JAIST Repository
https://dspace.jaist.ac.jp/

Title	モバイルエージェントを用いたサーバ環境監視方式に関する研究
Author(s)	清水，雅司
Citation	
Issue Date	2003-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/1683
Rights	
Description	Supervisor:敷田 幹文，情報科学研究科，修士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

修 士 論 文

モバイルエージェントを用いた
サーバ環境監視方式に関する研究

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

清水 雅司

2003 年 3 月

修 士 論 文

モバイルエージェントを用いた
サーバ環境監視方式に関する研究

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

110058 清水 雅司

提出年月: 2003 年 2 月

Copyright © 2003 by Masashi SHIMIZU

目 次

図 目 次

表 目 次

第1章　はじめに

本章では本研究の背景，目的を述べ，最後に本論文の構成について述べる．

1.1　本研究の背景

近年，コンピュータの普及，及びネットワーク技術の進歩によってネットワークシステムの大規模化が進んでいるが，それにつれてそのネットワークシステムの構成自体も複雑なものへとなっており，今後もこの大規模・複雑化の傾向は続いていくと思われる．また，学校や企業でもこれらのネットワークシステムを用いて教育・業務が日常的に行われており，ネットワークファイルシステムやメール，WWW などの基幹ネットワークサービスの可用性を確保する必要が出てきた．そして，必然的にそれらのサービスに対して非常に高い信頼性が求められるようになってきた [1]．この信頼性を保つために，ハード・ソフト両面で冗長性を実現するだけでなく，万一障害が発生した場合には，その障害を有効的に検知するシステムの設計が急務である．

監視システムの設計にあたって，監視対象であるサーバ及びサーバが提供するサービスの大きな特徴を考慮すべきである．それらの特徴とは多様性と頻繁なシステム変更・更新である．よって監視システムは多種多様なものに対応する必要があり，かつ，頻繁に更新される可能性がある．今後も提供されるサービスは増えていく事が予測できそれに伴って監視対象や監視項目が増加することは明らかである．従来の監視方式を今後のネットワークシステムでも運用していくことは困難であると考えられる．単純に監視にかかる通信量が大きくなるという問題だけでなく，システム自体の管理性で深刻な問題が起こる予測できる．以上のことを踏まえて，円滑なネットワークシステム運用を考える上で，その監視システムの管理性は大変重要になってくる [2]．現在，管理性に最も優れたネットワークシステム構成の一つとして基幹サーバ群を高性能な大型ルータに広帯域なネットワークで局所的に集中させたものが注目されつつある．このように，ネットワークシステムは大規模分散化する一方で，管理に関しては集中する傾向にある．

また，分散オブジェクト技術であるモバイルエージェントが我々の身近なソフトウェアとして存在するようになり，誰もが数あるシステムから選び，比較的容易に利用できる環境が整ってきた．そして，近年の研究成果により，モバイルエージェントの標準仕様化や悪意のあるホストやエージェントからの保護などのセキュリティ対策が講じられるようになってきており，適応分野として電子商取引やコミュニケーションなどで用いられる事が期待されるようになってきた．

1.2 本研究の目的

本研究では分散オブジェクト技術であるモバイルエージェントに注目し，その技術を用いて，大規模分散ネットワーク上で運営されている基幹サーバ群に対して集中管理に適した監視を行う方式を提案する．そして，その提案方式のプロトタイプシステムの実装を行い，評価実験を通して監視ツールとしての基本性能の検証を行う．また，既存方式との比較を行いながら監視システム自体の管理性などについても提案方式の有効性について評価する．

1.3 本論文の構成

ここでは今後の本論文の構成を紹介する．2 章では大規模分散システム及びそれらの管理形態について概要を述べる．そして，3 章で既存の監視方式として SNMP を用いた監視方式，常駐型エージェントを用いた監視方式とモバイルエージェントを用いた監視方式についてを例に挙げ，それらの利点及び，問題点について述べる．4 章では大規模なネットワークシステムにて有効であり，管理性の向上と通信量の低減などを目的としたモバイルエージェントを用いたサーバ監視方式を提案し，続く 5 章でその提案方式のプロトタイプシステムの実装について述べる．6 章でプロトタイプシステムを用いた評価実験について説明し，7 章でそれらの結果より議論を進めていく．最後に 8 章でまとめと今後の課題について述べる．

第2章　大規模分散システム

本章では大規模分散システムについての基本概念と現状などを述べる．

2.1　大規模分散システムとは

一般に数千台以上のネットワーク機器が接続されているようなネットワークシステムのことを大規模システムと呼んでいる．近年このような大規模システムが増加している．これに伴って，部署やフロアごとに複数のサブネットワークが形成され，クライアントPCやサーバ群などが各サブネットワークに分散して配置されるようになった．これらのサブネットワークは自律的に運用を行うことが可能である．このような複数のサブネットワークに計算資源が点在する大規模なネットワークシステムを本稿では大規模分散システムと位置づけている．

大規模なシステムを分散させることによって，各サブネットワーク単位でのシステムの構築，運用・管理が可能になる．機能別にネットワークを細分化することで無関係なサブネットワークへの無駄な通信を無くすことができる．また，万一の障害時でも，障害をそのサブネットワーク内で食い止めることができ，原因の特定の範囲が狭まることで，復旧が迅速に行える．一方で，これら分散させた機器類についての管理が大変難しくなっている．

