報属性を指定した場合のレスポンスパケットは 296 Bytes， RADIUS のユーザ情報属性を指定した場合は 1151 Bytes である．レスポンスのパケットサイズの差は Directory サーバに登録されてはいるが，利用をしない ‘objectclass’ 属性の値を返しているためであると考えられる．

次に Directory サーバに NIS で必要なユーザ情報の属性のみを設定した．この環境で属

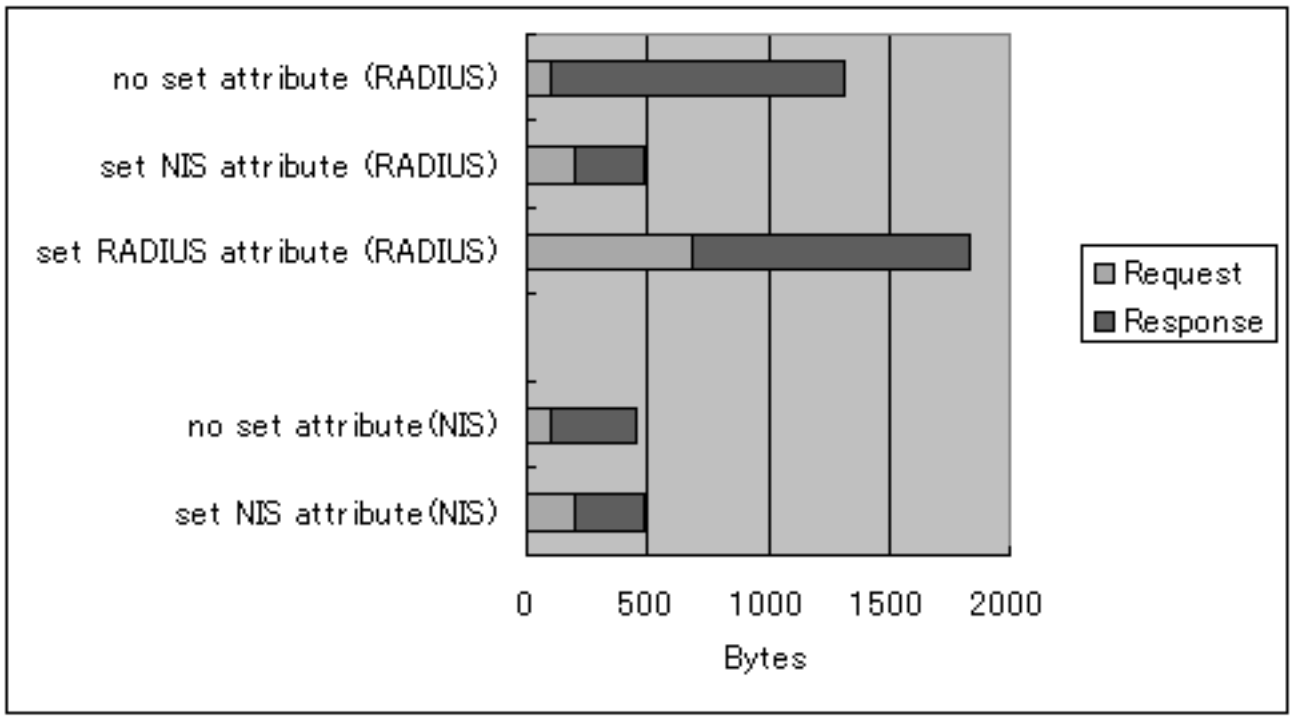

図 5.7: 属性値指定によるパケットサイズ

性を指定した検索処理と指定しない検索処理の応答速度を比較した．

表 5.2 に結果を示す．属性を指定した検索処理の方が，指定しない場合に比べ 2.21 ミリ秒遅くなる．

表 5.2: NIS 情報登録時の応答速度

	応答時間 (ミリ秒)
属性を指定しない	15.68
属性を指定する	17.89

次に Directory サーバに RADIUS で必要なユーザ情報の属性を設定した．この環境で属性を指定した検索処理と指定しない検索処理を行い応答速度を測定した．

表 5.3 に結果を示す．NIS のユーザ情報を指定した場合と RADIUS のユーザ情報を指定した場合で比較を行うと RADIUS のユーザ情報を指定した場合の方が 9.42 ミリ秒遅くなった．NIS のユーザ情報を取得するのに属性を指定する場合としない場合では，属性を指定しない方が 18.50 ミリ秒速くなった．

また，表 5.2 と表 5.3 から属性を指定しない検索処理の場合に， Directory サーバに属性が 1 個増加されると 0.04 ミリ秒の遅延が発生することが解る．

表 5.3: RADIUS 情報登録時の応答速度

	応答時間 (ミリ秒)
指定なし	17.67
NIS 属性を指定	36.18
RADIUS 属性を指定	45.60

属性を指定した場合としない場合のパケットサイズと応答速度の比較を行うと，パケットサイズが増加したことによる応答速度の低下よりも，属性を指定したことによる Directory サーバの応答性能の低下が大きくなった．

5.3 キャッシュヒット率

評価システムの Naming Server Layer に構築したキャッシュ機構の有効性を調べるために，実運用されているネーミングサービスのパケットデータを用い，キャッシュのヒット率を調査した結果について述べる．

5.3.1 運用データ採取環境

3 層構造ネーミングサービスの Naming Server Layer に構築するキャッシュのヒット率を調査するために，実運用されているネーミングサービスのパケットデータを採取した．図 5.8 の環境で NIS, Domain Controller それぞれのネーミングサービスのパケットデータを採取した．Domain Controller クライアントとして Windows 2000 Professional のマシンを 2 台，Windows XP Professional のマシンを 2 台， NIS クライアントとして Solaris 8 のマシンを 3 台, Solaris 9 のマシン 1 台を対象とした．ユーザは 4 名としそれぞれ Solaris のマシンと Windows のマシンを 1 台ずつ利用する．NIS サーバには Solaris 7 にバンドルされている NIS サーバを対象とした．管理されている情報は，本学で運用されている情報とした．Domain Controller は Samba 2.2.4-ja-1.0 の Domain Controller 機能及び WINS 機能を用い，Domain Controller のユーザ情報にはパケットデータ採取環境を利用する 4 名の情報を登録し設定した．ネーミングサービスのパケットデータは，それぞれのネーミングサーバのマシンで採取した．パケットデータの採取には Solaris にバンドルされている snoop コマンドを利用した，パケットデータ採取期間は 1 週間とし，その中から最もマシンの利用が高い 12 時間分のデータを取り，次節以降の測定に用いた．ユーザ情報に関するリクエスト数は NIS サーバで 1193 リクエスト，Domain Controller で 105 リクエスト，ホスト情報に関するリクエスト数は NIS サーバで 1870 リクエスト，Domain Controller で 761 リクエストを対象としキャッシュのヒット率を求めた．

