卒業論文　　2001年度(平成13年度)

分散環境における
アプリケーション共有利用システムの構築

指導教員

慶應義塾大学環境情報学部

徳田 英幸

村井 純

楠本 博之

中村 修

南 政樹

慶應義塾大学 環境情報学部

柳原 正

卒業論文要旨　　2001年度(平成13年度)

分散環境における
アプリケーション共有利用システムの構築

本研究では、広域分散環境におけるアプリケーションの共有利用を実現するソフトウェアの提供を第一の目的とする。このシステムが実現する資源処理指向型システムの考察を第二の目的とする。

近年のコンピューティング環境においては、高速ネットワークから多種多様なデータを様々なアプリケーションによる利用が可能となった。さらに、これらのアプリケーションは仕様の異なる機器へダウンロードし、実行が可能となった。ユーザはこのような環境において、アプリケーションの利用とデータの入手が容易となったが、同時に入手可能なデータとアプリケーションの種類や量が急増したため、有害なものも同時に入手しやすくなった。また、機器の仕様が異なるヘテロジニアスなコンピューティング環境が現実的となった現在において、仕様の異なる機器間における協調作業が困難となりつつある。

本研究では、上述した問題点を解決するため、ネットワーク上にデータとアプリケーションを共有し、アプリケーションとデータの柔軟な組合わせを提供するフレームワークの構築を行う。本研究において、このフレームワークを実現するプロトタイプシステム RaDiuS を構築した。

本論文では、まず近年におけるコンピューティング環境内のアプリケーションとデータの利用における問題点を指摘し、これらを解決する既存の分散システムの適応で解決が可能かを検証する。次に、アプリケーション及びデータの共有を実現可能とする RaDiuS を取り上げ、概要、設計及び実装方法を示す。最後に定量的及び定性的評価をもとに資源処理指向型分散システムの有用性を実証する。

慶應義塾大学 環境情報学部
柳原 正

Abstract of Bachelor's Thesis

Implementing an Application Sharing System Within Distributed Environments

In this paper, we will be focusing on two subjects; providing a framework for sharing applications within a distributed environment, and the realization of Resource-Application Oriented Systems.

Everyday computing allows users to use various forms of data on numbers of applications. Such computing provide users with devices which can download and execute applications, dispite of their differences in specifications. However, the ability to gain access to so much data also means facing harmful data more often. Also, the distribution of heterogeneous devices in computing environment means other devices will have more trouble to cooperate with one another in such an environment.

This paper proposes a solution to the problems mentioned above. We propose a framework where application and data sharing is made easy, enabling applications and data to combine with each other more easily. We have implemented a prototype of such framework, named "RaDiuS". With RaDiuS, users can combine applications and data with each other with ease.

First, we point out the problems when using applications with data in the computing environment today, and verify whether existing distributed systems are sufficient enough to become a solution. Next, we will be explaining about the summary, design, and implementation of RaDiuS as an answer to the problem. Finally, we will prove the possibilites RaDiuS has as we look at the qualitative and quantative analisys of RaDius itself.

Tadashi Yanagihara

Faculty of Environmental Information
Keio University

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　本研究の背景

近年のコンピュータは高速化すると同時に入手する値段が安くなり、入手が容易となっている。また、携帯電話のような携帯機器に内蔵させるものも登場することで、小型化に成功をしている。また、これらの機器に搭載されるソフトウェアはネットワークを介して入れ替えが自由に行えるようになった。例えばNTTDocomo社によって開発された携帯電話にはJava言語 [1] のサブセットであるKJavaを用いて作られたアプリケーションの入れ替えが可能である。このように、ユーザは利用可能な機器が多種多様となり、計算処理能力や仕様の異なるヘテロジニアスなコンピューティング環境が日常的に利用が容易となった。

同時に、家庭内ネットワークが普及し、ネットワークを構築するための機材が安価に入手が可能となった。例えば家庭内ネットワークの無線化を実現するIEEE 802.11b[2]の一般市販化、また高速ネットワークを低価格で提供するADSL [3]、CATV [4]、FDDI [5] の普及によって家庭内LANにおける常時接続環境を容易に構築できるようになった。この常時接続環境によってユーザは今まで以上に情報及びデータを入手することが可能になった。例えばインターネット内の友人と一対一の電子会話が行えるICQ[6]、または音楽ファイルの公開及び入手の空間を提供するmp3.com[7] のサービスが無料で利用できる。さらにインターネット内のデータの検索を提供するgoogle[8] では画像のように、種類別のデータによる検索サービスを提供している。このようにユーザはインターネットから人に関する情報、音楽や画像のようなデータ、またはファイルの交換などを行うための空間という**資源**を入手することができる。

以上で述べた環境においてユーザは多種な資源をネットワークから容易に入手が可能となり、これらを多様な機器で利用することが可能となった。しかし、これらの要素によって資源と機器との結び付きの複雑化、ヘテロジニアスな計算環境における機器間作業の困難化、有害な資源の入手経路の容易化、という三つの問題点が発生した。

資源同士の結び付きの複雑化

高速ネットワークの登場によって資源の種類が豊富になり、さらに入手する量も増大した。しかし、これによって資源と機器、より具体的には機器上の資源同士による組合わせが多くなった。例えば新しい形式の画像の画像を入手すると形式に対応したビューアのようなアプリケーションが必要となる。または、新しいアプリケーションを入手し、それを利用するためのデータが必要となった場合にそれを容易に入手でき

るようにしなければならない。
　このように、アプリケーションとデータといった資源同士の組合わせを構築するために資源の入手を容易とすることが望ましい。

ヘテロジニアスな計算機器における機器間作業の困難化

　近年の携帯機器は既存の IC チップベースの機器に比べ、容易にアプリケーションの入れ替えが行えるようになった。しかし、同時に実行可能なアプリケーションの種類が限定されるようになった。例えば携帯電話に搭載された Java 言語ベースのアプリケーションは機器に搭載された JavaVM の互換性の問題で利用することができない。あるいは、機器の計算処理能力において特定の作業を実行することが困難な場合が多い。圧縮ファイルの解凍で多くの計算資源を消費するようであれば一度計算資源の豊富な機器へ転送して解凍する方が良い場合もある。このように、機器間における計算処理能力及び仕様の差異を吸収する仕組みを提供することが望ましい。

多量な資源から有益なものの選出の困難化

　資源の入手が容易となることで有益な資源だけでなく、有害な資源の入手も容易となってしまった。例えば電子メールによってウィルスという有害な資源の入手が容易となったことが挙げられる。このように、ユーザから有益な資源の入手を妨げることなく、有害な資源から守る必要性がある。

1.2　本研究の目的

　本研究において、現在のインターネットを初めとした分散システムにおいて、各ホストが持つアプリケーション及びデータ、空間、人、ものといった資源を他ホストから利用することを実現したシステムの提供を目的とする。
　ユーザは本システムを用いることでネットワーク上に分散された資源の入手、またそれらを有効活用するためのアプリケーションを発見・利用することができる。これを元に他ホストを利用することで処理結果を得たり、特定の資源だけ他ホストへ転送し利用することで仮想的にローカルホスト上で利用したときと同様の結果を得たり、今まで利用することができなかった資源を新しく発見したアプリケーションによって有効活用することが可能となる。これによってネットワーク内で共有された資源を新規追加することなく有効活用が可能となる。

1.3　本論文の構成

　本論文は、以下の構成によって成る。

第 2 章では、既存のネットワーク利用形態の問題点を把握し、これを解決するネットワークモデルの考察を行い、適格なネットワークモデルを提案する。

第 3 章では、第 2 章で述べたネットワークモデルの機能要件を述べ、その実装例である RaDiuS とその特徴について述べる。

第 4 章では、RaDiuS の設計方針、全体構成、各サブシステムの設計について述べる。

第 5 章では、第 2 章で述べたネットワークモデルのプロトタイプである RaDiuS の実装環境、各サブシステムの実装の詳細について述べる。また、RaDiuS を用いた応用例についても触れる。

第 6 章では、実装した RaDiuS のプロトタイプシステムの定量的評価と評価結果の考察、既存システムとの性能比較を行う。

第 7 章では、全体としての結論について述べ、また今後の課題を明らかにし、本論文を締めくくる。

第2章　分散システム

2.1　本システムの機能要件

前章で述べたように、インターネットのような分散環境内の資源とユーザが利用するアプリケーションの組合わせを有効利用するために、資源と機器との結び付きが複雑になることで新種のデータを利用するためのアプリケーションの入手が必然となった。また、ヘテロジニアスな計算環境において、既存の機器との協調作業が困難となった。さらに、常時接続ネットワークの普及によってユーザにとって有害な資源が入手しやすくなってしまった。これらの問題点を解決するシステムを構築する上で必要な機能要件として分散ホスト発見機能、アプリケーション移送機能、改竄対策機能が挙げられる。

分散ホスト発見機能

現時のネットワークの利用形態として分散環境内でネットワークの規模及びネットワーク内のホストの存在の有無が不特定な分散環境で利用されるものが想定される。このため、本システム内で取り扱う資源でもあるアプリケーションはホストと結合させてはならない。これはホストそのものではなく、ホスト上のアプリケーションを元に検索を行うことを意味する。

このため、本システムを構築する上ではホストではなくアプリケーションを元に検索を行う。

アプリケーション移送機能

本システムで取り扱われる資源としては処理を行うアプリケーションと、処理が行われるデータが挙げられる。過去の分散並列処理モデルではデータはアプリケーションのあるホストへ転送され、処理されたのち、結果が返ってくるモデルとなっていた。しかし、このモデルで処理するデータの量が多くなるにつれ、不適格なモデルとなってしまう。

本システムでは状況に応じて、分散処理モデルのように処理ホストへデータを転送する手法以外に、アプリケーションをデータの格納されたホストにダウンロードしたのちに実行を可能とするモデルを提案する。これによってデータ及びアプリケーションの容量に関係なく、最適な処理方法を選ぶことが可能となる。

改竄対策機能

データを処理し結果を返される場合、またはアプリケーションをダウンロードし実行する場合において、処理を行うアプリケーションが改竄された場合の対処方法を考えなければならない。これには Peer-to-Peer モデルの特徴でもある冗長性を持って解決する。

Peer-to-Peer モデルのように複数のホストへ同じような内容を送り、結果を返してもらった後に同じ内容の結果を見て比較を行う。ここで同じような内容が多いものを正当性が高いものと判断する。

以上のように、機能要件を全て満たすシステムとして既存のヘテロジニアスなホストにおける並列処理を可能とする分散処理システムと、インターネット上のファイルを共有し、他ホストへのダウンロードを可能とする Peer-to-Peer システムの検証を行う。これはアプリケーションの有効利用のために分散環境の資源確保を行う分散処理システムと、資源を有効活用することを目的としたアプリケーション集である Peer-to-Peer システムという両側からの観点を持って問題を検証することが容易となる。

2.2 分散処理システム

IBM が世界で初めてのマルチプロセッサで動作するコンピュータを完成して以来、分散処理の原型である「並列処理」という新しい研究分野が開拓され、高速な CPU を数多く搭載したコンピュータが登場した。CPU 同士が通信し合い、処理の負荷を分散させるという並列処理を実現した原始的なシステムである。

しかし、先ほど述べたコンピュータに複数の CPU を搭載する研究が盛んに行われるようになったが、後に Ethernet という新しい技術の登場によってコンピュータ同士が通信することが可能となり、プリンタやデータなどといった資源の共有が可能となった。

このようなシステムではローカルホスト上に存在するアプリケーションを実行するために分散環境内における資源を有効活用するために構築されたシステムである。このようにアプリケーションのために資源が存在する理念によって構築されたシステムとも言える。このようなシステムにおいて、ヘテロジニアスな計算処理能力を持った機器の混在を許可したものをとして PVM[9]、 PopularPower[10]、Javelin[11]、Ninflet[12]、MetaSpace[13] を取り上げる。

PVM

Parallel Virtual Computing(以降、PVM) はネットワークに接続された UNIX ワークステーション上で並列処理を行うためのライブラリ及びコマンドである。モデルとしては pvmd というデーモンが稼動しているサーバと pvm というアプリケーションが稼動しているクライアントでシステムを構築する。クライアントから送られてきたアプリケーションはサーバで受け取られ、処理した結果が返される。

但し、PVM には以下のような欠点を持つ。

- ライブラリのマルチプラットホーム性を備えていないため、多様な開発言語で書かれたアプリケーションや実行環境を搭載した機器上での利用には向かない。
- PVM 上ではセキュリティ機構を備えていないため、不特定多数のホストが参加するような大規模ネットワーク内での利用に向かない。