2.2　大規模分散システムでのシステム管理業務

大規模分散システムでの管理業務について，現状，コスト，管理法などについて以下で述べる．

2.2.1　現状

多くの企業・学校がインターネットを利用したサービスを提供するようになった現在，それらサービスを提供するサーバには，速度，安全性，確実性の三つの要素が不可欠とされる．これらの条件を満たし，システムを24時間安定稼働させることが求められている．しかし，システム全体が正常にサービスを提供し続けるためのシステマチックな作業を確立することは大変難しい．

このような背景下，基幹サーバなどの管理は全て一カ所で集中して行う管理方式が注目を浴びるようになってきた．例として一カ所で集中的にサーバ群を監視する Internet Data Center（以下 IDC）を挙げることができる．IDC とは E コマースや ASP 事業などでサービスを提供する重要なサーバ群が配置されるサーバルームである．この部屋にはセキュリティ機能もあり入退室は厳しく管理される．また，そのバックボーンとなるネットワークは高帯域であり，高機能なネットワーク機器により構成され，かつ冗長化されている．一般的にネットワーク機器の冗長化だけでなく，電源設備も多重化され，耐火性や耐震性に優れている．そして，集中コンソールにおいて 24 時間 365 日体制で監視が行われている．

IDC のように一カ所に集めて管理することで安全かつ管理コストを低減が可能である．また，このような管理業務を専門の業者に完全に委託することで自らに運用ノウハウや安全で強固なシステムを構築する必要がなく，その需要はますます高まっている．

2.2.2 コスト

単純で小さなネットワークに比べてその管理コストは大変大きい．ここで扱う管理コストとしては主に人的なもの，時間的なもの，ネットワーク的なものについて考える．複雑な設定や高度な技術を用いたシステム構成の場合，それらを熟知した技術者の数が少ないという問題がある．複数箇所そのような環境があった場合，それら全てに熟練技術者を配置することは難しい．結局，一カ所で集中管理を行う必要がある．また，サービスプロセスのバージョンアップやインストールなど比較的単純な管理作業であった場合においてもその管理にかかる時間は一般的に管理対象機器の数に比例して増加し，変更が反映するまでの時間が長くなってしまう．そしてネットワークへの流れる管理にかかる通信量は監視対象が多ければ多いほど大きくなってしまう．

2.2.3 分散管理／集中管理

管理法には大きく分けて分散管理と集中管理がある．

- 分散管理

  各部署・フロアなどで分かれたサブネットワークで管理を行う方法である（図 2.1 参照）．理想的にはそれらサブネットワーク単位でネットワーク管理者が配置され運営管理がなされていることが望ましいが，現実はコスト的な問題で一部の管理者が全体のネットワークの管理運営を行っている．そのため，各サブネットワークごとの情報を集約するシステムを構築し，統合する必要が出てくる．分散管理に関して，協調作業を含んだ管理の支援 [3]，管理情報の共有 [4]，分散資源の統合 [5] に関する研究なども行われている．

- 集中管理

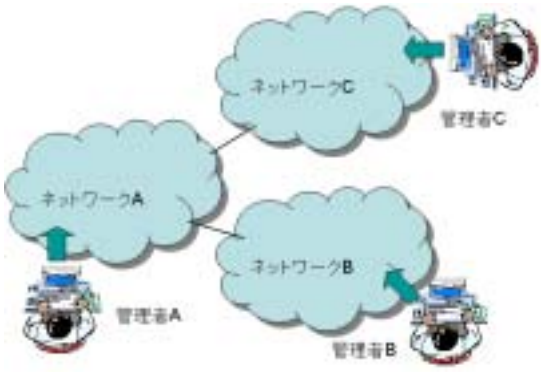

図 2.1: 分散管理の概観図

そのネットワークにおける全ての管理業務を一元的に行う方法であり，管理コストを低減することが可能である（図 2.2 参照）．上述した IDC のような管理方法も集中管理の一例である．ただしこれは管理対象を全て管理するための専用のネットワークを別途用意して，その一カ所で集中して管理を行っている方式であった．分散された管理対象機器に対して集中管理を行う際にはもっと複雑な管理業務になってしまう．分散された機器の監視では，管理者は対象となるネットワークの構成や機器だけでなく，それらのサブネットワークや各サーバの管理ポリシーなど管理上で必要な情報全てについて把握している必要がある．また，管理システムが非常に多くの監視対象となる機器を扱うために，一般的にはその管理システム中枢部に通信が集中してしまうといった問題もある．しかし，管理機器を自在に配置することによってスケーラビリティを向上させることができ，より柔軟な管理が行うことができることは明確であり，それらを有効的に管理する方法の確立が望まれている．現在，管理コストの低減が可能な集中管理方式は注目を浴び，それに関連して一元管理の省力化 [6] や VLAN を用いた運用ポリシーの適応 [7] などの研究が行われている．

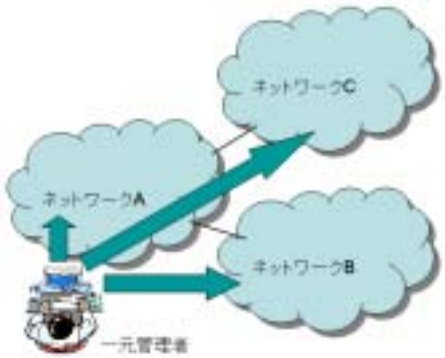

図 2.2: 集中管理の概観図

第3章　既存の監視方式

本章では既存の監視方式をとりあげていく．従来の監視方式として SNMP 方式と常駐型エージェント方式，モバイルエージェントを用いた方式を順に，それぞれの概要，利点，問題点を述べる．