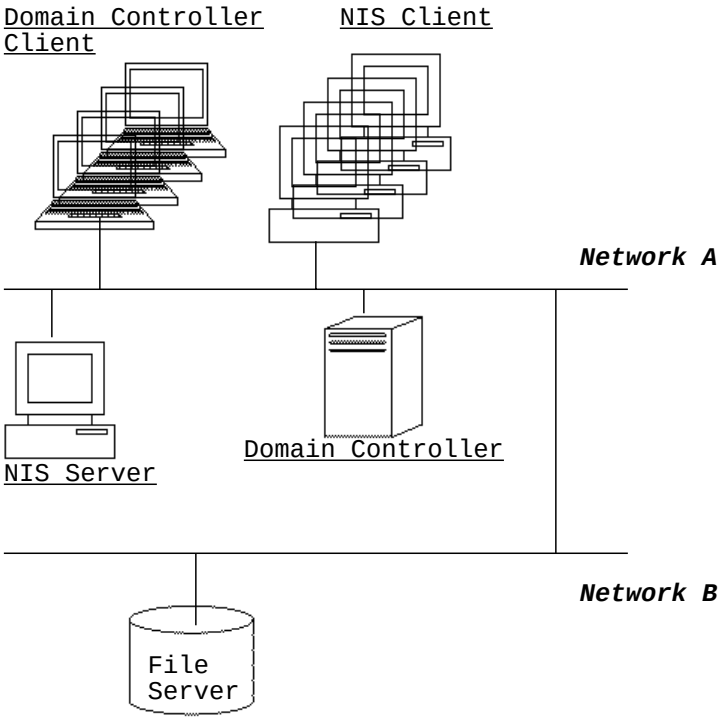

図 5.8: パケットデータ採取環境

5.3.2 ネーミングサービスの利用種別

5.3.1 節で採取した NIS サーバのパケットデータよりクライアントからリクエストがある情報種別の統計データを図 5.9 に示す．本学の環境では，ユーザの HOME ディレクトリやメールスプール，コンパイラ等が格納されたディレクトリが NFS で提供されている．そのため，これらのディレクトリに対するアクセスが発生すると NFS の Mount が発生し，その際に利用される ‘hosts.byname’，‘auto.local’, ‘auto.local.c’ の利用頻度が高くなっている．

5.3.3 ヒット率測定プログラム

キャッシュヒット率は，採取したパケットデータからヒット率を求めるプログラムを作成し求めた．測定プログラムでは，評価システムで実装を行っているキャッシュ機構をシミュレートした．評価システムではヒット率を求めるために，採取したパケットデータの実時間が必要となるが，測定プログラムでは有効時間を計算上の数字として扱うため，短時間でキャッシュのヒット率を求めることが出来る．また，利用する 12 時間分のパケットデータのうち，計算で用いた最大有効時間よりも大きい 2 時間分のデータをあらかじめキャッシュに登録する情報として利用し，残り 10 時間分のデータを利用してヒット率

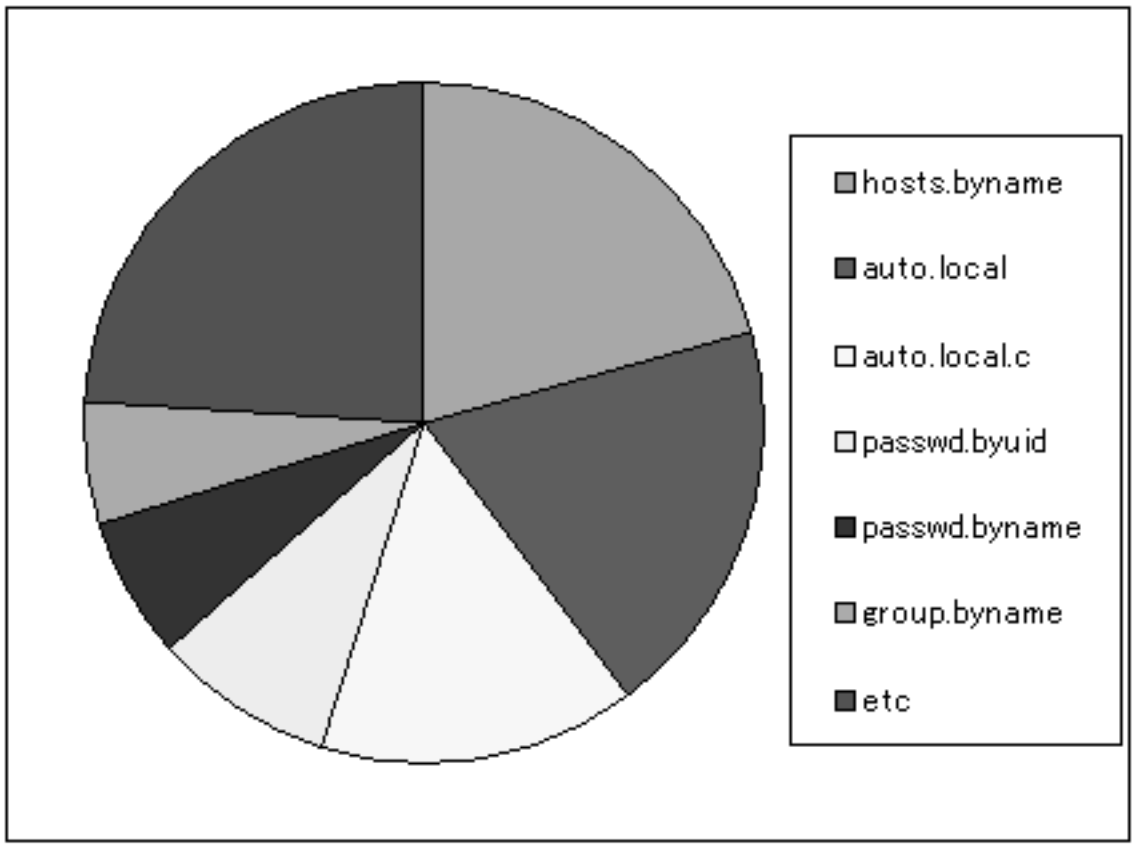

図 5.9: NIS 情報種別

を求めた．次節以降キャッシュヒット率はこのプログラムを用いヒット率を求めた．

5.3.4 キャッシュ有効時間

採取したパケットデータよりネーミングサービスにおいてキャッシュリストに登録しておく有効時間について求めた．対象の情報はユーザ情報とホスト情報とした．

図 5.10 に NIS の結果を示す．ユーザ情報の方がホスト情報に比べ，キャッシュの有効時間が短くても高いヒット率を示す傾向となった．ユーザ情報，ホスト情報共に約 1200 秒でヒット率がほぼ一定になる．また，キャッシュのヒット率を 85 % 以上を目標としキャッシュの有効時間を設定する場合，ユーザ情報は 800 秒，ホスト情報は 1200 秒必要となる．

図 5.11 に Domain Controller の結果を示す．Domain Controller クライアントは，短い時間間隔で同じユーザ名，ホスト名に対してリクエストする傾向がある．そのためヒット率の変化は，急激な変化を伴う傾向となった．

Domain Controller は NIS と異なり，ホスト情報の方がユーザ情報に比べ，キャッシュの有効時間が短くても高いヒット率を示す傾向となった．ユーザ情報は約 11300 秒，ホスト情報は約 2000 秒でヒット率がほぼ一定になる．また，キャッシュのヒット率を 85 % 以