このため、ヘテロジニアスな機器が混在した広域ネットワークが一般となった現在のユーザの利用形状に適応できない。

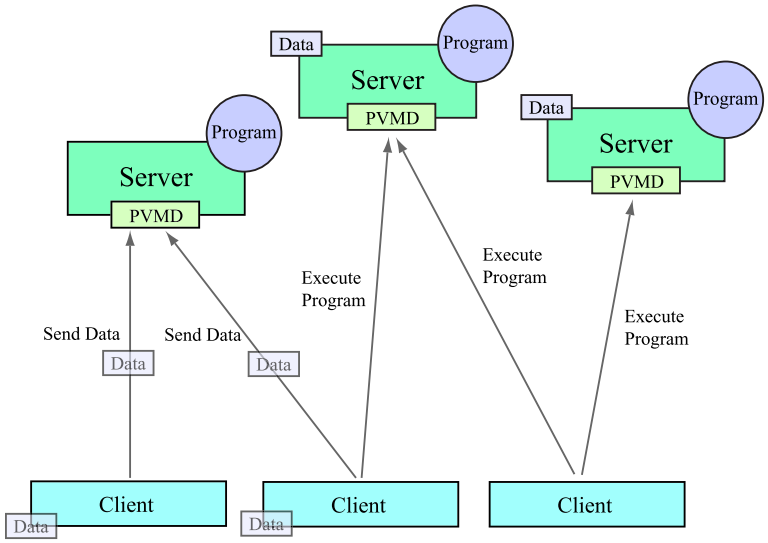

図 2.1: PVM の動作図

PopularPower

PopularPower は世界中の遊休状態のホストを有効利用してビジネス化を計ろうとしたプロジェクトである。ユーザがダウンロードしたクライアントはユーザがホストを利用している間は何もしないが、アイドル状態になったときに PopularPower サーバからインフルエンザの DNA 構造といったデータ内容をダウンロードし、解析を行った結果をサーバへ送り返す。後にホストの計算資源を利用する代わりに利用時間に応じた金額がユーザへ送られると言ったサーバクライアントモデルである。

しかし、このシステムではサーバが停止するとクライアントが機能しなくなるため、耐故障性に弱いことが挙げられる。

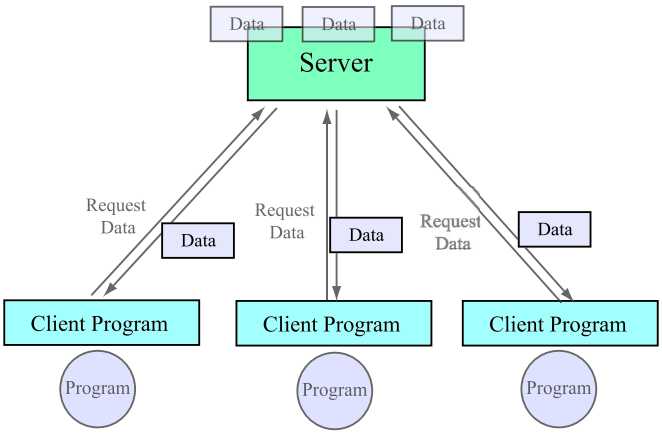

図 2.2: PopularPower の動作図

Javelin

Javelin は Minisoft 社が提供するサーバクライアント型のソフトウェアである。マシン上のアプリケーションを www サーバにアップロードし、Javelin を利用することで他ユーザはアプレットを通して www サーバ上のアプリケーションを利用する事が可能となる。Java VM が提供するセキュリティ機構に基づいたアプレットを www サーバ内の SSL 暗号化機能と連携させることで、大規模ネットワーク内での利用に向いている。さらにアプリケーションと実行するための環境がサーバ側にあるため、アプリケーションは利用される機器側で利用可能な言語及び実行環境に依存し無くて済む。

しかし Javelin は処理中であれば常にユーザインタフェースであるアプレットを起動していなければならない。これはネットワークに常時接続された機器であれば問題は起こりにくいが、既存の機器のようにネットワークへの参加・退去が不特定に行われる利用形状には向かない。

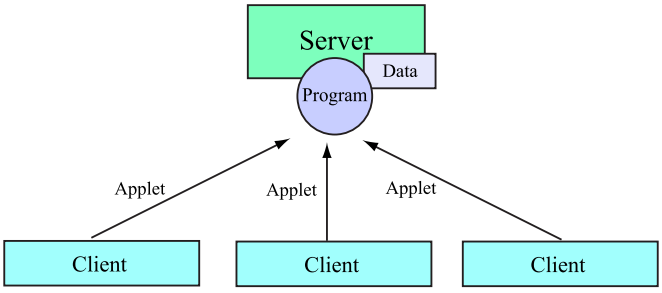

図 2.3: Javelin の動作図

Ninflet

産業技術総合研究所 (旧・電子技術総合研究所) の高木浩光氏によって実装されたソフトウェアが Ninflet である。世界中の遊休計算機の計算パワーを世界規模で共有及び共同利用を可能としたシステムであり、Java のセキュアでオープンプラットホームな特徴を活かしている。モデルとしては処理されるプログラムを格納したクライアント、プログラムの処理を行うサーバ、そしてクライアントのプログラムをサーバに送信する際に割り振りを行う dispatcher サーバがある。

クライアントから一度処理を行うプログラムが dispatcher サーバへ送信されると、dispatcher サーバからは他サーバから処理に必要な Java Class ファイルをダウンロードした後、処理を行うサーバへプログラム及び Java Class ファイルを送信する。

しかし、このときプログラムの実行を行うサーバが dispatcher サーバへ接続する際、自分の資源を共有するレベルの細かい調整をすることができない欠点を持つ。

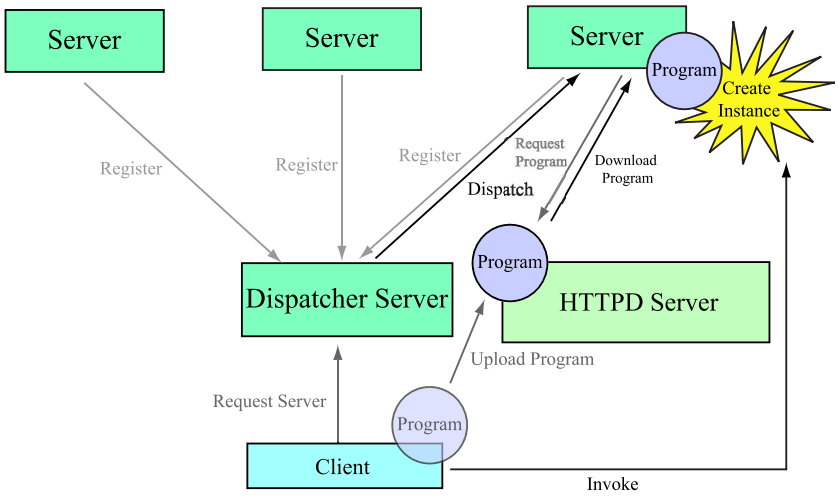

図 2.4: Ninflet の動作図

MetaSpace

MetaSpace は慶應義塾大学の藤村浩氏によって実装されたプロクシ型ソフトウェアである。特徴として処理中のアプリケーションを他ホストへ動的に移送が可能であることが挙げられる。これによってホストの故障によって処理が停止してしまった場合、他ホストへ移動し処理を再開することが可能となり、またこのときに移送先のホストに処理の最適化を行う事が可能となる。

システムは Ninflet とは近似しているため、モデルは同種のプロキシ型システムとなる。さらに Ninflet の欠点であったホスト別資源共有の調整を Ticket というような識別タグを利用することで実現している。これによってクライアントが必要とする計算資

源とサーバが提供可能な計算資源を柔軟に設定することが可能なため、ヘテロジニアスな計算処理能力を持ったホストの多様化にも対応できる。

しかし、実装ではプロキシが単一の場合を元に作成されたため、プロキシが停止してしまうとシステム全体が機能停止するという欠点を持つ。

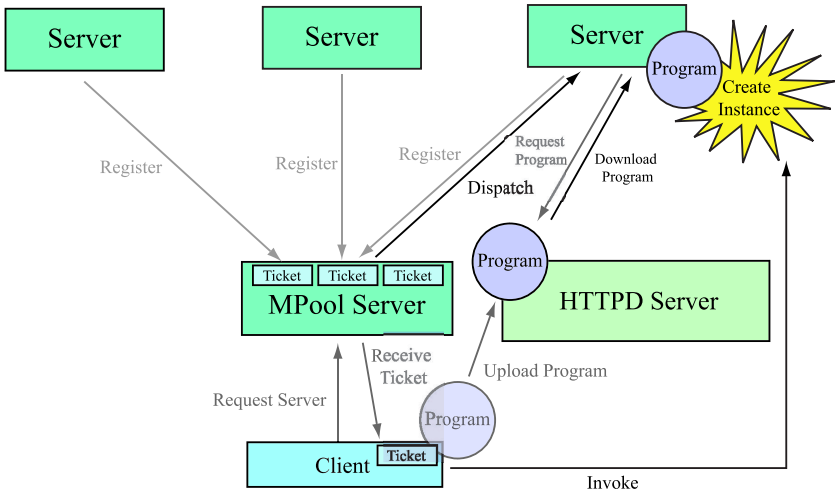

図 2.5: MetaSpace の動作図

2.3 Peer-to-Peer システム

今までの分散システムではプログラムの分散環境内での利用を考慮し、アプリケーションの実行を行うために分散環境内における資源 (人、空間、時間、もの) を検出し、それらを有効利用するために作られたシステムであった。一方、近年において今までのアプリケーションを中心に考えられていたシステムの思想とは逆に、分散環境内における資源を有効活用するためのアプリケーション及びフレームワークを提供するシステムが登場した。このようなシステムの代表例として Peer-to-Peer システムを取り上げる。

Peer-to-Peer システムは分散された資源 (人、空間、もの) を有効利用するためのネットワークである。ネットワーク自身の特徴としては、ネットワーク内の各ノードは自律的に動かなければならない。例えば DNS のような中枢管理機構に依存してはならず、またネットワーク内では資源とノード (ホスト) の間に依存関係があってはならない。つまり資源が任意のノード上にあってもよいモデルである。

このモデルによって検索の対象をホスト名や IP アドレスに特定する必要がない。例えば特定のホストを使用しているユーザやホスト上に存在するドキュメント及びサービスを検索の対象に行うことが可能である。これはホスト全てが同じ機能を提供するため、検索対象となっている資源がどのノード上にあるかを意識する必要がない。

以下に Peer-to-Peer システムの検索メカニズム及びホストの資源共有メカニズムを

検証するために以下の三つの既存システムを取り上げる。

Napster

Napster [14] は現在の社会に影響を与える存在となった Peer-to-Peer システムとして挙げることができる。Napster システムではサーバとクライアントが完全に分離したソフトウェアから成る。ユーザはクライアントをダウンロードし、実行することで Napster 社が管理する Napster サーバに接続される。このときクライアント上で共有されている音楽ファイルのデータベースがサーバ上に生成される。クライアントがファイルの検索を行うと、サーバを介することで検索対象となった音楽ファイルを発見することができる。図 2.6

しかし、このモデルの問題点としてはサーバが何らかの原因で停止してしまった場合、クライアントが稼動できないという耐故障性に弱い点が挙げられる。

2001 年 7 月を持って Napster サーバ上でファイルの共有を一切禁止され、結果として Napster ネットワークも同時に停止されることとなったが、現在ではこの Napster サーバの代わりに Napster プロトコルと互換性を持った OpenNapster プロトコルで通信を行う OpenNapster サーバが登場した。現在利用されている多くの Napster クライアントのフロントエンドはこの OpenNapster サーバと通信を行うことで今まで Napster ネットワークで利用されていたサービスを提供している。

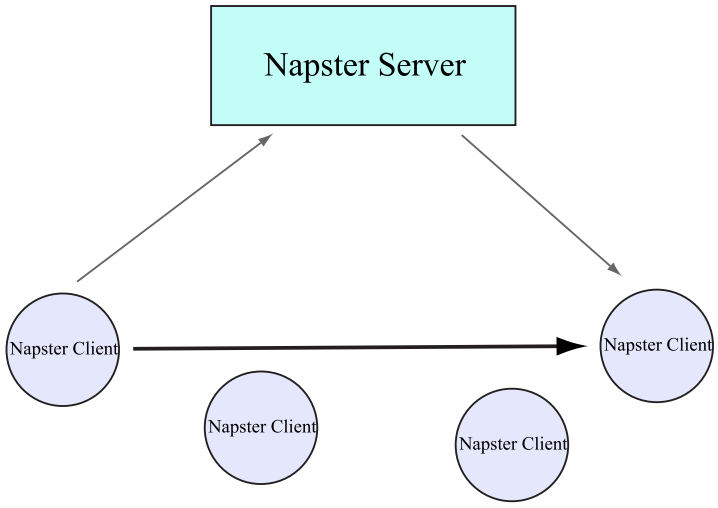

図 2.6: Napster Protocol の動作図

Gnutella

Gnutella [15] は Napster と同時期に発足した Peer-to-Peer システムの典型モデルとして挙げられる。但し、Napster とは異なり、Gnutella ではサーバ・クライアントのよう