3.1　SNMP 方式について

主にネットワーク機器の監視に一般的に広く使われている SNMP 方式による監視について述べる．

3.1.1　概要

集中監視型の SNMP 方式はマネージャ（監視ホスト）が複数のエージェント（監視対象機器）に対して定期的に 1 対 1 でポーリングを行うことによって，もしくは何らかの異常を検知してエージェント側からのトラップ機能を使うことによって，情報を取得し，その情報から障害を判断するというシンプルな構成で成り立っている（図 3.1 参照）．またネットワーク機器からの値の取得だけでなくネットワーク機器へ値を入力することも可能である．これらの各値は Management Information Base（MIB）と呼ばれる階層型のデータベースのオブジェクトとして取り扱い，OID と呼ばれる一意な識別子で表すことができる．現在，監視ツールのデファクトスタンダードとなっていおり，幅広く利用されている [8]．Sun Management Center 3.0 や net-snmp などが挙げられる．

3.1.2　利点

一つの MIB オブジェクトを取得するのには約 150〜200bytes の通信量が必要であるが，この値は現在のネットワーク事情を考えると非常に小さいものである．そして，各フィールドに対しての問い合わせに値を返すという簡素な仕組みであるため，他のアプリケーションからの利用が容易であり，それ単体の動作は非常に軽い．

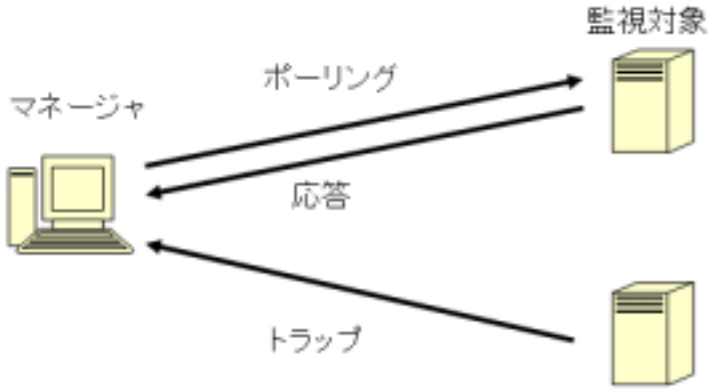

図 3.1: SNMP 方式の概観図

3.1.3 問題点

ネットワークの複雑化・大規模化に伴い，様々な問題が生じてきた．以下に列挙する．

- **マネージャへの通信の集約**

  マネージャは各監視対象機器に対して往復の通信路が必要であり，全ての通信がマネージャへと集約されるためにマネージャにかかる通信の負担が非常に大きい．

- **機器及び機器が持つ情報量の増加**

  ネットワークの大規模化に伴いそれを支えるネットワーク機器の数も多くなっている一方で，その各機器が持っている情報自体も多くなっている．そのため，取得する MIB のオブジェクト数は増加傾向にあり，SNMP での監視にかかる通信量も比例して増加する傾向にある．

- **監視システムの階層化**

  マネージャへの通信の集中を和らげるためにマネージャを複数設置すると数多くの監視ネットワークができてしまい，それらを統合するための専用のマネージャを配置する必要があり，監視システムが階層化してネットワークシステムのスケーラビリティを低下させてしまう．

- **ベンダ差**

  各ベンダ間で独自機能があり，ベンダ差をうまく埋めることが難しい．またベンダ差による監視の可否といった深刻な問題もある．

- **簡素な監視**

従来ネットワーク機器を主な対象としているため基本的には単純な値だけに依存した監視となっている．そのため，サーバ監視のような複雑な監視業務を行う場合，SNMP 単体では監視が困難であり，別途専用のプログラムを必要とする．複雑な監視を行う場合，一般的に後述する常駐型エージェントを用いる．

- **総監視プログラム数の増加**

  複雑な監視を行う際は，専用のプログラムを各監視対象にインストール，チューニングする必要があり，その管理コストがネットワーク規模に応じて大きくなる．

- **障害把握の難しさ**

  上述した複雑な監視プログラムが別途必要な場合，別システムを無理やり SNMP 方式に取り入れる結果となり，結果としては SNMP のトラップで簡単なメッセージが届くだけであり，障害が起ったときのトレースが複雑な作業となってしまう．

- **サービスの経路監視**

  様々なサービスを提供するサーバの監視においてネットワーク機器単体がそれぞれ正常であったとしても必ずしもユーザが正しくサービスを受けることができるとは限らない．正確なサービス状況を把握するためにはクライアント側からのサーバへの経路調査も必要である．

3.2 常駐型エージェント方式について

各監視対象に専用の監視エージェントが常駐する監視方式について述べる．

3.2.1 概要

サーバ監視など複雑な監視を行う場合，前述した SNMP 方式だけでは非常に実現が困難でありその構成も複雑なものとなってしまう．そのような複雑な監視において用いられる方式であり，各監視対象にその監視を行うエージェントを設置し，常駐させる．そして主に障害を検知した場合にのみマネージャへと報告する．複雑な業務を行う監視エージェントはそのプログラム自体が大規模である．また，SNMP 方式をシステムの中で併用する形態が多く見られる．監視業務に関しては専用の常駐型エージェントが行い，障害を検知するとその情報を MIB オブジェクトとして扱い，SNMP の機構を用いてマネージャへと通知するという手法が一般に採られている．常駐型エージェント方式一例として監視統合フレームワークである JP-1 の IntegratedManager と Base を用いた仕組みが挙げられる [9, 10].