上を目標としキャッシュの有効時間を設定する場合，ユーザ情報は 7900 秒，ホスト情報は 800 秒必要となる．

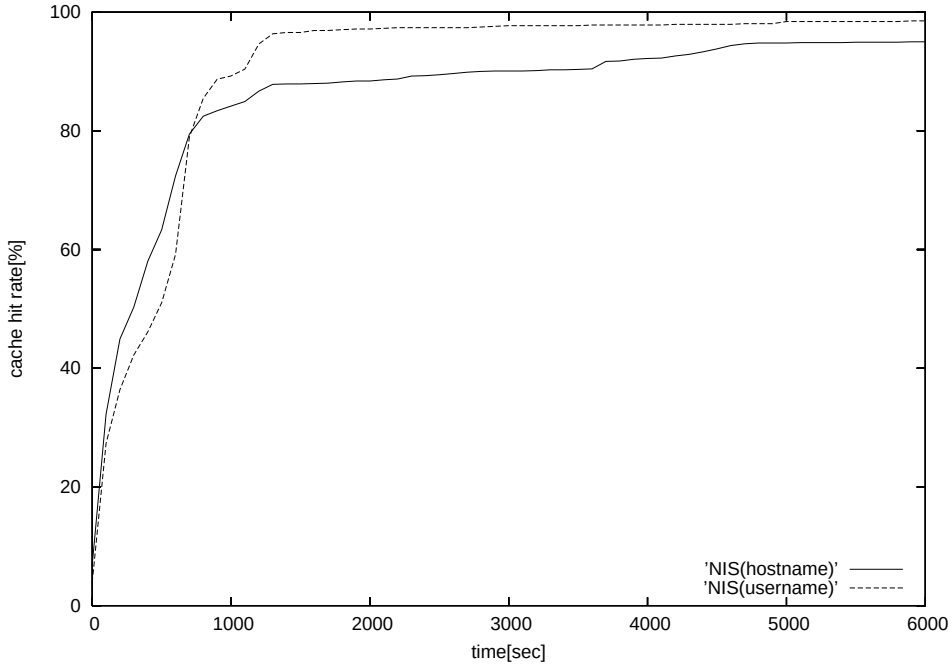

図 5.10: キャッシュリスト有効時間 (NIS)

5.3.5 ユーザ情報におけるヒット率

採取した NIS サーバと Domain Controller のパケットデータを混在させ，ネーミングサービスを統合した環境におけるユーザ情報のヒット率を求めた．NIS クライアントからみたヒット率を図 5.12 に示す．‘MIX-NIS(usrname)’ が混在環境のヒット率を示す．NIS クライアントでは，わずかながらヒット率があがるものの影響は少ない．

Domain Controller クライアントからみたヒット率を図 5.13 に示す．Domain Controller クライアントでは，統合環境の方がヒット率が向上した．向上の最も高いキャッシュの有効時間を 2800 秒に設定したとき，統合環境の方が 28 % 向上した．また，キャッシュのヒット率を 85 % とすると混在環境では 2700 秒の有効時間となった．これは，5.5.3 節の結果と比較すると統合環境の方が，ユーザ情報に関して 85 % 以上のヒット率を目指したキャッシュの有効時間を設定した場合， 8600 秒短く設定出来ることを示す．

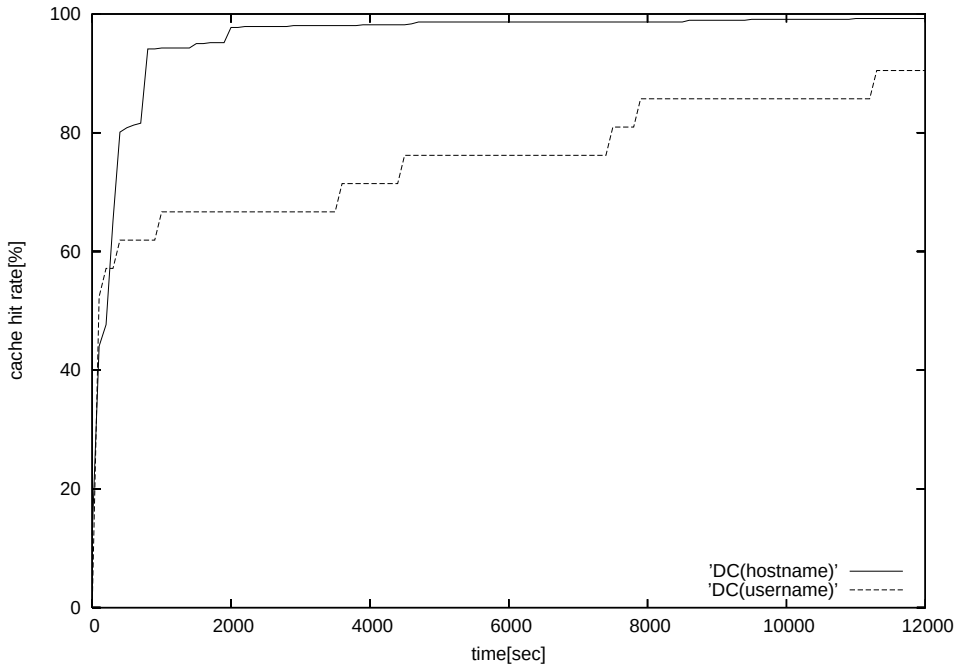

図 5.11: キャッシュリスト有効時間 (Domain Controller)

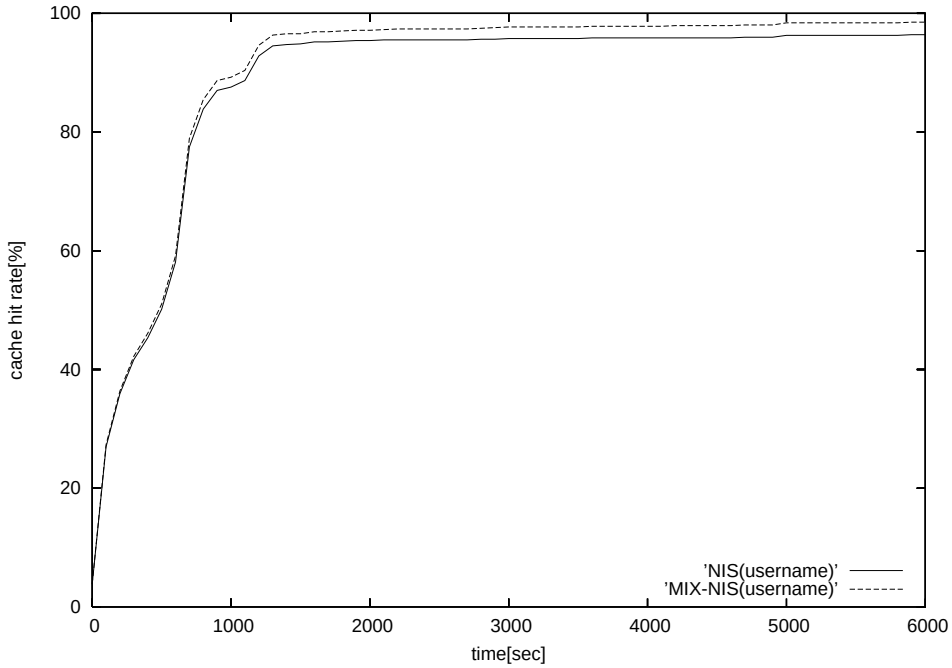

図 5.12: 混在環境における NIS クライアントからのヒット率 (ユーザ情報)