にソフトウェアの分離がない特徴を持つ。これは Gnutella ネットワーク内では全てのノードはサーバであると同時にクライアントでもあるという Servent という概念を元に作られているためである。Gnutella クライアントがネットワークに接続した時点でコンテンツのダウンロード及び検索を行う Gnutella サーバとしての役割も果たすことになる。このため、サーバの機能停止によってクライアントの機能停止を引き起こさなくなり、耐故障性に優れたシステムとなる。

検索方法はブロードキャストアドレスを介し近くのノードを発見するモデルを利用する。図 2.7 のように、特定のノードが一度の通信で多くの他ホストの存在を発見するメリットがある反面、相対的なトラフィック量が過去のサーバ・クライアントモデルよりも多くなってしまう。また、実質では常に上がっているノードへ最初に接続し、ネットワークに介入している他ノードの情報を入手する host cacheing という方法が多くの Gnutella のフロントエンドに実装されているが、この host cacheing を行うホストが停止した時点で途端に他ノードの発見が著しく効率悪くなる。

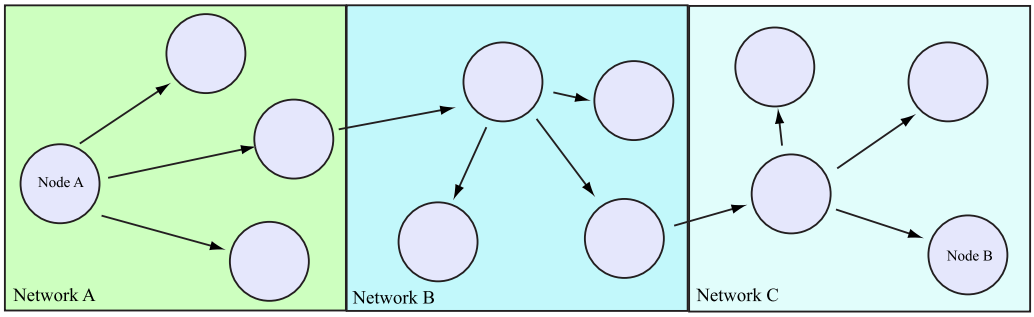

図 2.7: Gnutella Protocol の動作図

Freenet

Freenet [16] は Gnutella との近似システムであり、異なる点としてはブロードキャストアドレスを利用せず、ノードそれぞれで決められた評価基準に基づいて次のどのノードに渡すかが決まる。例えば過去に何度かデータが格納された履歴を持つノードは他ノードから高い評価を得ているため、他ノードからの通信が多いこととなる。また、ノードは一度検索していたファイルを発見すると、データを検査元のノードへ転送する際に途中経過するノード内にファイルのコピーを置いて行く。これによってリクエストの高いファイルが置かれるノードが格段と増える。図 2.8

他に特徴としては Freenet 内で扱われるデータにはそれぞれ固有の鍵が添付される。システムの利用者はこの鍵を元にデータの検索を行う。各ノード内に格納されたデータには鍵が作成され、さらにデータを格納するノードの IP アドレスと結合させ、さらにもう一つの鍵が作成される。このホストとホストに格納されたデータが結合された鍵を元にユーザは検索を行う。

しかし、このシステムでは以下の問題点を持つ。

- Freenet ノードはあらかじめ検索依頼を渡す次のホストの存在を知っていなければならない。
- 検索は鍵ベースとなるため、ユーザはあらかじめデータ固有の鍵を理解していなければらない。

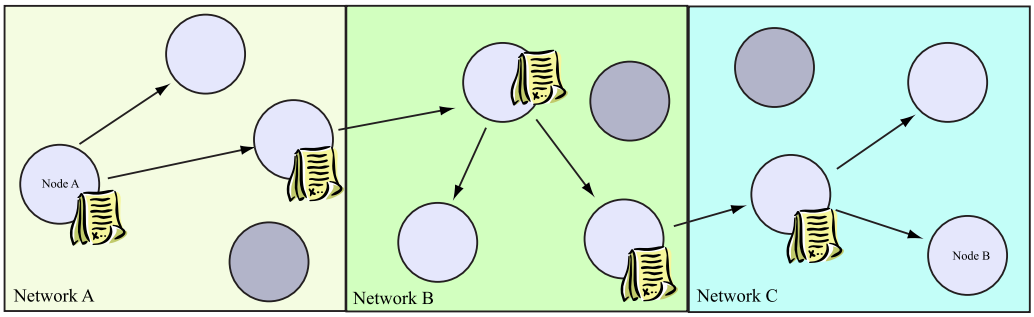

図 2.8: Freenet Protocol の動作図

2.4 分析結果

分散処理システムの性能表を表 2.1 に、Peer-to-Peer システムの分析結果を表 2.2 とを示す。

まず各分散処理システムにおいては分散ホスト発見機能を分析してみると、全てのシステムにおいて検索を行うための機能を備えていないことが分かる。PVM と Javelin では直接クライアントから処理用サーバをユニキャストでアドレスを指定しなければらならない。PopularPower では逆にクライアントがサーバへ接続しにいくため、処理を行うホストの発見を行う機能が不要である。Ninflets と Metaspace では処理を依頼するクライアントはプロキシサーバへ一度存在を通知し、処理内容を登録する。このプロキシサーバへはユニキャストによって行われるため、分散ホストの検索は自律的に行えないことが指摘できる。これは特定のホストに依存していたために、そのホストが機能停止したときにクライアントが機能できなくなる耐故障性の面において望ましくないと言える。

次にアプリケーション移送機能を検証する。PVM、PopularPower、Javelin ではホストに移送されるのはそれぞれのローカルホスト上に常駐しているアプリケーションが利用するデータのみであり、アプリケーション自身がホスト間において移動することができない。Ninflet と Metaspace ではアプリケーション移送を許可するように構築されている。

最後に改竄対策機構として主にアプリケーションの改竄対策に注目する。PVM、PopularPower、Javelin ではアプリケーションはホストから移動することがないため、ローカルホスト上で実行し続ける限りでは改竄の検証を行う必要がない。このため、これ

らのシステムでは改竄対策機能を備えていない。一方、Javelin、Ninflet、MetaSpace では大規模ネットワーク上で利用することを前提としているため、セキュリティ的に厳しい対策を取っている。これらのシステムでは利用するアプリケーションの検証を行い、改竄の疑いがあれば実行を中止する機能を持つ。

システム名	PVM	PopularPower	Javelin	Ninflet	MetaSpace
モデルタイプ	ServerClient	ServerClient	ServerClient	Proxy	Proxy
分散ホスト発見機能	×	×	×	×	×
アプリケーション移送機能	×	×	×	○	○
改竄対策機能	×	×	○	○	○

表 2.1: 既存の分散処理システムの性能表

次に Peer-to-Peer システムの分析を行う。全てのシステムは分散ホストの発見を目的としているため、発見機能を備えている。逆にファイル共有を目的としているため、アプリケーションの実行を行うことがない。このため、アプリケーションの移送機能を行う必要がない。改竄対策の分析においては、それぞれのホスト上で共有されたファイルの正当性を検証するという意味で改竄対策機能を備える必要があるかもしれないが、現時点では Napster 及び Gnutella にはこのような機能を備えていない。一方 Freenet では検索はファイルを暗号化した鍵を中心に検索をするメカニズムを取っているが、この仕組みのおかげでファイルが改竄された場合にファイルの名前である鍵が変わるため、改竄対策機能は備えている。

システム名	Napster	Gnutella	Freenet
モデルタイプ	Proxy	Servent	Servent
分散ホスト発見機能	○	○	○
アプリケーション移送機能	×	×	×
改竄対策機能	×	×	○

表 2.2: 既存の Peer-to-Peer システムの分析

2.5 本章のまとめ

本章を通して、一章で述べた問題点として既存の分散処理システム及び Peer-to-Peer システムの検証を行った結果、どちらのシステムが条件全てを満たさないことを検証した。分散処理システムでは処理における分散ホストの発見を行うためのメカニズムに欠け、Peer-to-Peer システムでは移送機能に欠けることが分かった。本来これらのシステムはこのような用途のために利用されることを想定して作られていなかったと考えられるが、三つの問題を解決するモデルとなれる。次章ではこれらの機能要件を満たすシステムである RaDiuS の設計について述べる。

第3章　資源処理指向型分散システム

3.1　資源処理指向型分散システムの定義

前章では既存の分散処理システム及び Peer-to-Peer システムでは一章で述べた機能要件を全て満たすことができないことを検証した。分散処理システムではアプリケーションを利用するための資源発見機能が提供されておらず、Peer-to-Peer システムでは発見した資源の利用方法を考慮して構築せず、機能要件を満たすことができない。

本システムを構築する上で、分散処理システムのように処理を行うアプリケーション及び分散された資源の有効活用を考慮した**資源処理指向型分散システム**としてのシステムモデルを提唱する。資源処理指向型分散システムでは分散処理システムが持つアプリケーション移送機能と Peer-to-Peer システムの分散ホスト発見機能のそれらの機能を持ち、これによって両システムが持たない欠点を補い合うことが可能となる。なお、セキュリティ機能は既存の分散処理システム及び Peer-to-Peer システムで利用されていたものを適応するには機能不足となることが予想されるため、セキュリティ機構は独自による実装を行われた方が好ましい。

3.2　資源処理指向型分散システム：RaDiuS

本研究において資源処理指向型分散システムのプロトタイプとして RaDiuS (**R**esource **A**pplication oriented **DI**strib**U**ted **S**ystem) を実装した。RaDiuS は分散環境内における計算を行うための計算資源及び情報資源の有効活用を目的としたシステムである。

RaDiuS には前章で述べた分散ホスト発見機能、アプリケーション移送機能、改竄対策機能を特徴としたシステムである。これらの機能を実現するにあたり、以下で述べる注意点に注意しながら設計を行う。

3.2.1　分散ホスト発見機構

ホストと資源との結合

分散環境内でネットワークの規模及びネットワーク内のホストの存在の有無が不特定な分散環境で利用されるものを想定している。このため、本システム内で取り扱う資源でもあるアプリケーションはホストと結合させてはならない。これは検索をホストではなく、アプリケーションを元に行うことを意味する。このように、分散環境内においてはホストではなくアプリケーションを元に検索を行うシステムとして構築す

ることを提案する。

特定のホストへの依存

RaDiuS のようなシステムではアプリケーションルーティングを元にノード間における通信を作成している。これには他ノードに関する情報を入手し、独自でルーティングテーブルを生成するわけだが、ここで他ノードの情報の入手方法に留意をしなければならない。それは特定のホストへの強い依存を持ってはならないことである。

これはルーティングテーブルの中に必ず通らなければならないノードが存在する場合に、そのノードがネットワーク内から切断したり故障で機能停止した場合に通信が行えなくなるからである。ルーティングが行えなくなることは同時にノードの機能停止し、結果としてネットワーク全体の停止を引き起こす。このように、分散環境内においては特定のホストへルーティング情報を依存することがあってはならないのである。この条件を満たす分散システムへの適応として Peer-to-Peer システムの採用を提案する。

全体的な通信量

Peer-to-Peer システムで特に議論となる議題としてシステム自身が生成する通信量がどのようにしてネットワークに影響を与えるかである。例えば Gnutella ではネットワーク内のノードの情報を得るためにブロードキャストと通信を行う。情報の収集が素早いというメリットを持つが、同時に通信帯域を多く消費してしまいかねない。

このように、Peer-to-Peer システムの多くが採用する集中管理サーバのような機構がないシステムを構築するときに全体的な通信と検索速度のトレードオフについて考えなければならない。

3.2.2 アプリケーション移送機構

アプリケーションとデータの状況別移送

本システムで取り扱われる資源として処理を行うアプリケーションと処理が行われるデータが挙げられる。過去の分散並列処理モデルではデータはアプリケーションのあるホストへ転送され、処理されたのち、結果が返ってくるモデルとなっていた。しかし、このモデルで例えばデータの量が多くなるにつれ不適格なモデルとなってしまう。

また、共有したアプリケーションのうちにマシンアーキテクチャー依存なものがあった場合にでも、アプリケーションを移送せずに実行するための環境を備えたローカルホスト上で実行することができる。

また、本システムでは状況に応じて、分散処理モデルのように処理ホストへデータを転送する場合以外に、アプリケーションをデータが格納されたホストにダウンロードしたのちに実行を可能とするモデルの採用を提案する。これによって例えデータ及

びアプリケーションの容量に関係なく、最適な処理方法を選ぶことが可能となる。

また、本システムを構築する上でアプリケーションのノード間の行き来を自由とするシステムモデルが必要である。これにはオブジェクトの送受信それぞれの機能を持つ**サーバント**としてそれぞれのノードを自律的に動作させることで実現が容易となる。