3.2.2　利点

普段の監視では監視対象に常駐したエージェントプログラムが監視業務を行うためにネットワーク上に通信が起きない．ネットワーク全体に流れる通信量が極めて小さい．また，専用の特別なエージェントを配置するため，複雑な監視作業を行うことができる．

3.2.3　問題点

- マネージャへの通信の集約

  全ての監視対象機器がマネージャへ通信を行う．そのためにマネージャへの通信が監視対象数に比例して大きくなってしまう．

- エージェント状況の把握

  通常，マネージャに報告がないため，各監視対象上で独立に動いているエージェントの動作を把握することが難しく，そのエージェントに対してポーリングを行い状態を確認する必要がある（図 3.2 参照）．

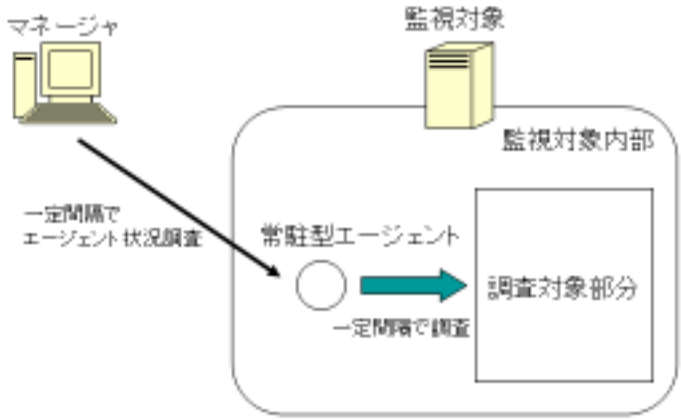

図 3.2: 常駐エージェントの状況把握

- 総プログラム数の増加

  専用のエージェントプログラムが必要であり，各監視対象にそれらを配置しなければならず，それらを管理するコストも高い．また，マネージャ側からエージェントの振る舞いを変えるなどの変化を行うことに非常にコストがかかり，監視システムを集中監視型として運用することが難しい．ネットワークシステムが大規模分散化する傾向にある昨今において，管理性の問題がこの方式の最大の問題である．

3.3　モバイルエージェント方式について

自律的に移動を行うエージェントを用いた監視方式についての既存研究について述べる．

3.3.1　概要

従来の SNMP による監視方式では非常に多くの MIB オブジェクトを通信によってマネージャ側で受取り，その受け取った値からマネージャが障害を判断している．しかし，モバイルエージェントを用いた監視方式では基本的に通信はエージェント自体の移動だけであり，さまざまな値はその移動先の監視対象上でエージェントが参照することができる．そして，その後障害判断をエージェントが行う．

3.3.2　利点

必要な結果だけを集約してマネージャに報告する事で大規模なデータを扱う場合においては管理にかかる通信量の低減を実現している [11]．そして，管理性が非常に向上する [12]．例えば，管理しているネットワークシステムに変更があれば，従来方式で複雑な監視を行っている場合，監視対象に専用のソフトウェアを別途インストール，チューニングしていなければならず，それらのソフトウェアを必要に応じてチューニングしなければならない．しかし，モバイルエージェントを用いた監視方式では移動するエージェントの障害調査プログラムの移動先や監視項目などの変更を行うだけでよく，監視対象サーバに対しては何ら手を加える必要がない．結果としてシステムのスケーラビリティも向上させることができる．集中監視に適した監視方式だと言うことができる．

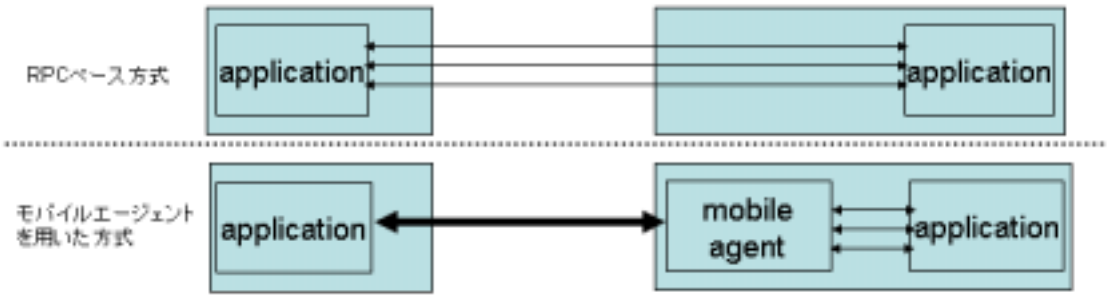

図 3.3: 方式による通信の違い

3.3.3 問題点

既存のモバイルエージェントを用いた監視方式にはいくつかの問題点もあり，以下にそれらを挙げる．

- **エージェント自身に対する障害対策**

  自律的に動き回るモバイルエージェント自身の位置を把握することが難しく，エージェント自身に対して何らかの障害が起ってしまった場合，そのエージェントの障害が監視システム全体に影響してしまう．

- **エージェント移動による障害情報の遅延**

  エージェントが障害情報を格納し，マネージャまで移動して報告するといった方法では，逐次変化していく障害状態を把握することが難しく，また障害情報の遅延にもつながってしまう．

第4章　モバイルエージェントを用いた提案方式

本章ではモバイルエージェントを用いたサーバ監視方式を提案し，その設計方針や概要及び，各エージェントの働きやモードについて述べていく．

4.1　設計方針

まず，サーバの監視システムの設計を行うにあたって，その監視システム自体が堅牢なシステムであり，かつ障害が迅速に検知できる事が大前提である．

また提案方式ではモバイルエージェントを用いるために，その利点と問題点について考慮しながら設計を進める．エージェントを用いる最大の利点は，監視対象上で様々な作業を柔軟に行う事ができる事である．この利点と下記の問題点との折り合いを考慮する必要がある．