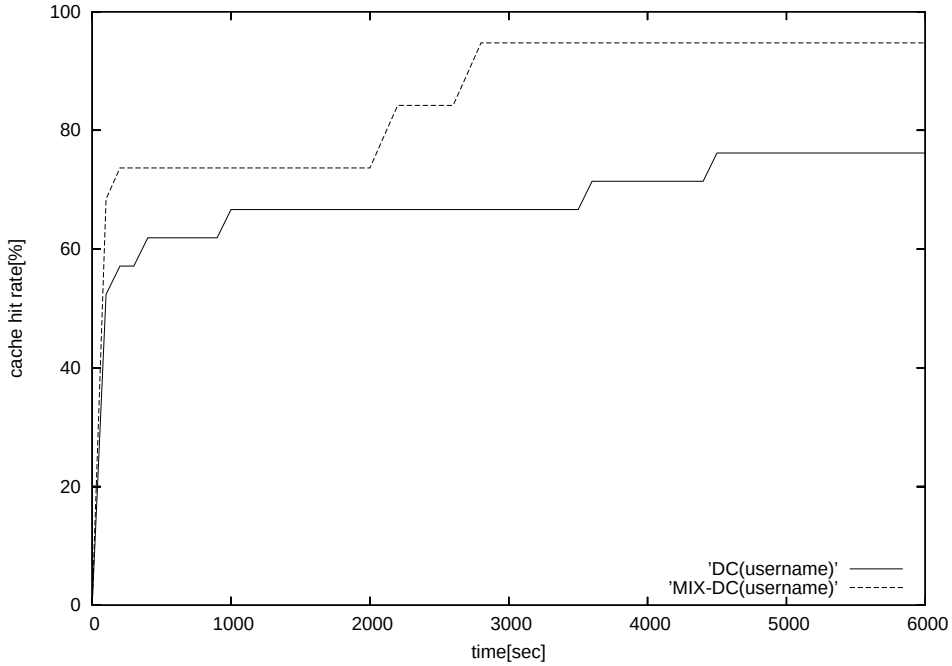

図 5.13: 混在環境における Domain Controller クライアントからのヒット率 (ユーザ情報)

5.3.6 ホスト情報におけるヒット率

採取した NIS サーバと Domain Controller のホスト情報のパケットデータを混在させ, ネーミングサービスを統合した環境におけるホスト情報のヒット率を求めた．採取した NIS サーバと Domain Controller のパケットデータから図 5.14 を想定した実験データを作成し，ヒット率を求めた．‘Mount Environment 1’ (以下 ME1 と略す) はユーザ自身のホームディレクトリが格納されている File Server を示すのに Domain Controller クライアントは Domain Controller が提供している資源として利用する．つまり Domain Controller クライアントのユーザは，あたかも Domain Controller 自身のディスクであるかのように利用する環境である．‘Mount Environment 2’ (以下 ME2 と略す) では NIS クライアントが利用している File Server を直接 Domain Dontroller クライアントが利用する環境である．

図 5.15 に NIS クライアントからのヒット率を示す．‘MIX-NIS(hostname)’ が ME1, ‘MIX2-NIS(hostname)’ が ME2 をそれぞれ示す．ME1 では，ヒット率の向上は少ない．これに比べ ME2 では，ヒット率の向上は大きい．向上の最も高いキャッシュの有効時間の 300 秒に設定したときに，Domain Controller の情報と統合していない環境と比べ ME1 は 8 % 向上し，ME2 は 30 % 向上した．ホスト情報に関して 85 % 以上のヒット率を目指したキャッシュの有効時間を設定した場合，ME1 では 1000 秒，ME2 では 700 秒の有効時間となった．

図 5.16 に Domain Controller クライアントからのヒット率を示す．‘MIX-DC(hostname)’

が ME1, ‘MIX2-DC(hostname)’ が ME2 をそれぞれ示す．ME1, ME2 いずれの環境においてもほぼ同程度のヒット率の向上がある．向上の最も高いキャッシュの有効時間の 200 秒に設定したときに，統合していない環境と比べ ME1 は 21 % ，ME2 は 25 % ヒット率が向上した．ホスト情報に関して 85 % 以上のヒット率を目指したキャッシュの有効時間を設定した場合，ME1, ME2 共に 700 秒の有効時間となった．