以下にそれぞれの機能を備えたアプリケーション移送機構の動作を表した図 3.1 を示す。

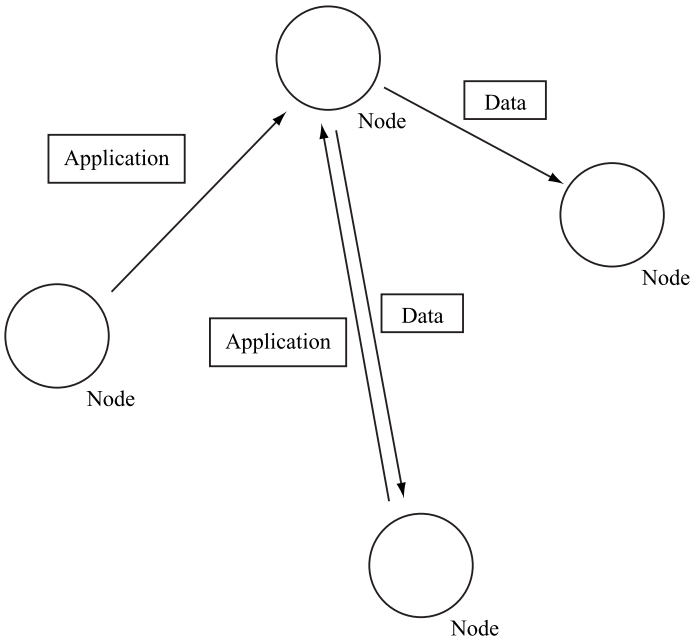

図 3.1: 他ノードとのアプリケーション及びデータの移送状況を表した図

3.2.3 改竄対策機構

アプリケーションの改竄への対策

データを処理し結果を返される場合、またはアプリケーションをダウンロードし実行する場合において、処理を行うアプリケーションが改竄された場合の対処方法を考えなければならない。この問題点に対する解法として以下に述べる。

一つの解法としては各アプリケーション別に署名をし、正当性を証明する方法が挙げられる。しかし、署名における問題としては署名の正当性を確かめる機構が必要となる。これは分散システム内で調べる場合においてはサーバクライアントモデルのように証明の集中管理機構がない限りはほぼ不可能な作業となる。

もう一つの解法としてはアプリケーション実行時にコードの実行範囲を制約する方法が挙げられる。これは例えば Java 言語で言うアプレットのように利用可能な機能にかかる制約が厳しくなるが、制約を個別で設定することも可能となっている。しかし、この方法を実現するためには処理中のプロセスが行う動作のうち、アプリケーション

が本来行うべきでない処理を検出するようにしなければならないため、アプリケーションの種類が多い環境では対応が困難となる。

最後の解法としては図 3.2 のように、アプリケーションの指紋 (MessageDigest) を採取し、実行前に同じ値のものを正当性が高いものとして判断し利用することができる。一見単純なやり方ではあるが、改竄されたアプリケーションの実行予防に対しては効果はある。RaDiuS では以下の作業過程を経て実行を行う。

1. 必要とするアプリケーションを検索をし、他ノードへ検索内容を送る。
2. 結果は Message Digest という形で返す。
3. 結果が同じなものを正当性の高いものとして判断する。

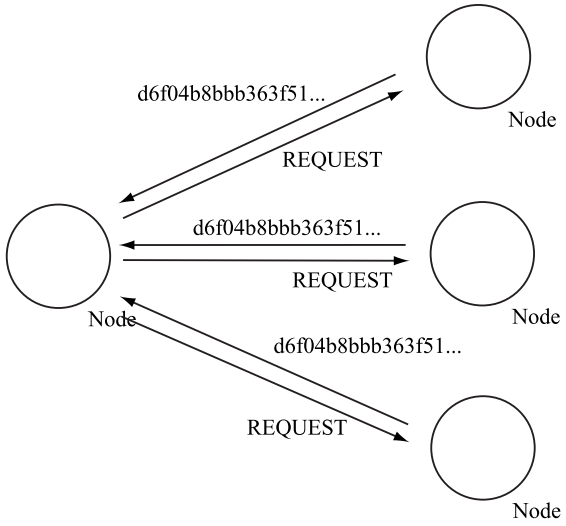

図 3.2: 他ノードとの MessageDigest を表した図

3.3 RaDiuS のシステム全体動作

RaDiuS のシステムの全体動作図を図 3.3 に表す。図 3.3 のように、ノードはそれぞれ自律的に動作し、他ノードからアプリケーションやデータをダウンロードしたり、他ノードへアップロードを行うことが起こるシステムである。この際に他ノードが持つ資源の検索を行ったり、あるいはネットワークに新規接続したノードからの要求に応じて返事を返す動作を行う。

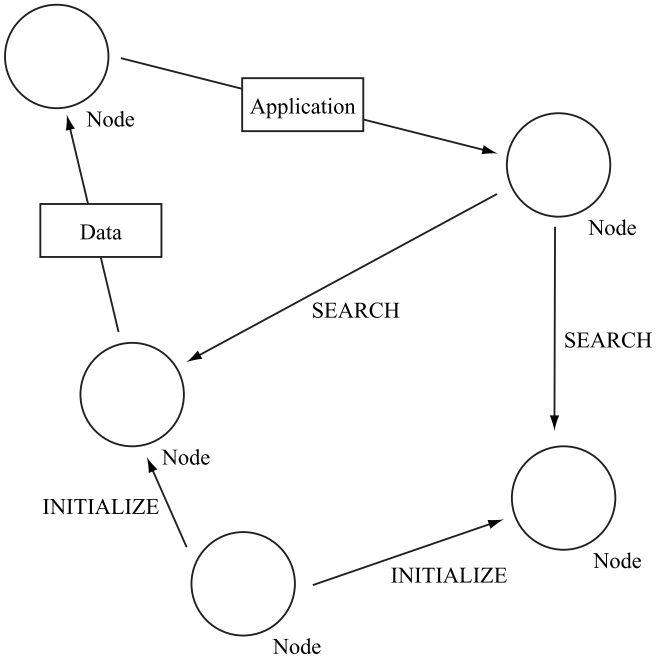

図 3.3: RaDiuS の全体動作図

3.4 本章のまとめ

本章では資源処理指向型 RaDiuS が提供する機能とそれらを実現するために必要とする機能要件について述べた。RaDiuS は様々なアプリケーションの混在を許可したシステムであるが、上述したように、アプリケーション及びデータの移送が特徴である。また、資源の発見が行え、直接アプリケーションと結び付けることで処理を行うために必要とする資源を補うことを容易としている。

次章では RaDiuS を構築するための設計について述べる。

第4章　RaDiuSの設計

前章では資源処理指向型 RaDiuS が提供する機能とそれらを実現するために必要とする機能要件について述べた。機能要件は分散ホスト発見機能、アプリケーション移送機能、改竄対策機能が必要である。本章ではこれらをソフトウェアレベルでの実現をするための設計に触れる。

4.1　設計方針

本研究の目的は資源とアプリケーションの発見及び利用を効率的に行えるシステムを提供することである。このため、二章で述べたように、既存の分散処理システムと Peer-to-Peer システムの利点を RaDiuS の構築時に適応し、RaDiuS が持つ三つの特徴を実現する。これらの特徴をシステム内の機構として構築するために、以下に述べる設計方針を定める。

4.1.1　分散ホスト発見機構

柔軟性

RaDiuS が前提とするネットワークはホストによる不特定な接続及び切断があるネットワークである。このため、ホストによる急激な増加及び減少によってシステム全体のパフォーマンスが低下するようなネットワークであってはならない。RaDiuS ではネットワークへのホスト数の増減によってパフォーマンスの低下が起こらないようにノードの増減に適応なネットワークが構築できるように設計を行う。

耐故障性

RaDiuS が構築するネットワークでは特定のホストが停止することでネットワーク全体が停止または不安定になるようなことがあってはならない。このため、既存の分散システムのようにサーバ・クライアントモデルやプロキシモデルを利用することで耐故障性を備えたシステムの構築が困難となる。本システムを設計する上ではネットワーク内のノードが自律的であるサーバントモデルの採用をしつつ、既存のサーバントモデルの問題点である多量の通信の発生を解消するように設計方針を定める。

4.1.2　アプリケーション移送機構

非依存性

従来の分散処理システムでは処理を行うために必要なアプリケーションを格納したホストとデータを格納したホストはそれぞれ完全に別々のものとされていた。このため、例えデータが莫大な量があったとしてもネットワークを介して処理ホストへ転送するようなモデルを取っていた。RaDiuS ではアプリケーション及びデータを含め、それを格納するホストとの結合がないように分離していることを前提としている。このため、アプリケーションを実行するホスト、データを格納しているホスト、アプリケーションを格納するホストの組合わせがより自由に行える。このため、それぞれのアプリケーションとデータがホストに極力非依存なようにシステムを構築しなければならない。

4.1.3　改竄対策機構

安全性

RaDiuS 上で共有するアプリケーションにおいて悪意を持ったものが混在する可能性がある。このため、あらかじめ悪意のあるアプリケーションが利用されないように安全性を考慮した上で設計を行う。

4.1.4　システム全体の設計方針

簡易性

RaDiuS では専門性の高い知識を持っていない一般ユーザの利用も想定しているため、RaDiuS がどのようにして動作し、またどのようにしてアプリケーションを共有するかと言った複雑な操作をさせてはならない。このため RaDiuS の設計ではアプリケーションを共有及び発見を簡単に行えるように設計を行う。

4.2　本システムの概要

前節で述べた設計方針を基に本システムの設計を行った。本システムではアプリケーションの共有、検索と利用を支援するためのソフトウェアから成る。本システムを用いることでネットワークへ容易にアプリケーションを共有し、さらに他ノード上で共有されたアプリケーションを発見し利用することが容易となる。

4.3 全体構成

全体構成について述べる。本システムは図 4.1 で示すように、四つの部より成る。

- 他ノードへの存在の通知と発見を行う Export 部
- 通過するデータ及びアプリケーションの指紋を所得する Checksum 部
- アプリケーション及びデータを格納するための Storage 部
- アプリケーションの実行を行う Runtime 部

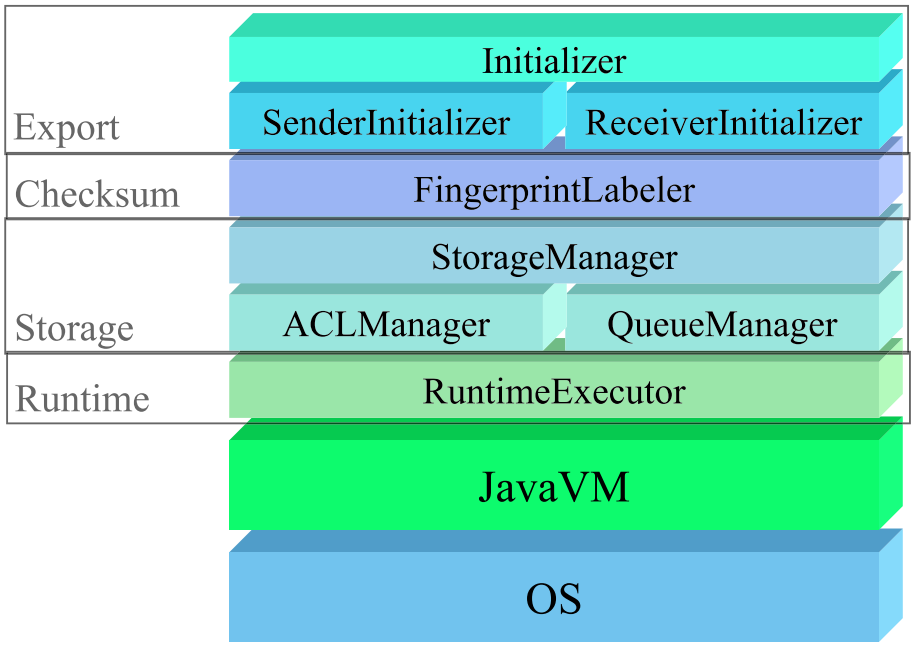

図 4.1: システム全体図

以下にそれぞれについて詳しく取り上げる。

4.4 Export 部の設計

4.4.1 Export 部の特徴

Export 部では他ノードの発見及び通信を行う部分にあたる。他ノードから通信が来た場合には返事を返し、ユーザが入力した検索内容を他ノードへ通知する役割を持つ。また、他ノードからは Export 部より内部にある Checksum 部、Storage 部、Runtime 部を参照することができないため、他ノードは必ずこの Export 部を介してノードと通信を行わなければならない。