モバイルエージェントを利用するにあたって最も大きな問題は通信量の大きさである．SNMP を用いた MIB オブジェクトの取得にかかる通信量と比較するとモバイルエージェントの移動にかかる通信量は大きいものである．そこで通信量の低減を実現するために以下の点に留意した．

- エージェント自体のサイズを小さくする

  エージェント自体のサイズに比例して通信量は増加してしまうために必要最低限の機能だけを持ったエージェントにする．

- エージェントが格納する情報は少なくする

  各監視対象上で情報は参照するだけにし，格納する情報はそれらの値から得られる結果をまとめて必要な情報のみ格納する．

- 機能別にエージェントを分ける

  一つのエージェントのサイズを減らして目的別に使い分ける．

- 可能な限りメッセージ通信を使わない

  通常時はメッセージ通信を行わず，障害時など必要となった場合のみメッセージによる通信を行うことで通信量を抑える．

4.2 提案方式の概要

機能別にモバイルエージェントを複数用いてそれぞれを協調作業させることによってサーバの監視を行う．またサーバ単体だけでなく，その周辺のネットワークの監視としてクライアント側からの経路調査を併せて行う．

監視エージェントとして3種類設け，それらのエージェント及びエージェントの持つ情報を管理する1種類のマネージャを設けた．具体的に監視作業を行うエージェントは表4.1の通りである．

表 4.1: 各監視エージェント

定期巡回エージェント	サーバのサービス監視
集中監視エージェント	異常検知時の監視を行う
経路監視エージェント	サーバの経路状態監視

4.3 提案方式の特徴

モバイルエージェントを用いた提案方式では大きな特徴として，通信の分散と総プログラム数の抑制，そして変化への柔軟な対応が考えられる．

従来の監視方式では監視システム中枢ネットワークに集中し，大きな問題となっている．提案方式ではモバイルエージェントがマネージャと監視対象との1対1の往復の通信路だけを利用するのではなく，監視対象同士間のネットワーク全体を通進路として利用し，監視にかかる通信が分散し，結果として監視システムにかかる通信の負担は少なくなる．

また，ネットワークシステムの規模に応じて管理性の向上への恩恵は大きくなる．総プログラム数を抑える事で管理性が大きく向上し，監視システムの変更や日常のメンテナンス作業において有効である．

そして，エージェントが動くことによって従来方式よりも変化により柔軟な対応が可能であり，障害状況にあった監視方法を選ぶことができる．

既存のモバイルエージェントを用いた監視システムには無かったエージェント損失時の保証機構を設け，監視システムの信頼性を向上させた．また，障害情報だけは検知から通報までの時間を縮めることが可能できる．

4.4 提案方式の構成

ここでは提案方式で用いられる４つのエージェントとマネージャの働きを述べていく．

4.4.1　定期巡回エージェント

「障害の一次検知」

巡回すべき監視対象のサーバリストをマネージャから受取り，それを見ながら自律的にサーバ群を順番に巡回していって調査を行う．その際の移動ではマネージャと各サーバ間の往復を避けて，巡回による移動を行う．各サービスの存在や応答，CPU・メモリ・NIC などの負荷を監視項目とし，異常を検知した場合はその旨をマネージャへと報告する．報告を受けたマネージャは即時該当サーバへ後述する集中監視エージェントを送信する．その後，完全に復旧するまでその集中監視エージェントにそのサーバに関する全ての監視業務を任せる．集中監視エージェントが障害の起ったサーバを監視している間は定期巡回エージェントはそのサーバへは移動しない．正常な情報は格納して移動を続け，マネージャに戻った時に報告する．また，応用として MAC アドレスまでも調べることで例えば HA システムなど従来では障害検知が難しい複雑なシステムにおいても障害の検知を行うなどの処理も可能であると考える．定期巡回エージェント動作の概観図を図 4.1 に示した．