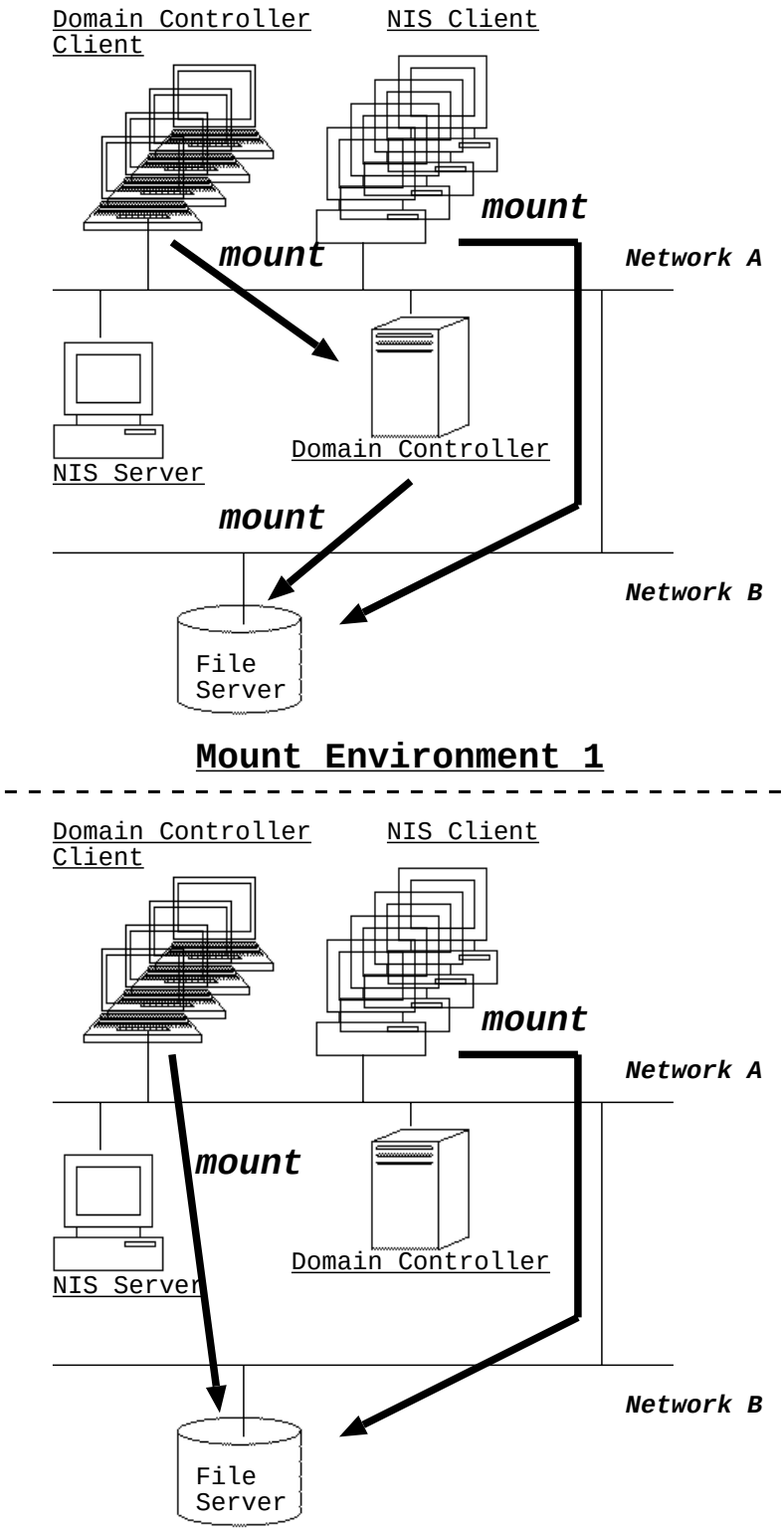

図 5.14: 混在環境における Mount 設定

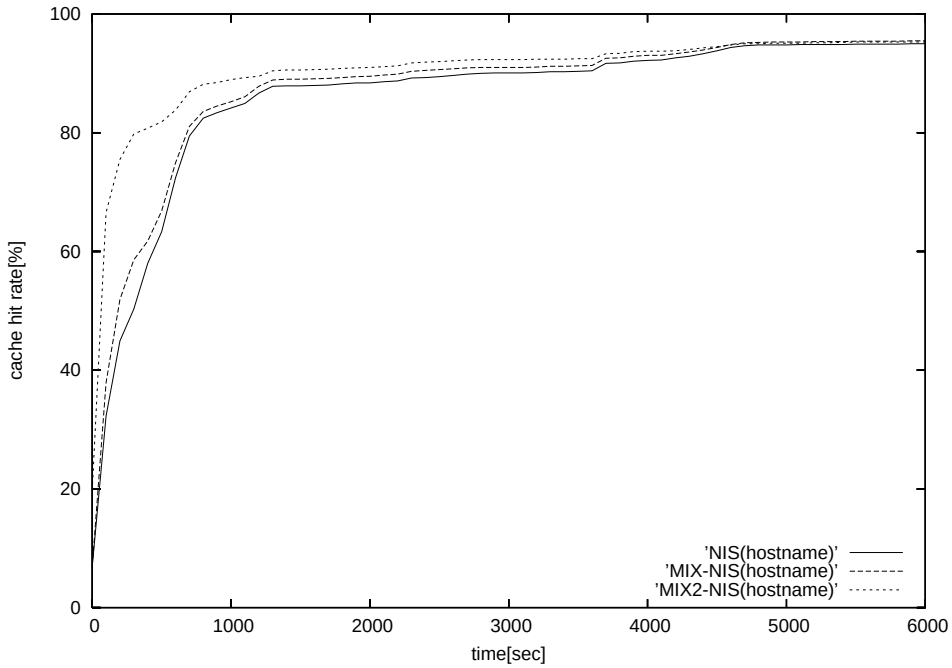

図 5.15: 混在環境における NIS クライアントからのヒット率 (ホスト情報)

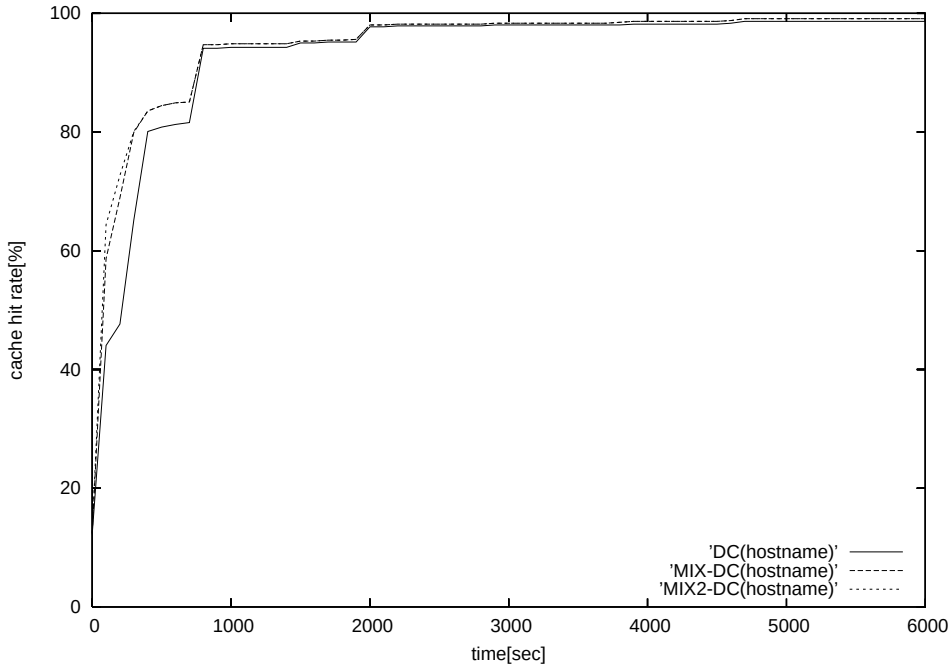

図 5.16: 混在環境における Domain Controller クライアントからのヒット率 (ホスト情報)

第6章　議論

提案方式について，管理性，性能，運用方法の 3 つの観点から議論を行う．管理性では関連研究との比較，性能では従来方式との比較を述べる．運用方法では，提案方式の有効的な運用方法である組織間を跨った運用方法について述べる．

6.1　管理性

3 層構造ネーミングサービスでは 2 層構造ネーミングサービスに比べ下記項目の管理性を向上させることが出来ると考える．

1. 従来システムからの移行性が向上
   大規模環境において，最も台数が多いクライアントに統合したネーミングサービスを利用するためのソフトウェアをインストールする必要がないため，従来システムからの移行性を向上することが出来る．Naming Server Layer でクライアントのネーミングサービスに対応させることで，ユーザは意識することなしに一元管理された情報を利用出来る．また，これまでに Database Layer に格納された情報に対して変更することなく新たなネーミングサービスに対応することが出来る．
2. データベース間の同期が不要
   データベース間で同期を取らず一元管理を行うことが出来るため，情報に変更があった場合のタイムラグを無くすことが出来る．また，従来方式で利用していたネーミングサービス用のデータベースを統合することが出来るため，ネーミングサービスで利用するデータベース数を減らすことが出来る．
3. 特定システムに依存しない情報の格納
   Naming Server Layer で，クライアント種別に応じたフォーマットへの変換・生成を行うことで，データベースを柔軟に設計することができ，情報の整合性を保つことが容易になる．データベースに特定システムに依存しない形式で情報で格納出来るために，他のシステムからの情報の利用性が高く，情報の集中管理を行い易くなる．
4. 情報量増加に対するスケーラビリティが向上
   管理する情報量の増加により，単体のデータベースでは対応しきれなくなった場合でも Naming Server Layer で複数のデータベースにアクセスするように設定を変更することで，水平方向のスケーラビリティを向上させることが出来る．
5. 共通のフォーマットでの情報提供が可能

Naming Server Layer でクライアントに返す情報を生成することで，管理者は自由にクライアントに提示する情報のフォーマットを変更することが出来る．これは，クライアントに返す情報に特定の値を追加する必要が生じた場合，従来までは全ての情報に対して変更をする必要があったが，提案方式では Naming Server Layer のフォーマット変換機構の設定ファイルを変更するだけで，全ての情報が変更出来る．例えば，NIS の ‘GECOS’ に氏名のみを提供していたシステムにおいて，新たに電話番号の情報を追加したいと考えたとき，従来までは全てのユーザの情報を変更する必要があったが，提案方式ではフォーマット変換機構の設定ファイルを変更するだけで良い．