4.4.2 Export 部の構成

Export 部は複数の構成からなる。主に SenderInitializer と ReceiverInitializer という二個のコンポーネントよる成る。

SenderInitiliazer は生成されると任意のマルチキャストへ自分の存在を登録し、ReceiverInitiliazer は他ノードからの通信を受け取ると返事を返す。また、マルチキャストアドレスに接続し、同じマルチキャストアドレスと接続している他ノードへ自らの存在を公開する。ここでもし他ノードから検索内容が送信されると Checksum 部を介して Storage 部と通信を行う。Storage 部では ReceiverInitializer から送られた検索内容と Storage 内部にあるアプリケーションの照合が行われ、一致すれば SenderInitiliazer を介して検索内容が送られたノードへ一致したことを通知する。逆に一致しなければ次のホストへ検索内容が送信される。

SenderInitializer は ReceiverInitializer が所得した他ホストに関する情報を格納する。各ノード及びそのノード上に割り当てられた鍵とノード自身のネットワークトポロジー上の所在情報 (IP アドレスなど) が一致したハッシュテーブルを所持し、他コンポーネントから鍵の要求があれば鍵が存在する所在情報を返す。表 4.1 にハッシュテーブルの記述例を示す。

c29cddcb82c7ff96073bd011ddd1ecf3	133.27.170.2
612970a69c12156db55055fe7e351aa4	133.27.170.10
c7c589628f463dc15046d6539e1fa742	133.27.171.223
28e4c27a0e968c79501c941b01caed94	133.27.171.224

表 4.1: SenderInitializer が所持するハッシュテーブル表

また、ハッシュテーブルは近いノードと遠いノードのそれぞれの情報を格納したハッシュテーブルに分けられている。ノードから他ノードへの遠近の基準は実装次第であるが、RaDiuS では TCP/IP 通信が 1 ホップ (TTL が 1) が届く範囲を元に実装を行う。これらのハッシュテーブルを元に、ノードは近いノードからの通信が送られると遠いノードへ通信し、遠いノードへ送られる通信は近いノードへ通信を行う。

なお、Export 部間通信で利用されるプロトコルは以下の機能要件を満たさなければならない。

A ネットワーク自身の耐故障性

B ネットワーク規模の拡張への柔軟な対応

A を実現するためには各ノードが自律的に動作する必要がある。このため、特定のノードに依存してはならない。このような点においては Napster Protocol が実行されるモデルは不適格となる。一方、B を実現するためにはノード間通信が少なくてはならない。このため、Gnutella Protocol のようにブロートキャストアドレスを参照する際にネットワーク帯域を通信で埋めるようなモデルは不適格となる。Freenet Protocol は上

述した A と B を満たすが、各ノードが検索を行う際にはあらかじめ他ノードの IP アドレスなどと言った情報を知る必要がある。RaDiuS ではこのようにあらかじめ他ノードに関する情報は理解していないことを前提としているため、Freenet Protocol と同じようなモデルを適応することが困難である。

RaDiuS ではプロトコルが持つ問題点を解消する **RDS**(Remotely Distributied Search) プロトコルを提案する。以下にプロトコルの動作手順を次のフェーズ順で示す。

ネットワーク接続時の動作手順 (INITIALIZE フェーズ)

INITIALIZE フェーズではネットワークに接続されたノードはまず他ノードの所在についての情報を収集する。本システムではまずマルチキャストアドレスを参照し、同じセッションを開いている他ノードへ新規接続したことを通知する。通知を受け取ったノードはユニキャストで発信元のノードへ返事をすることができる。このとき、返事をしたノードは自分が持つ他ノードに関する情報の一部を発信元のノードへ送ることでキャッシュ生成を早めることができる。

他ノード検索時の動作手順 (SEARCH フェーズ)

SEARCH フェーズではノードが持つ検索内容を他ノードを介して該当する情報を持ったノードを発見する。INIT フェーズで生成された所在表を基に検索内容を渡す。このとき、受け取ったノードに該当する内容を持っていなければ次のノードへ検索内容を渡す。もし該当する内容があれば検索内容が辿ったノードを逆順で発信元のノードへ検索の結果を返す。このとき途中に通過するノードの所在表には該当する資源と該当するホストの情報が所在表に記載される。これによって次回同様の検索があった場合には直接該当ノードへ接続することが可能である。また、該当ノードが記載されたエントリは所在表の一番上へ記載される。これは検索内容の多い項目が表内に残るようにするために行っている。

アプリケーション実行時の動作手順 (EXECUTE フェーズ)

EXECUTE フェーズではノードがアプリケーションの実行を行うための資源 (実行環境、データ、アプリケーション) の所在を所得してから行われる。まず実行を行うノードは実行環境を共有したノードの利用権限を所得し、そのノードを介してアプリケーションやデータを所得を行う。実行環境を持ったノードへデータ及びアプリケーションをダウンロードしたのち、実行を行った後に結果を実行命令を発したノードへ返す。

ネットワーク切断時の動作手順 (EXPIRE フェーズ)

EXPIRE フェーズではノードがネットワークから切断する前に、マルチキャストアドレスを介し、再度ノードの情報を所得後、ノード自身が持つ所在表を細かく分割し

ネットワーク内の各ノードに配分される。このとき他ノードの所在表には真ん中に分割された情報が入る。これは他ノードの情報を分散させることで再利用が可能となり、結果的に所在表を更新するための通信コストが減るために行われる。この後所在表を配分し終えたノードはネットワークから切断される。

4.5 Checksum 部の設計

4.5.1 Checksum 部の特徴

Checksum 部では Export 部と Storage 部の間に行き来するデータの指紋 (Message Digest) を取る。さらにアプリケーションまたはデータが送信される際にそれらのファイルの Message Digest を書き込む。これはデータまたはアプリケーションとそれぞれの Message Digest と対応させたハッシュテーブルであり、後に Storage 部でのアプリケーション及びデータの管理、または最終的に Runtime 部で実行するために必要となる。

4.5.2 Checksum 部の構成

Checksum 部の FingerprintLabeler は通過するデータやアプリケーションの指紋 (Message Digest) を取る。このときアプリケーションまたはデータに添付されているファイルに Message Digest を書き残す役割を持ち、このファイルはアプリケーションまたはデータを共有した時点で作成され、内容が複製されたものが転送される。これを元にデータやアプリケーションの正当性を調べることとなる。

なお、Storage 部に共有されたアプリケーションがあり、他ノードへの送受信を行うためには必ずこの Checksum 部を介して通信を行わなければならない。同時に、この Checksum 部はただ Message Digest を取り、書き込む機能さえ持っていればよい。

4.6 Storage 部の設計

4.6.1 Storage 部の特徴

Storage 部では公開するアプリケーションとデータ、または外部から送られたアプリケーション及びデータを格納するための場所である。Storage 部に格納されるアプリケーションは直接実行することができず、Runtime 内の RuntimeExecutor によってデータと組合わせて初めて実行される。

4.6.2 Storage 部の構成

Storage はアプリケーションとデータが対となったタスク管理を行う ResourceHandler、Runtime への処理を行い際のタスクキュー、または他ノードから要求される資源のキュー

管理を行うQueueManager、ノード別の資源へのアクセス制限を行うACLManagerの三つのコンポーネントからなる。なお、Storage部はあくまでデータ及びアプリケーションを保管するための空間であるため、アプリケーションはこのStorage内から直接実行することができない。アプリケーションを実行するためにはRuntime内のRuntimeExecutorへ渡す方法しかない。

ResourceHandler コンポーネント

ResourceHandler コンポーネントはダウンロード・アップロードされるアプリケーション及びデータの排他制御と各アプリケーションとデータの保管を担当する。具体的にはダウンロードの要求があれば要求があった資源を送信する準備をし、送信を行う。または他ノードからアップロードされた資源を保管する際に、各資源とそれらのMessage Digestを対応させたハッシュテーブルを持ち、これを元に他ノードからの要求に応じて資源の送信を行う。図4.2に示す。

ResourceHandlerが格納するデータ・ハッシュテーブル	
d6f04b8bbb363f519ef9a85d522977be	spacemusic.au
89d1f20f7e4e88bf8aff8664d369b9be	sample.jpg
b30fd36d0f21357b4502ff2dbdfe39d0	AcroDist.exe
36872ac87021a8456dc3521d6a8e92f4	guntar

表4.2: ResourceHandlerが所持するハッシュテーブル表

QueueManager コンポーネント

複数のノードから同じ資源の要求があった場合、またはランタイムへのタスクが多くあった場合に資源のキュー管理を行わなければならない。このキュー管理はQueueManager部によって行われる。キューは誰がどの資源を必要とするかというハッシュテーブルによって管理される。QueueManagerはこのハッシュテーブルを参照することで資源の配分が行えるようになる。

ACLManager コンポーネント

資源を提供する側は外部に公開したい資源の管理だけでなく、共有の制限を柔軟に行えるようにすることが望ましい。このアクセス可能な資源の制御を行うコンポーネントとしてStorage部のACLManagerがある。ACLManagerは資源別におけるアクセス制限を行い、ユーザによる柔軟な資源の提供を可能としている。

4.6.3 Storage 部のコンポーネント間関係

Storage 部内のコンポーネントにおける関係図を図 4.2 に示す。保管された資源の入れ替えができるのは ResourceHandler のみであるが、他ノードの資源へのアクセスを最終的に決定するのは ACLHandler と QueueHandler である。まず ResourceHandler は ACLManager へのノードのアクセスを許可しているかを確認する。ノードのアクセスがなければ通信は切断されるが、もし制限があれば制限通りのアクセス権が与えられる。同時に QueueManager に現在リクエストされている資源は使用中であるかを問い合わせられ、使用中であればユーザのリクエストはキューに入り、資源が解放されるまで待つ。ACLManager と QueueManager ともに許可が降りることで ResourceManager ははじめてノードへのアクセス権を与えることができる。

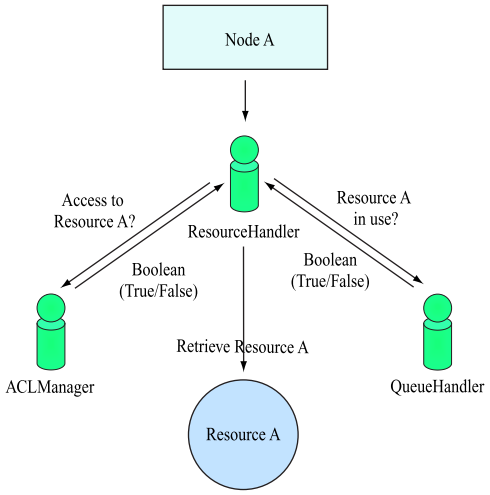

図 4.2: Storage 部のコンポーネント間の関係

4.7 Runtime 部の設計

4.7.1 Runtime 部の特徴

Runtime 部はアプリケーションを実行するための部分にあたる。本部分が提供するのは実行環境のみであり、アプリケーションとは別である。このため、Runtime 部では実行のみを担当するコンポーネントさえあれば十分である。Runtime 部ではこの役割を果たすのは RuntimeExecutor である。

4.7.2 Runtime 部の構成

RuntimeExecutor コンポーネント

Runtime 部はアプリケーションを実行するための環境を提供する。Runtime 内の RuntimeExcecutor が Storage 部の ResourceHandler からアプリケーション及びデータを受け取り、実行を行う。これは UNIX でいうシェルのようなものであったり、RMI プログラムのスタブファイルのようなものであったもよいように、システムとのインタフェースを自由に入れ替えられるように、各インタフェースに対するモジュールを用意する。つまり RuntimeExecutor 自身には複数の種類がいることを意味する。

4.8 本章のまとめ

本章では分散ホスト発見機構、アプリケーション移送機構、改竄対策機構の設計方針として柔軟性、耐故障性、非依存性、安全性、簡易性を重視することを目的として定める。この際、これらの機構を実現するために、RaDiuS のサブシステムである Export 部、Checksum 部、Storage 部、Runtime 部の設計について述べた。また、これらのサブシステムを構築するために必要なソフトウェア・コンポーネントについて述べ、ソフトウェア全体の構成を具体化した。次章では RaDiuS の設計を基に、プロトタイプの構築方法について述べる。

第5章　RaDiuSの実装

5.1　実装環境

本システムの実装はJava言語を用いて行った。実装言語としてJava言語を選んだ理由は以下の通りである。

マルチプラットホーム性

Java言語で書かれたプログラムは一度中間バイトコードを生成し、その後Java Virtual Machineと呼ばれるインタプリタを用いて実行される。このため、一度コンパイルされたプログラムはマシンアーキテクチャが異なるJava Virtual Machineであっても再コンパイルせずに実行ができる。本システムはネットワークに接続されたマシンアーキテクチャが同じとは限らないホスト上で利用されることを前提としているため、この特性は有益である。

ネットワーク指向型

本システムにおける作業時間の多くはネットワークを利用したアプリケーション及びデータの転送である。このため、ネットワークを考慮したプログラミング言語を用いて開発を行うことが前提となる。Java言語はネットワークに関するライブラリが豊富であり、ネットワーク通信を実装したアプリケーションの構築が容易である。