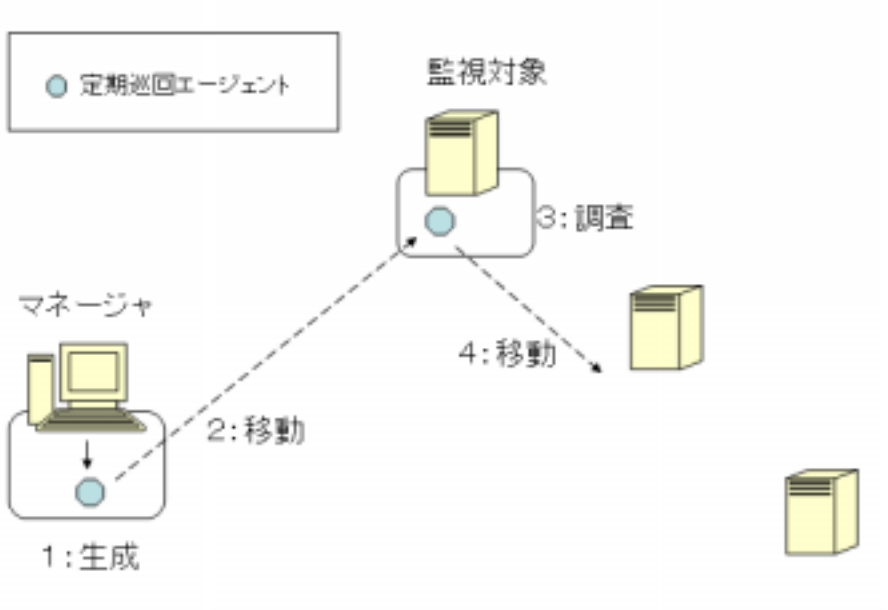

図 4.1: 定期巡回エージェントの概観図

4.4.2　集中監視エージェント

「障害状況の把握」

自らは移動機構を持っていない，常駐型のエージェントである．マネージャによって指定監視対象先へと送出される．移動先のサーバ上で障害状況を監視し，改善・悪化などの

変化があればマネージャへ通信を行って報告する．サーバ上に常駐するために定期的なポーリングでネットワーク上にトラフィックが発生せずに，通信量を抑えることができ，かつ，障害の変化を逐次マネージャへと伝えることができる．また状況に応じてポーリング間隔を変えることでより障害の検知を早めることも可能である．完全にサーバが障害から復旧すれば役目を終え消滅する．

4.4.3 経路監視エージェント

「クライアント側からの調査」

マネージャからルータリストと経路表を受け取り，ルータリストを見ながらクライアントとサーバ間でクライアント側に近いルータやゲートウェイに順次移動して，そこから各監視対象サーバへ問い合わせを行う．このエージェントも巡回移動しながら監視を行う．受け取った経路表を元に，実際にそこからクライアントがサーバまで正常な経路で，かつ，指定時間内に通信を行うことができるかどうかを調べる．もし，異常を検知した場合は即時にマネージャへ通信を行って報告する．一方，正常な値はエージェントが保持して一巡した後，マネージャへと戻り正常値を報告する．

4.4.4 マネージャ

「障害情報の収集」

まず，定期巡回エージェントと経路監視エージェントを生成し，自分の持っているサーバリストとルータリスト・経路表をそれぞれに渡す．そして，それらのエージェントを生成した後は一カ所に留まり，各エージェントからの報告によってログの収集を行い，障害情報を管理者へ知らせる．

サーバリストには巡回すべき監視対象サーバと，その各サーバの基本情報として，そのサーバの OS 名，サービスしているプロセス名，サービスプロセスのバージョンが記述されている．一例を表 4.2 に記した．このリストを元に定期巡回エージェントはそのサーバに適した行動をとる．

表 4.2: 各サーバの基本情報例

Host	OS	Process	Version
nameserver	Solaris	bind	9.2.1
www	FreeBSD	apache	1.3.26

ルータリストには巡回すべきルータ・ゲートウェイが記述されており，経路表はそのルータリストに記載されている各ルータ・ゲートウェイからの各監視対象への経路及び障

害を判断する応答時間の閾値が記述されている．

正常情報は少しの遅延であれば影響はないものと考えられる．そのため，通信量を考慮して各監視対象に対する正常報告は定期的にエージェントが移動でマネージャに戻ってきた時にのみ行われ，障害報告は障害を検知したら即時に各エージェントからマネージャへ行われる．よって，障害報告だけは可能な限り早くマネージャは知ることができる．

4.5 追跡モード

監視システムの信頼性を上げるために，モバイルエージェント自身が損失した場合の保証機構を設ける．

通常時においては各監視エージェントは移動しながら監視を続け，巡回リストに要素が無くなればマネージャへと戻ってくる．このエージェントのマネージャへの帰還が監視システムが正常に動作している保証であった．

そこで，通常時において送出したエージェントが一定時間内に戻ってこなかった場合についての対策として追跡モードを取り入れる．追跡モードではメッセージ通信を行ってエージェントの位置を把握させる．このため，通常時と比べ監視にかかる総通信量は多くなってしまうが，障害の発生箇所をつきとめることを最優先することにする．

追跡モード時の定期巡回エージェント及び経路監視エージェントは各サーバ／ルータへ移動する度にマネージャへ自分の位置を報告し（図 4.2(1)）マネージャはそのデータベースを更新する（図 4.2(2)）．もし一定時間内にエージェントが戻って来なかった場合はマネージャはそのデータベースを用いて問い合わせを行う（図 4.2(3)）．反応が無かった場合，その位置のサーバ／ルータに集中監視エージェントを送信し（図 4.2(4)），巡回リストからそのサーバ／ルータを除いてエージェントを再生成する．一定時間内にエージェントが戻ってきた場合は，マネージャは通常モードへと切り替わる．

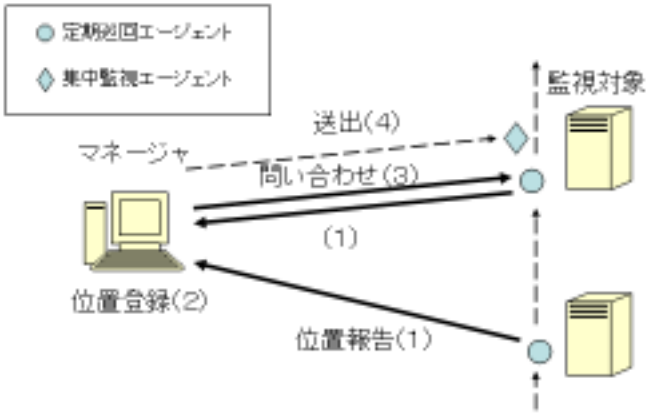

図 4.2: 追跡モード時の位置報告

第5章　評価システム

本章ではまずプロトタイプシステムの実装環境について述べる．そして次に，その実装に用いたバイルエージェントフレームワークである Aglets の概要を述べる．その後，プロトタイプシステムの概要，各エージェントの動作の細部や移動部分の実装，エージェント間通信について述べる．