6. 情報変更についてのアクセス制御が設定可能
   管理上ユーザに変更して欲しくない属性にたいしてユーザが変更要求をした場合，各クライアントに要求エラーを返すことができ，より細かな情報管理が出来る．これは，評価システムでは実装していないが Naming Server Layer において，クライアントからの要求を受けた際，属性を判断することで可能となる．また，Naming Server Layer で判断を行う方法の他に Database Layer へ要求し Database Layer でのアクセス制御を利用することも可能である．例えば Directory サーバでは，属性についてアクセス制御を出来るだけでなく，変更要求を出すユーザに応じてアクセス制御を設定することが出来る．これは，ある組織において部署やグループといった更に細かな組織での管理者を置き，その管理者に関しては，管理下にあるユーザの情報にたいして書き換え可能といったアクセス制御を設定することが出来る．Domain Controller より多くの情報を扱う Active Directory では，自分の上司や部署名といった実社会での情報も扱う．このような情報を有効利用することにより，より細かな組織での管理者を容易に設定することが出来る．いずれの方法においても，Naming Server Layer で対応することができ，クライアントたいして変更は生じない．一般的な制御に関しては Naming Server Layer で設定するほうが適しており，部署などで細かく設定が必要な場合には Database Layer の機能を利用したほうが良い．

提案方式による利点は，わずかな移行作業でこれまで利用してきた環境のユーザ情報の一元管理をすることが出来るようになり，さらに様々なシステムが混在する大規模なシステムにおいて管理性を高め，管理コストを削減することが出来る．

また，問題点として以下の項目がある．

1. パスワード情報のような特殊な情報が存在
   パスワード情報のような特殊な情報については，他の情報と比べ情報の統合が困難になる場合がある．これは，セキュリティの問題や表現方法の違いによるためだが，パスワード情報は他の情報に比べそれぞれが異なる情報であっても，ユーザには比較的受け入れやすいという性質がある．また，運用時に全てのシステムで同一のパスワードを利用する場合，それぞれ異なったパスワードを利用する場合と運用する環境によってもパスワードの取り扱いのポリシーが異なる．そこで，同一のパスワードを利用する場合には全てのパスワードを一斉に変更するプログラムを提供する．Database

Layer の情報格納にはシステム毎のパスワードという形式ではなく，暗号方式や表記法毎のパスワードという形式にすることにより，多くのネーミングサービスを統合した場合，登録する情報量を減らすことが可能となる．

2. 障害発生時に全てのシステムに影響
   従来までは UNIX のネーミングサービスに障害が発生した場合には UNIX 環境のみ，Windows のネーミングサービスに障害が発生した場合には Windows 環境のみしか影響を受けなかった．しかし提案方式では，全ての環境に影響する．これについては．NIS などの従来方式と同様の障害対策が可能であり Naming Server Layer を複数配置し障害発生の影響を少なくし，Database Layer については HA (High Availability) Cluster 構成を取る事で，従来方式と同等以上の障害対策をとることが出来る．また，Namign Server Layer ではデータを持っていないため，メンテナンス性が良い，
3. Naming Server Layer, Database Layer 間のセキュリティーが脆弱
   評価システムにおいては，Database Layer への問い合わせに暗号化がされていないプロトコルである LDAP を用いている．そのため，Naming Server Layer と Database Layer 間のセキュリティーが低くなっている．ネーミングサービスで扱う情報にはパスワード情報のような機密性の高い情報も含まれているために高いセキュリティーが要求される．そこで暗号化のされていないプロトコルの LDAP ではなく，SSL (Secure Sockets Layer) によって暗号化された Directory サーバへのアクセスプロトコルである LDAPS を利用することで，高いセキュリティーを保つことが出来る．しかし，LDAPS を利用した場合，暗号化処理が追加され応答性能が低下することが考えれるが，Naming Server Layer と Database Layer とのアクセス回数をキャッシュ機構によって減らすことで，低下を抑えることが出来る．

提案方式は，ネーミングサービスを 3 層構造にすることによって従来方式より管理性を向上し，問題点については運用方法で回避することが出来る．

次に提案方式と 2.3 節で紹介した関連研究との比較を行う．比較結果を表 6.1 に示す．関連研究はいずれも複数のデータベースを同期させる構成をとっており，この同期作業に関して各サーバ間のセキュリティーを考慮していないが，提案方式では Database Layer へ LDAPS を用いる構成にすることでサーバ間のセキュリティーを高めることが出来る．また，関連研究は既存製品のみで提案・運用されているのに対し，提案方式では Naming Server Layer を構築する必要がある．しかし，既存製品ではクライアントのシステムが対応していないと利用することが出来ないが，提案方式ではクライアントのネーミングサービスに Naming Server Layer が対応することにより，より多くの種別のクライアントが混在した環境においても情報を一元管理することが出来る．

表 6.1: 関連研究との比較

	田中の研究	倉前の研究	提案方式
従来システムからの移行性	×	×	○
データベース間の同期	×	×	○
情報の依存性	×	×	○
スケーラビリティ	×	×	○
共通フォーマットによる情報提供	×	×	○
情報変更のアクセス制御	×	×	○
暗号化の統一	○	○	×
障害時の他システムへの影響	○	×	×
サーバ間のセキュリティー	×	×	○
既存製品のみで構築	○	○	×
対応クライアント種別	少	少	多

6.2 性能

5.5.2 節で従来方式の NIS とキャッシュ機構がない提案方式とのクライアントの応答性能を比較をおこなった結果，従来方式より遅くなった．遅延の要因としては，Naming Server Layer でのフォーマット変換処理やネットワークアクセスの回数が増加することが考えられる．そこで 5.2.3 節，5.2.4 節で遅延の要因を測定し調査した結果，フォーマット変換処理での遅延は小さくネットワークアクセスの回数が増加することが主な遅延の原因として考えれる．つまり 3 層構造の構成を取ることにより Naming Server Layer， Database Layer 間のネットワークアクセスが問題となる．また Directory サーバ固有の問題ではあるが，取得する属性を指定したリクエストと指定しないリクエストでの性能差に問題がある．5.2.6 節の測定結果より Directory サーバに問い合わせをする場合，属性を指定しないリクエストに比べ指定したリクエストは，応答性能が低下するだけでなく，リクエストパケットのサイズが大きくなる．また，Directory サーバに利用しない情報が多数格納されている状態で属性を指定しないリクエストを行った場合，最低限の情報しか格納していない場合に比べ，無駄な情報も多数取得することになり，その結果応答性能が低下することになる．即ち，Directory サーバに 1 ユーザあたり必要最低限の情報のみを登録し，問い合わせには属性を指定しない方法が最も応用性能が高くなる．しかし，複数ネーミングサービスの情報を統合することにより 1 ユーザあたりの情報は増加し，Directory サーバにより多くの属性を設定する必要があり，応答性能を低下させてしまう．これら 2 つの問題を解決する方法して以下の 2 つの方法が考えられる．

1. キャッシュ機構による解決方法

   1 つめのネットワークアクセスによる応答性能の問題は Naming Server Layer に

キャッシュ機構を構築することで応答性能の低下を防ぐことが出来る．5.2.5 節 で Naming Server Layer にキャッシュ機構を設定し Naming Server Layer と Database Layer とのネットワークアクセスを無くした評価システムで測定した結果，2 層構造ネーミングサービスとほぼ同等の応答性能が得られた．これらの結果より Naming Server Layer にキャッシュ機構を構築し，応答性能を低下させる主な原因となっているネットワークアクセスの発生を極力抑えることで，2 層構造ネーミングサービスと同等の応答性能をユーザに提供することが可能である．