ライブラリの充実

本システムはネットワーク上のホスト上へばらまき、稼動する状態であることを前提としている。このため、本システムを動作するために必要とする実行環境はそれらのホスト上に存在しなければならない。さらに、本システムが提供する機能が必要とするライブラリがそれらの実行環境において提供されていなければならない。このため、ライブラリが充実した開発言語を用いることが適格である。

以上の理由により、本システムの実装言語としてJava言語を採用する。

本システムの実装環境を表5.1に示す。

項目	説明
ハードウェア	AMD Athlon 900MHz
オペレーティングシステム	Microsoft Windows 2000 5.00.2195
開発言語	Java Standard Development Kit 1.3.1

表 5.1: 実装環境

5.2 Export 部の実装

5.2.1 Export 部の機能要件

前章で述べたように、Export 部は他ノードとの通信を行うための部分である。この部は以下の機能をサポートしなければならない。

- 他ノードとのデータのアップロード・ダウンロードにおいて双方向通信が行えること
- INITIALIZE, SEARCH, EXECUTE, EXPIRE フェーズのそれぞれのサポートすること
- 他ノードの発見及び他ノード上の資源の発見をサポートすること

5.2.2 Export 部の構成コンポーネント

Export 部を構成するコンポーネントは主に以下のものが挙げられる。

- Initializer コンポーネント
- SenderInitializer コンポーネント
- ReceiverInitializer コンポーネント

Client と Server コンポーネントではそれぞれデータのダウンロード・アップロードをサポートするためのモジュールである。ClientThread と ServerThread は並列でそれぞれの処理が行えるようにするためのモジュールである。このようにすれば Server としても Client としても機能を果たす Servent を構築することが可能である。

また、既存の Peer-to-Peer システムのように帯域を多く消費しないように RaDiuS を構築す。これには設計で述べたように、Gnutella とは異なりブロードキャストアドレスを利用せずにマルチキャストを利用することでトラフィックを減らすことができる。また、他ノードへの通知のみをマルチキャストで行い、返事はユニキャストで行うことでさらにトラフィックを減らすことができる。

Initializer コンポーネント

RaDiuS のコアでもある他ノード発見機能の総合管理を行うのが Initializer である。総合管理としては外部からのパケットを受け取る Server スレッドと Client の機能を提供するスレッドを起動し、終わるまで待つという内容である。なお、incomingPacket() はマルチキャストから送られるパケット (“NEW” というヘッダのついたパケット) へユニキャストへ送り返すためのメソッドである。 これを行うために Server スレッドから Client スレッドへ要請のあったノードへ返事するために一度両スレッドを管理する Initializer へ介して渡されるように実装した。インタフェースを図 5.1 に示す。

なお、インタフェースを実装したクラスとして図 5.2 を示す。

```
public interface Initializer {
    public abstract void incomingPacket(java.lang.String);
}
```

図 5.1: Initialiazer の構成

```
public class Initializer_impl extends java.lang.Object \\
implements Initializer {
    public Initializer_impl(java.lang.String);
    public static void main(java.lang.String[]);
    public void incomingPacket(java.lang.String);
}
```

図 5.2: Initialiazer_impl の構成

SenderInitializer コンポーネント

SenderInitializer は他ノードへの通信を送信するための Client スレッドを提供するためのコンポーネントである。Export 部における他ノードの Export 部との通信はこのコンポーネントを介して行われる。パケットは送り出される前に setHeader() によってヘッダをつけかえてから送り出される。ヘッダの種類は表 5.2 に示す。

sendPacket() はマルチキャストアドレスへパケットを送るためのメソッドである。これは現在 “NEW” のヘッダがついたパケットであるネットワークに新規接続されたときのみに呼び出される。その他のパケットは他ノードへユニキャストで送信するので sendUnipacket() メソッドを利用する。インタフェースを図 5.3 に示す。

一方、’FWD’ ヘッダは特殊であり、これは遠隔ノードへ命令を実行するために利用するヘッダである。まず’FWD’ ヘッダの真後ろに実行命令を行うヘッダをつけ、ノードへ送信する。受け取ったノードはまず FWD のヘッダを発見すると FWD ヘッダを送信内容から切り落とし、次に記述されたヘッダを参照し、そのヘッダに対応した動作を行う。

```
public interface Initializer {
 public class SenderInitializer extends java.lang.Object {
    java.lang.String address;
    int port;
    java.lang.String header;
    java.lang.String message;
    java.lang.String remoteMessage;
    java.lang.String localAddress;
    java.lang.String uniAddress;
    public SenderInitializer(java.lang.String);
    public void run();
    public void setHeader(java.lang.String);
    public void sendPacket();
    public void sendUnipacket();
 }
}
```

図 5.3: SenderInitialiazer の構成

NEW	ネットワークへ新規接続したノードが送信する
ACK	NEW を受け取ると発信したノードへ返す
HAS	検索するときに送信される
HIT	検索内容と内容が一致した資源を所持時に返す
GET	資源を受信するときに送信される
PUT	資源を送信するときに送信される
FWD	遠隔ノードへ命令を実行するときに送信する

表 5.2: RDS プロトコルで利用可能なヘッダ表

ReceiverInitializer コンポーネント

ReceiverInitializer は Sender とは逆にノードへ送信されるパケットを受信するためのコンポーネントである。getPacket() は受信したパケットを Initializer を介して SenderInitilializer へ他ノードへ ACK パケットを返信するためにある。インタフェースを 5.4 に示す。

```
public class ReceiverInitializer extends java.lang.Thread {
    int port;
    java.lang.String address;
    java.lang.String message;
    java.lang.String remoteMessage;
    java.lang.String localAddress;
    Initializer initializer;
    public ReceiverInitializer(Initializer);
    public void run();
    // implement to send packets.
    public void getPacket();
}
```

図 5.4: ReceiverInitializer の構成

5.2.3 Export 部の実装状況

Export 部の実装状況を表 5.3 に示す。

他ノード発見機能	○
他ノード間他ノード発見機能	○
他ノード応答機能	○
キャッシュ採取機能	○
キャッシュ解放機能	×
資源検索機能	×

表 5.3: Export 部の実装状況表

5.3 Checksum 部の実装

5.3.1 Checksum 部の機能要件

Checksum 部では扱うデータ及びアプリケーションの指紋を取る仕組みが必要である。これには現在最も普及されている 128bit メッセージダイジェストが可能な MD5 ア

ルゴリズムを使用する。しかし、近年では MD5[17] で異なる値を持って同じハッシュを生成することが確認されているため、将来的の利用用途を考慮し、160bit メッセージダイジェストが可能な SHA[18] アルゴリズムが容易に行えるように実装を行う。

- メッセージダイジェストが生成できること

5.3.2 Checksum 部の構成コンポーネント

Checksum 部は以下のコンポーネントから成る。

- FingerprintLabeler コンポーネント

FingerprintLabeler コンポーネント

FingerprintLabeler は Export 部から Storage 部へアプリケーション及びデータを渡す際にそれぞれの正当性を確かめるためのコンポーネントである。前章で述べた通り、ただ値を渡すだけでハッシュが返ってくるようにすればよいのである。このため、プロトタイプでは Message Digest を生成する機能のみを実装しており、これによって処理後の結果を送信する際に MD5 の送信結果も送るように実装を行った。なお、将来性を考慮して用意に他のアルゴリズムへの切り替えが行えるようにすることが望ましい。プロトタイプではダイジェストアルゴリズムを容易に SHA へ移行できるように実装を行った。インタフェースを図 5.5 に示す。

```
public class FingerprintLabeler {
    java.lang.String algorithm;
    java.security.MessageDigest messageDigest;
    public FingerprintLabeler(java.lang.String);
}
```

図 5.5: FingerprintLabeler の構成

5.3.3 Checksum 部の実装状況

以下に Checksum 部の実装状況を示す。

機能	実装状況
メッセージダイジェスト生成機能	○

表 5.4: Checksum 部の実装状況表

5.4 Storage 部の実装

5.4.1 Storage 部の機能要件

Storage 部ではノードが持つデータ及びアプリケーションの管理を行うための部分である。前章で述べたように、Runtime 部及び他ノードからの資源のアップロード・ダウンロード、さらにキュー管理やアクセス権の管理が行えるようにしなければならない。具体的には以下の五つの機能が提供できればよい。

- Export 部からダウンロードされたアプリケーション及びデータの保管
- Export 部からの要求に応じてアプリケーション及びデータのアップロード
- Runtime 部へのアプリケーション及びデータの転送
- Runtime 部からのデータの受け取り
- Runtime 部のキュー管理

5.4.2 Storage 部の構成コンポーネント

以下に Storage 部の構成コンポーネントを示す。

- ResourceHandler コンポーネント
- ACLManager コンポーネント
- QueueManager コンポーネント

ResourceHandler コンポーネント

本プロトタイプでは ResourceHandler コンポーネントの実装を行った。ResourceHandler は Intilializer と同様、二つのスレッドの管理を担当する役割を持ったコンポーネントである。同時に、ファイルの書き込みと呼び出しを担当するコンポーネントである。現時点のプロトタイプでは名前による検索内容のサポートのみ、ACL とキュー管理ともに実装は完了していないため、ファイルの入出力のみしか行えない。図 5.6 にインタフェースを示す。

ACLManager コンポーネント

上述したように、実装は完了していない。しかし実装は困難なものではなく、ハッシュテーブルへの書き込みと読み出しさえあれば簡単なものは実装できる。また、ハッシュによるエントリの制御を行うことでホスト、ドメイン、人などのアクセス対象種

```
public class ResourceHandler extends java.lang.Thread {
    java.lang.String resourceName;
    java.util.Hashtable hashtable;
    java.io.FileInputBuffer fileInputBuffer;
    java.io.FileOutputBuffer fileOutputBuffer;
    public ResourceHandler(Hashtable hashtable);
    public void run();
    public void setResourceName(String resourceName);
    public void putResource(String resourceName);
    public void getResource(String resourceName);
    public void checkACL(String resourceName);
    public void checkQueue(String resourceName);
}
```

図 5.6: ResourceHandler の構成

類の直接指定を行うことなく ACL 制御が可能となる。図 5.7 にサンプルインタフェースを示す。

```
class ACLManager extends java.lang.Thread {
    java.lang.String resourceName;
    java.lang.String hashEntry;
    java.util.Hashtable Hashtable ACLControl;
    ResourceHandler resourceHandler;
    public void run();
    public String addAccess(String entry, String resourceName);
    public String removeAccess(String entry, String resourceName);
}
```

図 5.7: ACLManager の構成

QueueManager コンポーネント

QueueManager コンポーネントも ACLManager コンポーネントと同様、実装は完了していない。QueueManager はキュー管理を行うための Hashtable、ResourceManager へ資源の使用が解放されたことを通知する機能を持っていれば簡単なものが実装できる。また、資源の仕様が解放されたことを ResourceManager へ通知するためのメソッドを実装するとよい。図 5.8 にインタフェースを示す。

5.4.3 Storage 部の実装状況

表 5.5 に Storage 部の実装状況を示す。

```
class QueueManager extends java.lang.Thread {
    java.lang.String resourceName;
    java.lang.String entry;
    java.util.Hashtable Hashtable QueueSpool;
    ResourceHandler resourceHandler;
    public void run();
    public String addQueue(String entry, String resourceName);
    public String removeQueue(String entry, String resourceName);
    public void fireEvent();
}
```

図 5.8: QueueManager の構成

機能	実装状況
資源管理機能	○
キュー管理機能	×
資源 ACL 機能	×
多種の検索内容への対応	×

表 5.5: Storage 部の実装状況表

5.5 Runtime 部の実装

5.5.1 Runtime 部の機能要件

Runtime 部ではアプリケーションとデータを組合わせ、実行を行うための実行環境を提供する部分である。Runtime 部はアプリケーションとデータを取り込み、プロセスを生成することができる実行環境が提供できればよいのである。

- プロセス生成が可能

5.5.2 Runtime 部の構成コンポーネント

Runtime 部は以下の構成により成る。

- RuntimeExecutor コンポーネント

RuntimeExecutor コンポーネント

RuntimeExecutor はアプリケーションをプロセス化する機能さえあればよい。プロトタイプではコンストラクタ経由で class ファイルからプロセスが生成できるように構築した。しかし、耐故障性を高めるためにプロセスのステータスを監視する機能を追加

することが望ましい。また、プロトタイプの RuntimeExecutor はスレッド化していない。これは排他制御が複雑になるのを避けるためにキュー管理を導入するためである。インタフェースを図 5.9 に示す。

```
public class RuntimeExecutor {
    java.lang.String appName;
    java.lang.String dataName;
    java.io.InputStream inputStream;
    java.io.OutputStream outputStream;
    public RuntimeExecutor(String appName, String dataName);
    public void taskReceiver();
    public void taskSender();
}
```