5.1　実装環境

サーバサイドのアーキテクチャとして多種多様なものを想定する必要があることを考慮して，エージェントプログラムは Java による実装を行う事に決めた．また，モバイルエージェントプログラムは Java で開発されたものが多く，今回は IBM 東京基礎研究所が開発し，IBM public license の下，配布している Java の API 群と実行環境を含めた Aglets Software Development Kit(ASDK) version 2.0.2 を利用した [13]．また評価システムのプロトタイプシステムの実装は Solaris8 上で Sun Microsystems が提供する JDK1.3.0 を用いて行い，そのテストは Solaris8，FreeBSD4.7，Linux2.4 上で ASDK 付属の Aglet サーバである Tahiti version 1.0b5 を利用して行った．

5.2　Aglets の概要

ここではモバイルエージェントの概要やプロトタイプシステムの実装に用いる Aglets の概要と仕組み，イベントについて述べていく．

5.2.1　モバイルエージェントとは

エージェントとは実行環境下において目標を達成するために自律的に，かつ反応的に動くプログラムの事であり，モバイルエージェントとは実行を開始したシステムに束縛されないエージェントである [14].

また，一般的にモバイルエージェントは次の性質を持つ．

- 移動性

  ネットワーク上を移動しながら処理ができる性質

- 自律性

  エージェント自らが決定し，処理ができる性質

- 協調性

  他のエージェントと通信し，協調作業ができる性質

- 適応性

  移動先の環境に応じて処理ができる性質

- 局所性

  移動先の情報に基づいて処理ができる性質

5.2.2 Aglets について

Aglets は IBM が開発した Java 言語で記述されたモバイルエージェントのためのフレームワークである．Aglet 自身は移動できる実行可能な Java オブジェクトであり Aglet サーバがそれをインスタンス化することでタスクを実行している．ASDK で提供されるクラスライブラリを利用することで，プラットフォーム非依存なモバイルエージェントプログラムを比較的容易に作成することができる．

また，開発環境として ASDK ではエージェントの生成，削除，などエージェントの制御のための Tahiti も用意されており，ネットワーク上をを動き回るエージェントの制御を比較的簡単に行うことができる．

エージェント認証による悪意のあるエージェントからの移動先コンピュータへの攻撃に対する保護やセキュリティドメイン認証による悪意のあるコンピュータからエージェントへの攻撃に対する保護などのセキュリティ対策がなされている．

Java の高い移植性のためにほとんどの計算機上で Aglets を実行でき，モバイルエージェントシステムとして現在も幅広く利用されている．

Aglets 以外のモバイルエージェントシステムとして代表的なものを表 5.1 に挙げた．これらのモバイルエージェントシステムの相互作用を促進させる目的で ObjectManagement-Group の MASIF がモバイルエージェントの標準仕様とされている．

5.2.3 Aglets の用語，仕組み

Aglets によるモバイルエージェントシステムを理解する上で重要な用語と仕組みを述べる．

- Aglet

表 5.1: 他のモバイルエージェントシステム

AgentSpace	研究開発用フレームワークを目的にお茶の水女子大佐藤一朗氏が開発．Java ベース．
Plangent(picoPlangent)	東芝研究開発センターで開発．Java ベース，エージェントは独自言語で記述．現在，picoPlangent へと移行．
Telescript	General Magic 社による世界初の商用ソフトウェア．独自のオブジェクト指向言語 Telescript で記述．
Oddyssey	General Magic 社が Telescript のコンセプトを Java で実装したもの．
Voyager	ObjectSpace 社による商用ソフトウェア．Java ベース．

Aglets システムにおいてネットワーク上を自律的かつ反応的に動く Java オブジェクトのことである．また，全ての Aglet は実行環境として下記の AgletContext 内にて動作する事ができる．

- AgletProxy

  各 Aglet に対して 1 対 1 の関係であり，その Aglet に関する全ての情報を管理する．他の Aglet やアプリケーションからの全てアクセスを仲介し，Aglet を保護する．AgletProxy は必ずしも Aglet と同じホストに存在する必要がないため，Aglet の位置透過性を実現する．

- AgletContext

  Aglet の実行環境そのものである．Aglet 環境変数を元にコードベースやクラスを管理し，それらを用いて Aglet を生成する．また，Aglet の消去や移動，同一コンテキスト内の通信なども行う．基本的には，一つの Aglet サーバに対して一つ以上の AgletContext がある．

5.2.4 Aglet のイベント

Aglet の動作は所定のタイミングで呼び出されるコールバックメソッドにより制御されている（表 5.2）．Aglet のイベントは，生成・消去・送出・撤収・活性化・非活性化の大別できる．以下にそれらの各イベントについて Aglet の実行手順を記述する．

生成

Aglet の生成は AgletContext 内にて行われる．また生成には一から作り出す方法と既存のエージェントの複製をとる方法の二通りある．どちらの方法も Aglet は生成されると一意な AgletID を与えられ，これにより識別が行われる．一から新たにエージェントを作り

表 5.2: コールバックメソッド

メソッド名	実行される時
onCreation()	create() による生成直後
onCloning()	clone() による複製直後（オリジナル側）
onCloned()	clone() による複製直後（複製側）
onDisposing()	dispose() による消去直前
onDispatching()	dispatch() による送出直前
onArrival	移動先へ到着直後
onReverting()	retract() による撤収直前
onDeactivating()	deactivate() による待避直前
onActivating()	activate() による復元直後