また，キャッシュを有効的に利用するために，キャッシュリストに登録した情報の有効時間は 5.3.3 節の測定結果より，ユーザ情報とホスト情報で有効時間を変える必要がある．キャッシュのヒット率を 85 % 以上として考えた場合の，キャッシュの有効時間について表 6.2 に比較結果を示す．複数のネーミングサービスを統合する Naming Server Layer では 5.3.4 節の測定結果よりユーザ情報は 2700 秒，5.3.5 節の測定結果よりホスト情報は 700 - 1000 秒のキャッシュの有効時間を設定する必要がある．つまり，複数システムが混在する環境において共有資源を様々なシステムで統一し集中管理を行うことで 3 層構造ネーミングサービスにおけるキャッシュ機構のヒット率を向上させることが出来る．

表 6.2: キャッシュ有効時間の比較

	有効時間 (秒)
ユーザ情報	2700
ホスト情報 (共有資源の別ホスト名環境)	1000
ホスト情報 (共有資源の同一ホスト名環境)	700

次に，キャッシュ機構がある場合のユーザ情報の応答時間について求める．応答時間については式 (6.1) で示すことが出来る．キャッシュ機構のヒット率を a，キャッシュ機構にヒットした時の応答時間を Th，ミスした時の応答時間を Tm とすると，キャッシュ機構がある場合の応答時間は Tr となる．

$$Tr = aTh + (1 - a)Tm \tag{6.1}$$

式 (6.1) と 5.2.6 節の結果を利用してキャッシュでのヒット率が 85 % の応答時間を求めた場合，NIS クライアントである UNIX からのユーザ情報に対する応答時間は 9.90 ミリ秒となり NIS と比べ 1.03 ミリ秒の差となる．

表 6.3 にヒット率による応答時間の比較結果を示す．比較結果より，Naming Server Layer にキャッシュ機構を設けることで，ヒット率が 85 % の場合，キャッシュ機構がない場合よりも 1.58 倍，ヒット率が 100 % の場合では 1.78 倍の応答性能が向上する．また，現在の評価システムにおいては Naming Server Layer から Database Layer に対して要求があった場合，新たに Directory サーバへのコネクションを生成し検索

を行っているが，Directory サーバへのコネクションの生成を起動時もしくは最初の要求時に生成するのみとし，以後は確立されたコネクションを用いて Directory サーバへアクセスを行うことで，更なる応答速度の向上が見込まれる．このようなコネクション生成の回数を減らすことで Directory サーバをもちいたネーミングサービスにおいて，応答速度が向上するとの報告 [16] がある．

表 6.3: キャッシュヒット率による応答性能比較

ヒット率 (%)	提案方式の応答時間 (ミリ秒)	応答性能向上 (倍)
0	15.68	1
85	9.90	1.58
100	8.88	1.78

2. 利用頻度による Directory サーバ分割方法

2 つめの Directory サーバの属性が増加する場合における応答性能の低下の問題は，利用頻度により Directory サーバを分割することで通常の利用での応答性能を改善することが出来る．図 6.1 に構成を示す．UNIX や Windows といった通常良く使われ

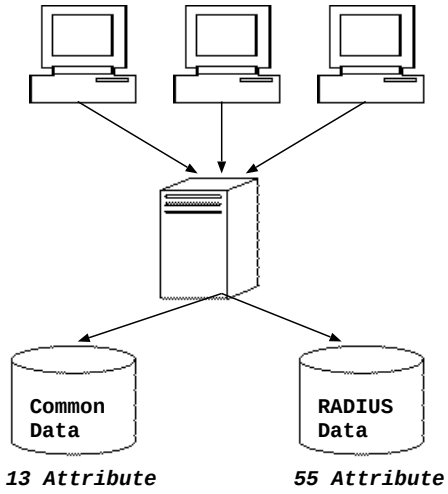

図 6.1: 利用頻度による Directory サーバ分割構成

るシステムのネーミングサービスの情報は頻繁に利用されることが多いが，ダイアルアップ接続等で利用される RADIUS のようなシステムの情報は頻繁に利用されない．5.3.1 節から UNIX のユーザ情報にたいするリクエスト数は 12 時間で 1000 回以上もあるが，ダイアルアップ接続時にしか利用されない RADIUS のリクエスト数は数回程度であると考えられる．また，頻繁に利用される情報には共通する項目が多くあり，NIS と Domain Controller の統合環境においては，2 つのネーミングサービス

で利用可能なユーザに関する情報を 4.4 節で示した 13 属性で管理することが出来る．利用頻度の低い RADIUS のようなシステムでは，そのシステムのみで利用するような情報が多くあり 55 属性が必要となる．

表 6.4: Directory サーバ分割構成における UNIX 利用時の比較

	応答時間 (ミリ秒)	パケットサイズ (Bytes)
単一構成 (属性指定なし)	17.67	1313
単一構成 (属性指定あり)	36.18	490
分割構成	15.88	542

表 6.4 に利用頻度が高い UNIX 利用時の応答時間とパケットサイズについての比較を示す．UNIX, Windows, RADIUS の 3 つのシステムを統合した環境において Direcotory サーバを分割しない場合，5.2.6 節の結果から 17.67 ミリ秒の応答速度となる．それに比べ，利用頻度により Direcory サーバを分割した場合，UNIX, Windows での利用においては利用頻度の高い情報を格納した Directory サーバへ属性値を指定せずにアクセスを行うため，そのときの応答速度は 5.2.6 節の測定結果から 15.88 ミリ秒となる．RADIUS 利用時においては利用頻度の高い情報を格納した Directory サーバと利用頻度の低い情報を格納した Directory サーバにアクセスを行う必要があり，そのときの応答速度は 24.74 ミリ秒となる．このように，利用の少ない RADIUS に関しては応答性能が低くなってしまうが，通常利用の多い UNIX, Windows において応答性能の低下は少なく，UNIX 利用時においては Directory サーバが単一構成の場合に比べ応答性能は 1.11 倍向上する．

また，Solaris 8 以降の機能である NativeLDAP は，ネーミングサービスに Directory サーバを利用することが出来る機能であり， Directory サーバへの問い合わせに，属性を指定したリクエストを行っている．これは，Directory サーバに画像データなどのデータサイズの大きな情報が格納された場合でも，属性を指定することで無駄な情報の取得を無くし，レスポンスのパケットサイズを肥大させないようにしているためである．

これに対して，利用頻度により Directory サーバを分割する方式では，利用頻度が低いと考える画像データなどのデータサイズの大きな情報は，利用頻度の低い情報を格納する Directory サーバに格納することにより，利用頻度の高い情報を格納した Directory サーバへは属性を指定せずにアクセスすることが出来る．従って，分割構成にすることで，NativeLDAP のアクセス方法である Directory サーバが単一構成で属性を指定した場合よりも，応答性能を 2.28 倍向上させることが出来る．

6.3 運用方法

提案方式では，Database Layer に独自のデータベースを利用せずに，どのようなシステムからも利用可能なデータベースとして Directory サーバを選択した．特に従来のネーミングサービスにおいては，独自のデータベースに情報を独自の形式で管理している傾向が強くある．このことは，システムを跨った情報の一元管理を妨げる要因となっている．

例えば，学生情報を管理している学生課，コンピュータの管理を行う情報科学センターの 2 つの管理部門があるとする．従来は，学生に学生課にて氏名やメールアドレスといった情報を，学生課が管理しているデータベースに登録し，後日情報科学センターにてコンピュータで利用するユーザ名や氏名，メールアドレスをネーミングサービスに別途登録する必要がある．これは，複数のデータベースに同一ユーザに関する情報が重複して登録されているだけでなく，管理コストも余分に必要となっている．このような問題を解決するためには，様々なシステムから容易に利用できデータベースで他のシステムからも利用し易い形式で格納しておく必要がある．