図 5.9: RuntimeExecutor の構成

5.5.3 Runtime 部の実装状況

Runtime 部の実装状況を表 5.6 に示す。

コンポーネント名	実装状況
プロセス生成機能	○
プロセス状態管理機能	×

表 5.6: Runtime 部の実装状況表

5.6 応用例

応用例として以下に本システムを利用した応用例を示す。

ホスト透過的な GUI ベースアプリケーション

RaDiuS を用いることでホスト透過的なアプリケーションの構築が容易となることは本論文内ですでに述べたが、例えば特定のアプリケーションへ適応することで新しい利用方法が可能となる。例えばブラウザやワードプロセッサと言った GUI ベースのアプリケーションを例に挙げると、RaDiuS 内で動作させることで他ノード上で表示されているアプリケーションをマウスでクリックし、ドラッグアンドドロップでユーザが利用するノード上へ移送することが可能となる。これによって例えばプロセスマイグレーションなどの機能をサポートすることで特定のノード上で実行中のアプリケーションを実行中の状態のまま移送することが可能となる。

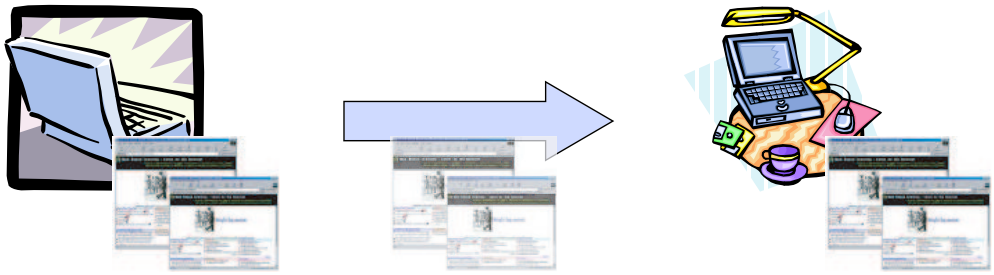

図 5.10: ホスト非依存なアプリケーションの例

ファイル共有

RaDiuS を既存のファイル共有システムの一つとして利用することが容易である。但し、既存のファイル共有アプリケーションである Napster や Gnutella とは異なり、本システムの Export 部が行う RELEASE フェーズを利用することでアップロードにおける動的負荷分散が可能となる。例えば、ホスト A からファイルをダウンロードしているホスト B があったとする。ここで例えばホスト A がネットワークから切断しなければならないとき、RELEASE フェーズにてキャッシュを参照し、同じファイルを持ったホストがあった場合にそのホストへダウンロード中のホストを参照する。

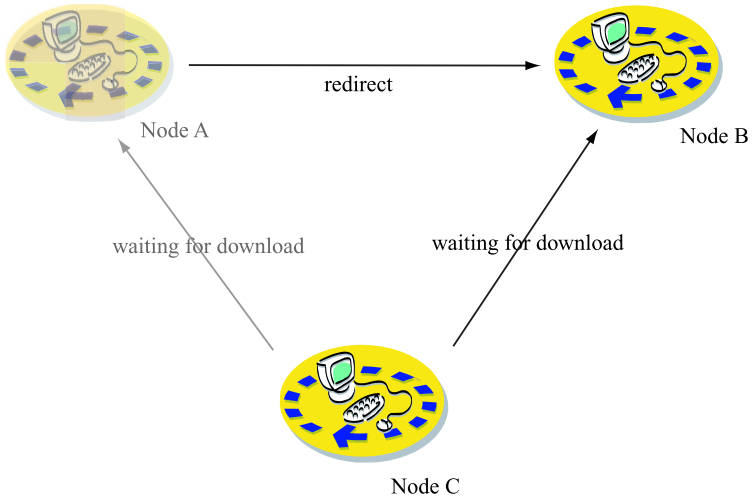

図 5.11: 動的負荷分散ファイル共有の例

ビデオストリーミング

既存の Peer-to-Peer システムでは動画ファイルをダウンロードしてから再生ソフトウェアを利用することで見ることができる。RaDiuS ではこの例を応用して、Runtime 内に動画データを分割し、RTP[19] をはじめとしたストリーミングプロトコルを用いることでホストへ転送することが可能となる。これによって Peer-to-Peer システム内においてストリーミングサービスを提供することができるだけでなく、ダウンロード前に動画のプレビューが可能になる。またはダウンロードはさせたくないがメディア配信はしたいという利用用途に適応できる。

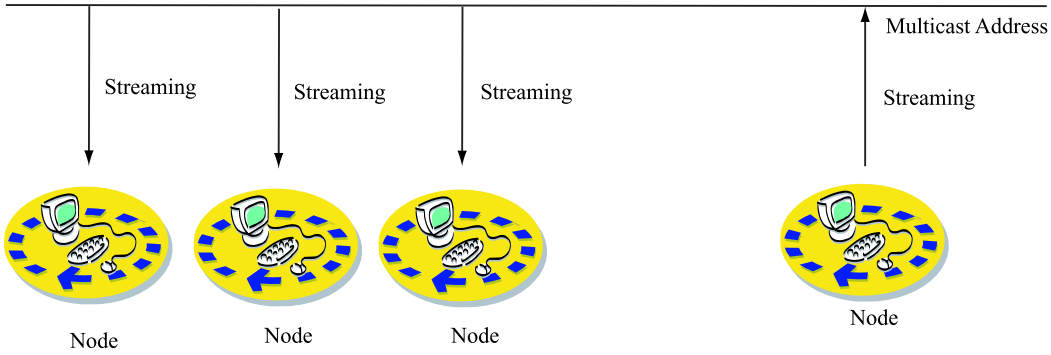

図 5.12: 分散ビデオストリーミングの例

暗号解読

本システムはマシンアーキテクチャに非依存なモバイルコードなアプリケーションを利用することで暗号解読を例に挙げられる。例えば数年前に行われた CSC プロジェクト [20] では、複数のホストにクライアントをダウンロードし、解読用のデータを格納したサーバへデータを受け取り、処理した後にデータをサーバへ返すモデルである。本システムではサーバという概念はないが、アプリケーション及びデータを送信した元ノードへデータを転送することを可能にしている。このため、例えば複数のノード上で処理を行い、ノードへ結果を転送することでデータの並列処理を行うことができる。

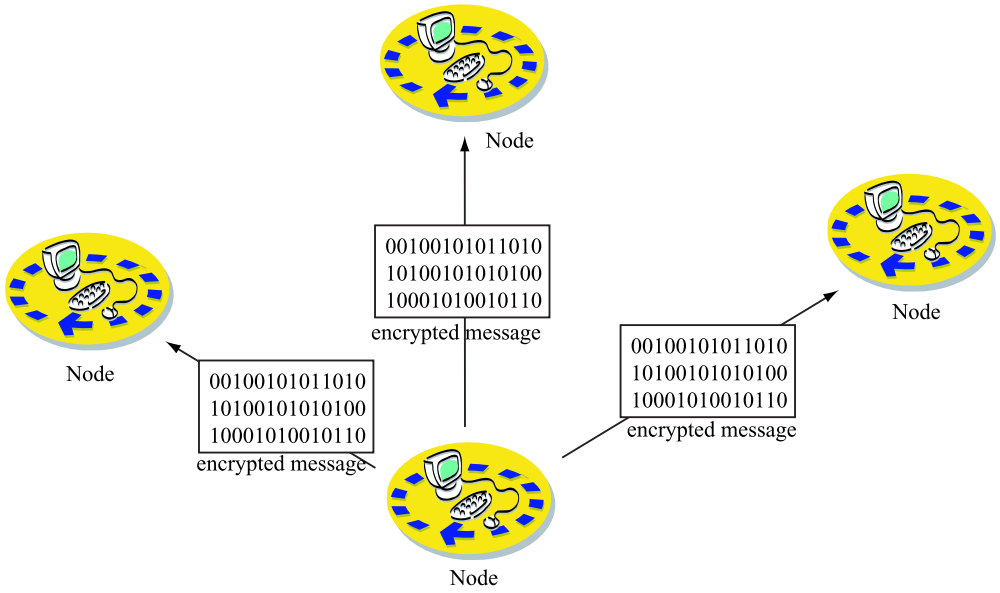

図 5.13: 分散暗号解読の例

5.7 本章のまとめ

本章では資源処理指向型分散システムの実装例である RaDiuS のプロトタイプの実装について述べた。まず RaDiuS の実装環境について述べた。次に Export 部、Checksum 部、Storage 部、Runtime 部の実装について述べた。本章の最後では RaDiuS を用いた応用アプリケーションの具体例としてホスト透過的な GUI アプリケーション、ファイル共有、ビデオストリーミング、暗号解読の四つを示した。

第6章　システムの評価

6.1　定量的評価

6.1.1　測定環境

測定環境として本研究室と共同研究を行っている MKG[21] に環境を借り、本研究室のネットワークと MKG のネットワークを介して測定を行った。環境は以下の構成となっている。

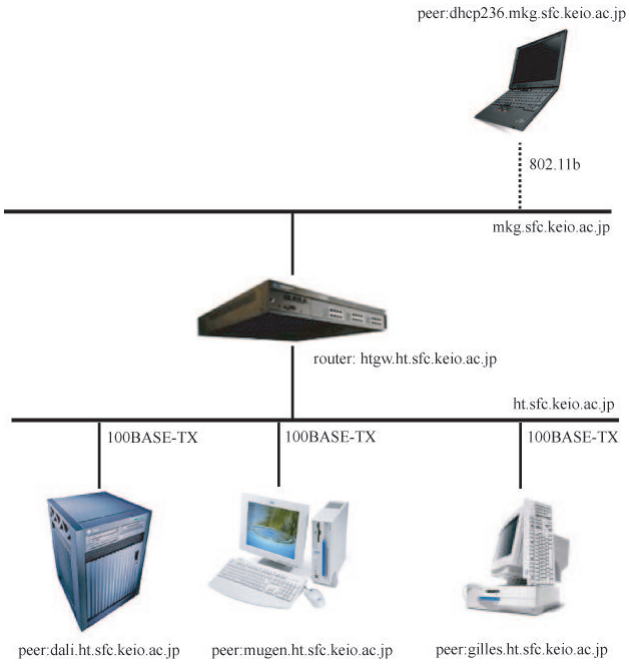

図 6.1: 測定環境

各マシンの仕様は表 6.1 に示す。

項目	mugen	gilles	dali	laptop
ハードウェア	AMD Duron 800MHz	AMD Athlon 1GHz	UltraSparcII 248MHz x2	Pentium III 800MHz
オペレーティングシステム	FreeBSD 4.5-RC	Linux 2.4.14	SunOS 5.7	Windows2000 5.00.21.95
実行環境	JDK 1.3.1 (FreeBSD-native)	JDK 1.4-rc	JDK 1.3.1	JDK 1.3.1

表 6.1: 測定で利用したマシンの仕様表

6.1.2 測定方法

各ホスト上に RaDiuS を立ち上げ、それぞれを本ネットワークの peer として機能してもらうようにした。dali と gilles というマシンは常時立ち上げておき、mkg ネットワークへ無線 LAN で接続されたラップトップに DHCP アドレスを振って立ち上げた。ここに mugen というマシンをネットワークに参加させ、全ノードの発見が行えるかを検証した。

6.1.3 測定結果

図 6.2 に測定結果を折れ線グラフを示す。

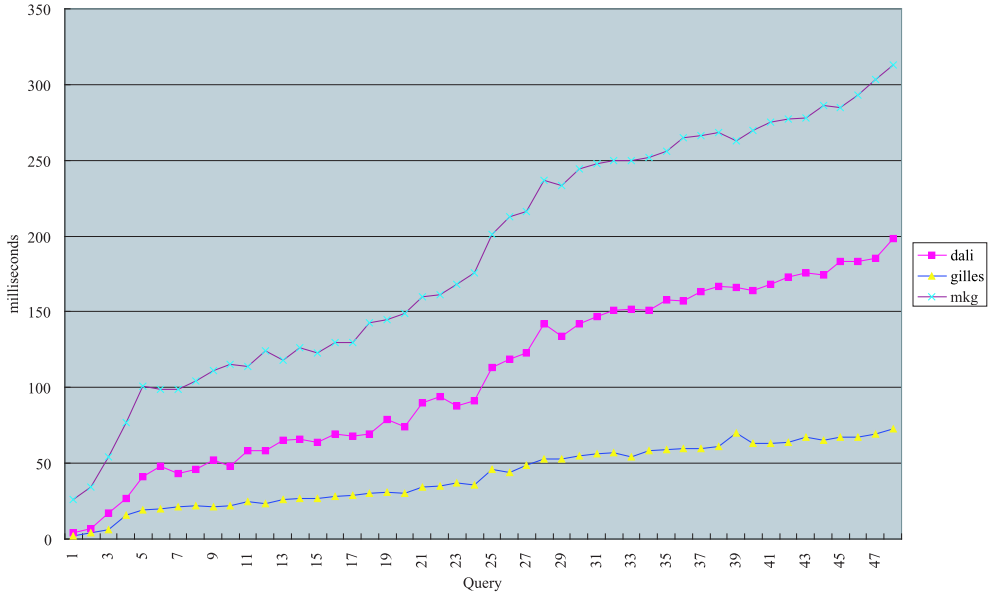

図 6.2: 測定結果

図 6.2 からは同ネットワークセグメント内のノードであっても検索内容一つにつき遅延は 2millisec まであり、1 ホップ先のネットワークでは 6millisec の遅延があることが分かる。これによってユーザは検索内容一つで 3 秒までの遅延を許すものだとすれば本システム内で扱う TTL(ノード間通信の最大数) は 4 から 5 くらいに設定するのが適格であることが分かる。