本提案方式では Naming Server Layer においてフォーマット変換機能を加えることによりデータベースには特定のシステムに依存しない形式での情報格納を可能としているために，他のシステムから情報を利用し易く，管理部署を跨った情報の一元管理ができる．

Meta-Directory のようなデータベース間で同期をとり，全体としての情報の一元管理を目指すアプローチがあるが，情報は一元管理できるものの全体のデータベースサーバの数は減少しない．しかし，提案方式のアプローチのように様々なシステムから集中管理されたデータベースに対してアクセスを行うことで，全体のデータベースサーバの数を減少することが出来る．組織全体で管理するサーバを集中管理することによって，管理コスト削減と情報の安全性を保つことが出来る．

従って，提案方式のネーミングサービスを用いて情報管理を行うことにより，組織全体での情報の一元管理を，より一層促進することが出来る．

第7章　おわりに

本章では，これまでのまとめを述べた後，今後の課題について述べる．

7.1　まとめ

本研究では様々なシステムが混在する大規模環境で有効なネーミングサービスとして，3 層構造ネーミングサービスを提案・構築し，その有効性を検討した．この提案方式によって，システム種別で別々に管理を行ってきたネーミングサービスを統合することが出来た．また，単純にネーミングサービスを統合するだけでなく，3 層構造にすることによって，データベースにスケーラビリティを持たせることができ，今後，更に増加するであろうと予測される管理情報にも対応することが可能となった．クライアントにたいして特別なソフトウェアを提供するのではなく，ネーミングサービスのサーバにおいて各クライアントに対応することで，ユーザは意識することなく情報が一元管理されたネーミングサービスを利用することが出来る．

6.1 節での関連研究との比較の結果，パスワードがデータベース内で統一することが出来ないことと，障害時の影響の広さの問題点があるが，運用によって回避することができると考える．管理者にとっては，移行性や運用のし易さ等の多くの管理性が向上する．また，情報の格納にオープンな仕様でアクセス可能なデータベースを利用することで，他のシステムとネーミングサービスとの情報の統合が容易に実現することが出来る．即ち，ネーミングサービスの情報を一元管理するだけでなく，組織全体で部署間を跨った情報の一元管理をすることが可能となる．その結果，管理者の負担が軽減するだけでなく，組織内のデータベースサーバも減らすことができ，管理コストの削減が見込まれる．

性能においては，キャッシュ機構を構築することにより，従来までのネーミングサービスと同等の応答性能を得られる結果となった．キャッシュ機構が無い場合に比べ，キャッシュ機構を構築することでヒット率が 85% の場合で応答性能は 1.58 倍向上することを示した．これは，ネーミングサービスで扱う情報は更新頻度が少なく，また複数のシステムが混在する環境においては，それぞれのシステムから同一の情報を参照するケースが多いという管理する情報の特性がキャッシュ機構に適しているためである．また，本提案方式では情報を格納するデータベースとして Directory サーバを対象として評価を行ったが，Directory サーバ固有の問題として属性を指定した検索要求で応答性能が悪くなってしまうという問題があった．更に，Directory サーバに画像データのようなデータサイズの大きな情報を格納した場合，属性を指定しないと無駄な情報を取得することになり，レスポ

ンスのパケットサイズを肥大させてしまう．このような問題にたいして，情報の利用頻度によりデータベースを分割し管理を行う方法を提示し，通常良く利用される UNIX において NativeLDAP と同様のアクセス方式を用いる場合よりも，応答性能が 2.28 倍向上することを示した．

本研究で提案した 3 層構造ネーミングサービスは，ネーミングサービスだけでなく組織全体の情報管理における管理性を向上させ，ユーザにはこれまでと変わらない応答性能を提供することが出来る．これは，大規模環境において様々なシステムが混在する環境が加速しても，容易に各クライアントに対応することが出来るネーミングサービスである．

7.2 今後の課題

本提案方式は Directory サーバを対象として 3 層構造ネーミングサービスを提案したが，Directory サーバにおいて検索要求における問題があったように，他のデータベースを Database Layer に利用した場合においても，データベースに特有の問題が生じる可能性がある．

また，本論文ではある組織内のネーミングサービスについて評価実験を行い評価を行ったが，インターネット上のユーザ情報を一元管理するなどの組織間を跨るような環境が今後増加することが見込まれる．

今後の課題は，様々なデータベースによる評価と，インターネット上において組織間を跨るような環境における評価である．

謝辞

本研究を進めるにあたり，敷田幹文助教授には日頃よりご指導，ご助言いただき心より深く感謝致します．また，多くの助言，励ましを頂いた敷田研究室の皆様に深く感謝すると共に，厚く御礼申し上げます．最後に，学生生活をより豊かなものにしてくれた友人や後輩達に感謝致します．

参考文献

[1] Timothy A.Howes, Mark C.Smith, and Gordon S.Good. *UNDERSTANDING AND DEPLOYING LDAP DIRECTORY SERVICES*. Macmillan Technical Publishing U.S.A, 1998.

[2] Tim Howes and Mark Smith. LDAP インターネットディレクトリアプリケーションプログラミング. ピアソン・エデュケーション, 1997. 松島栄樹，岡薫　訳.

[3] Tom Bialaski and Michael Haines. *Solaris and LDAP Naming Services*. Prentice Hall, 2001.

[4] Hal Stern, Mike Eisler, and Ricardo Labiaga. *NFS & NIS*, Vol. 2. 株式会社 オライリー・ジャパン, 2002. 砂原秀樹　監訳，木下哲也　訳.

[5] Alistair G.Lowe-Norris. 詳説 Active Directory. 株式会社 オライリー・ジャパン, 2001. アクロバイト　監訳，山本浩，イエローレーベル　訳.

[6] 田中哲朗, 安東孝二, 吉岡顕. 複数 OS 環境におけるユーザ管理. 情報処理学会分散システム／インターネット運用技術研究会報告, 99-DSM-16, pp. 49–54, 1999.

[7] 倉前宏行, 島野顕継, 木村彰徳, 松本政秀, 亀島鉱二. ディレクトリサービスを用いた教育用 PC クラスタシステムの学生ユーザアカウント管理. 情報処理学会分散システム/インターネット運用技術シンポジウム 2001, pp. 93–98, 2001.

[8] 倉前宏行, 島野顕継, 木村彰徳, 松本政秀, 古野良樹, 亀島鉱二. 複数 OS 利用のためのユーザアカウントデータベース統合. 情報処理学会分散システム/インターネット運用技術シンポジウム 2002, pp. 63–68, 2002.

[9] Novell DirXML
http://www.novell.com/products/edirectory/dirxml/.

[10] iPlanet Meta Directory 5.0 SP1
http://docs.sun.com/db/coll/S1_ipMetaDir_50sp1?l=ja.

[11] OpenLDAP
http://www.openldap.org/.

[12] iPlanet Directory Server 5.1
http://docs.sun.com/db/coll/S1_pDirectoryServer_51_ja?l=ja.

[13] Samba Users Group Japan
http://www.samba.gr.jp/.

[14] W.Richard Stevens. UNIX ネットワークプログラミング, 第 2 巻. ピアソン・エデュケーション, 2000. 篠田陽一　訳.

[15] Sun Directory Services 3.1
http://docs.sun.com/db/coll/351.1?l=ja.

[16] 羽二生篤. ネーミングサービスにおけるクライアント OS の性能比較. 北陸先端科学技術大学院大学 敷田研究室 副テーマ報告, 2003.