次に RDS プロトコルと近似する Gnutella Protocol を理論的な面から分析をする。

ネットワークセグメントが許可する最大ホスト数を m、ネットワーク内に接続されているホスト数を n、そのうち Peer-to-Peer システムのノード数を o として表す。(但し、m >= n >= o とする。)

Gnutella ではネットワークのブロードキャストアドレスへパケットを送り返事を待

つのだが、ここでブロードキャストアドレスを参照するため、Gnutella ノードではないホストから返事が来てしまう。このことを式に表すと、一回の検索内容によって、

$$m + n$$

分の通信量を必要とする。

一方、RDS プロトコルではネットワークに新規参加した場合において、マルチキャストアドレスを参照することで他ノードの情報の返事を依頼する。他ノードはマルチキャストアドレスから存在の通知を行うその後ユニキャストで返事を送る。このため、マルチキャストアドレス上に流れるパケット数は o、返事はノード分の返事をユニキャストで返事をするので、

$$2o$$

分の通信量が発生する。

ここで (m >= n >= o) により、

$$(m + n) >= 2o \tag{6.1}$$

となる。RDS プロトコルは 19 バイトであるのに対して Gnutella プロトコルは 20 バイトであるため、例えこの結果にパケットの大きさを掛けても結果は変動することがない。

以上により、ブロードキャストアドレスを利用した検索方法である Gnutella に比べ、RDS プロトコルによって発生する通信量がより少ないことが証明できた。

6.2 定性的評価

定性的評価としては資源指向型システムである分散処理システムと資源指向型システムである Peer-to-Peer システムとの比較を行う。なお、実装は完了しなかったため、これらの評価は設計レベルの RaDiuS との比較を行う。

6.2.1 既存の分散処理システムとの比較

本項では既存の既存分散処理システムとの機能比較を行う。評価項目及び評価結果を表 6.2 に示す。

PVM は UNIX コンピュータ群を並列処理コンピュータとして利用できるライブラリ及びコマンド集である。このため、ライブラリはそれぞれ UNIX のベンダー別で記述されるため、利用可能なアプリケーションも同種の UNIX 上でしか利用できない。さらに実行ファイルをコンピュータアーキテクチャー別でメンテナンスしなければならない。

PopularPower はコンピュータの遊休状態時にあらかじめダウンロードされたクライアントソフトウェアを介してサーバへ接続する。計算処理を行うためのデータを入手後、計算をし、サーバへ結果を返す仕組みとなっている。現時点では Windows, Linux,

Macintosh 別のクライアントが用意されているが、PVM と同様それぞれのプラットホーム上におけるメンテナンスに関わる煩わしさが残っている。

これに対してプラットホームに依存しないシステムとして Javelin, Ninflet, MetaSpace を挙げることができる。

Javelin はアプレットを介してホスト上のアプリケーションの遠隔利用を可能とするシステムである。アプレットを利用していることにより、アプリケーションによる処理が完了するまでの間、アプレットを起動し続けなければならないため、耐故障性は弱い。

Ninflet と MetaSpace はグローバルレベルにおける計算資源利用を目的とした動的分散並列処理を目的としたシステムである。これらのシステムには耐故障性として実行中のプロセスを常に見張り、定期的に計算結果をバックアップしている。もし処理ホストが故障などによって機能停止した場合、復旧したときにバックアップから作業の継続を行うことが可能である。RaDiuS ではこのような機能はまだ実装を行っていないが、Runtime 部にこのような機能を提供することは４章の設計段階で考慮しており、実装を行う予定である。

	PVM	PopularPower	Javelin	Ninflet	MetaSpace	RaDiuS
モデルタイプ	ServerClient	ServerClient	ServerClient	Proxy	Proxy	Servent
プラットホーム依存性	×	×	○	○	○	○
耐故障性	×	×	×	○	○	△
資源発見が行えるか	×	×	×	×	×	○

表 6.2: 既存の分散処理システムとの機能比較表

6.2.2 既存の Peer-to-Peer システムとの比較

本項では既存の Peer-to-Peer システムとの機能比較を行う。評価項目及び評価結果を表 6.3 に示す。

Napster は Napster サーバを介することで他 Napster クライアントを発見することができる。発見後はクライアント同士でファイルの通信を行う。しかし、同時に Napster サーバが停止すると同時にクライアントの機能も停止する。このようなモデルは耐故障性に弱いと言える。

Gnutella ではブロードキャストアドレスを介して他ノードの存在を知ることができる。他ネットワーク内のノードの存在を知るためには特定のノードへ接続し、情報を入手することも可能としている。このようなモデルにおいて Napster のように耐故障性に弱い点を持たないが、上述した特定のノードが停止すると同時にネットワークの検索効率が悪くなる。

Freenet では Gnutella のようにブロードキャストアドレスを利用せずに、あらかじめ設定された他ノードへのルーティングを行う際に参考となるキャッシュを所持している。このキャッシュは検索時におけるヒット率の高いノードの情報で入れ替わることで、いずれはヒット率の高いノードを入手することを可能としている。しかし、この

キャッシュを生成することには時間がかかり、ネットワークに新規接続したノードに取ってはあまり望ましくない。

本システム内で利用する RDS プロトコルは Freenet と類似しているが、異なる点としてはマルチキャストアドレスを利用することで同ネットワーク内のノードから情報を入手することを用意としている。また、マルチキャストアドレスによっては複数のネットワーク内のノードを発見することも可能となっているため、ルーティング用のキャッシュを早急に入手することが可能となる。

各プロトコル対応状況				
	Napster	Gnutella	Freenet	RDS
ホスト非依存性	×	△	△	○
ネットワーク負荷が低いか	○	×	○	△
キャッシュ入手の容易さ	○	○	×	○

表 6.3: 既存の Peer-to-Peer プロトコルとの機能比較

6.3 本章のまとめ

本章では RaDiuS プロトタイプの評価を行った。これらの比較は実装が完了した状態を仮定とする。本章の前半では RaDiuS プロトタイプの特徴である分散ホスト発見を可能とする RDS プロトコルについてのパフォーマンスについて定量的評価を行い、分散ホストの発見が適格に可能であることを実証した。本章の後半では、RaDiuS を関連研究のシステムと機能比較を行い、処理資源型分散システムを構築する上で適格なモデルであることを示した。

第7章　おわりに

7.1　今後の課題

RaDiuS システムの今後の課題として示以下の四点を挙げる。

Storage 部の完全実装

RaDiuS で未実装な機能はたくさんあるが、その中でも資源の共有を行う Storage 部の実装は必然である。例えばどのノードにどの資源を提供するための ACL 機能や、Runtime 部への処理内容のキュー管理は実用的な運用を考えると必然な機能である。また、今後の発展として Export 部との連携で通信帯域の制御を実装することを検討している。

検索メカニズムの拡張

今回実装した RaDiuS のプロトタイプでは文字列による名前の検索しか行うことができない。今後の予定として、鍵ベースの検索を実装し、さらに mime との連携によって絞り込み検索を可能とした検索メカニズムを提供するように予定している。

セキュリティの強化

今回の実装では有害資源における予防対策は実装できたが、予防しきれなかったときの対処方法は実装することができなかった。例えば Java 言語のみにしぼれば、Java 言語が提供する柔軟なセキュリティポリシーやインタプリタを利用した不正スタックエラーの防止などと言った対処方法を実装する方法がある。このように、広域ネットワーク内での利用へより適格とするためにはセキュリティの強化は必然である。

プロトコルの改良

RDS プロトコルでは単純な実行及びダウンロードまではできたが、RaDiuS の特徴でもある RELEASE フェーズで行われるキャッシュの解放を現時点のプロトコルでは実現することはできない。これは GET でキャッシュを送信するとノードはデータが来たと誤認識し、ACK が来ると反応しないため、ルーティング情報であるキャッシュを

取り込むことができない。今後はこの機能の実装とともに、より表現能力の高いプロトコルの拡張を予定している。

7.2 まとめ

本論文では、アプリケーション、それらを実行するための環境である機器、実行する際に必要とする資源の入手の容易に、かつ多種多様なものが入手できることが可能となった現在のコンピューティング環境において、資源と機器の結び付きの複雑化、ヘテロジニアスな計算機環境内における機器間作業の困難化、有害な資源の入手経路の容易化という三つの問題点を指摘した。これらを解消し、かつ資源とアプリケーションの有効利用を可能とするシステムとして資源処理指向型分散システムを提唱し、そのプロトタイプである RaDiuS システムの特徴である分散ホスト発見機構、アプリケーション移送機構、改竄対策機構の三つを取り上げ、それらの具体化するためのシステムの設計と実装を行った。

その後に定量的評価及び定性的評価を行い、本システムは資源処理指向型システムとして適格なモデルであることを実証した。今後は検索メカニズムの拡張、セキュリティの強化、プロトコルの改良の実装に取り組む。

謝辞

本研究を進めるにあたり、御指導を賜わりました、慶應義塾大学環境情報学部教授 徳田英幸教授に深く感謝致します。

慶應義塾大学徳田・村井・楠本・中村・南研究会の先輩方には、折りにふれ貴重な助言及び御指導を頂きました。特に慶應義塾大学政策・メディア研究科 修士二年の岩井将行氏と修士一年の青木崇行氏には本論文執筆を終えるまでにあたり、絶えざる励ましと御指導を賜わりました。また、Keio Media Space Family(KMSF) 研究グループのメンバーには、本研究に関する様々な議論をして頂きました。

この場を借りて深く感謝の意を表したいと思います。

平成 14 年 2 月 19 日
柳原 正

参考文献

[1] James Gosling, Bill Joy, and Guy Steele. “The JAVA Language Overview”, 1998.

[2] “The Working Group for Wireless Lans”. http://grouper.ieee.org/802/11/.

[3] “Asymmetric Digital Subscriber Line”. ANSI Standard T1.413-1998.

[4] Ed. M. St. Johns. “Community Antenna Television”, 1999. RFC 2670.

[5] “ANSI X32T12 (FDDI)”. http://www.nswc.navy.mil/ITT/x3t12/FDDIFAQ.html.

[6] “ICQ: I Seek You”. http://web.icq.com/company/about.html.

[7] “mp3.com”. http://www.mp3.com/.

[8] “Google”. http://www.google.com/.

[9] V.S. Sunderam. “PVM: A Framework for Parallel Distributed Computing”. In *Concurrency: Practice and Experience*, Vol. 2-4, pp. 315–339, 1990.

[10] “Popular Power”. http://www.popularpower.com/.

[11] Bernd O.Christiansen, Peter Cappello, Mihai F.Ionescu, Michael O.Neary, Klaus E.Schauser, and Daniel Wu. “Internet-Based Parallel Computing Using Java”. In *ACM Workshop on Java for Science and Engineering Computation*, 1997.

[12] 高木浩光, 松岡聡, 中田秀基, 関口智嗣, 佐藤三久, 長嶋雲兵. “Ninflet: Java による World-Wide High Performance Computing 環境”. In *Internet Conference*, 1997.

[13] 藤村浩, 中澤仁, 大越匡, 徳田英幸. “動的分散ソフトウェアツールキット MetaSpace の設計と実装”. 情報処理学会システムソフトウェアとオペレーティングシステム研究報告, 第 81 巻, pp. 119–124, 1999.

[14] “Napster”. http://www.napster.com/.

[15] “Gnutella”. http://gnutella.wego.com/.

[16] “Freenet”. http://freenet.sourceforge.net/.

[17] Ronald L. Rivest. “The MD5 Message-Digest Algorithm”.

[18] National Institute of Standards and Technology. “The Secure Hash Algorithm (SHA-1)”.

[19] Audio-Video Transport Working Group. “RTP: A Transport Protocol for Real-Time Applications”, 1996. RFC 1889.

[20] “Project CSC”. http://www.distributed.net/csc/.

[21] “Multimedia and microKernel Group”. http://www.mkg.sfc.keio.ac.jp/.