出す場合，Aglet は createAglet() メソッドにて生成される．複製する場合，複製対象となる Aglet が clone() メソッドにて自分を複製する．

消去

自分自身，もしくは他のエージェントなどから dispose() メソッドにて処分され，完全に AgletContext 内から消去される．

送出

dispatch() メソッドにより，Aglet は現在の AgletContext から移動先の他の AgletContext へと移り，実行環境を変える．

撤収

retractAglet() メソッドにより他の AgletContext から撤収要求を受けるとその要求先の AgletContext に移動する．

ディスクへの待避と復元

deactivate() メソッドが呼ばれると Aglet は一時その実行を停止し，その状態をディスクに待避する．また，このディスクへ待避した Aglet を activate() メソッドにて活性化させることができる．

5.3 システム構成とエージェント動作

前章で述べたマネージャと各エージェントなどと実際のクラスとの対応は下表 5.3 の通りである．

Manager の実行手順

表 5.3: 各クラスファイルの役割

Manager	マネージャ
Patrol	定期巡回エージェント
TracedPatrol	定期巡回エージェント（追跡モード）
RtCheck	経路監視エージェント
Inspector	集中監視エージェント
ServerList	サーバリスト
RouterList	ルータリスト
RouteTable	経路表
SeqItinerary	移動制御

1. getInfoFromFiles() メソッドにて巡回すべきサーバとそのサーバが提供するサービスプロセスを取得し，それらを引数にして ServerList を生成する．
2. ServerList を引数にして createAgent() メソッドにて Patrol を生成する．
3. getInfoFromFiles() メソッドにて巡回すべきルータとそのルータから各サーバへの経路表を取得し，経路表を RouteTable クラスに格納，先に作った ServerList と RouteTable を引数に RouterList を生成する．
4. RouterList を引数にして createAgent() メソッドにて RtCheck を生成する．
5. 待機し，エージェントからメッセージ通信があった場合は handleMessage() メソッドによりにメッセージに応じた処理を行う．
6. 通信があった場合はその記録を logStatus() メソッドにてファイルへと保存する．

Patrol の実行手順

1. ServerList を引数にして SeqItinerary を生成，SeqItinerary に従い移動を試みる．
2. サーバへ到着すると run() メソッド内にて OS 付属の ps コマンドを用いてプロセステーブルより該当サービスの存在を確認する（TracedPatrol の場合，サービスの存在確認の前にメッセージ（Kind は LCREPORT）を用いて到着したサーバの位置をマネージャに報告する）．
3. checkDaemon() メソッドにて該当サービスの応答を確認する．
   今回対象としたサービスプロセスは sendmail と httpd である．近年ではどちらも多くのユーザに利用される基幹ネットワークサービスの一つである．
   sendmail デーモン調査
   監視対象上で port25 番で tcp での通信を行って sendmail デーモンからの応答をチェックする．成功コードが返ってくれば正常とみなす．
   http デーモン調査
   監視対象上で port80 番で通信を行い，http デーモンからの応答をチェックした後，指

```
#telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 is14e9u99.jaist.ac.jp ESMTP Sendmail 8.10.2+Sun/8.10.2;
Thu, 30 Jan 2003 23:59:00 +0900 (JST)
```

図 5.1: Sendmail デーモンの応答

定 URL をの HEADER の取得を試みる．成功コードが返ってくれば正常とみなす.

```
#telnet localhost 80
Trying 150.65.5.208...
Connected to localhost.
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 30 Jan 2003 23:59:00 GMT
Server: Apache/1.3.27 (Unix) mod_ssl/2.8.12 OpenSSL/0.9.6g
Last-Modified: Tue, 02 Apr 2002 23:59:00 GMT
ETag: "2042e-45a-3ca92c9f"
Accept-Ranges: bytes
Content-Length: 1114
Connection: close
Content-Type: text/html

Connection closed by foreign host.
```

図 5.2: http デーモンの応答

4. OS 付属の uptime コマンドを用いて CPU 平均負荷率を調査する．1 分前，5 分前，15 分前の CPU の平均負荷率を調べ，これらの値が指定した閾値を越えていれば異常とみなす.
5. 異常があればマネージャへ報告し，なければ時刻と正常情報を Vector クラスの normalReport へと格納する
6. SeqItinerary に従って次のサーバへと移動する.

RtCheck の実行手順

1. RouterList を引数に SeqItinerary を生成して，移動を試みる.
2. ルータへ到着後，checkRouteTable メソッドにて OS 付属の traceroute コマンドを用いて各サーバへの経路と RouteTable の内容と比較する．もし違っていればメッセージ通信によって経路異常報告をマネージャへ行う.

3. OS 付属の ping コマンドを用いて 3 回 ICMP の ECHO リクエストを送り，その応答の ECHO リプライの平均応答時間を計り，規定内の秒数であれば正常情報を Vector クラスの normalReport へ追加する．もし異常が確認できれば，メッセージ通信によってサーバへの到達時間が遅い事をマネージャへ知らせる．
4. SeqItinerary に従って次のルータへと移動する．

Inspector の実行手順

1. Manager の orderInspector() メソッドから呼ばれた createAgent() メソッドにて生成され，引数として受け取った URL に自分自身を dispatch() メソッドを用いて移動する．
2. 調査を行う（調査部分は Patrol と同等である）．
3. 異常があればマネージャへと報告し，なければ正常情報を報告し，消滅する
4. 2 に戻る

また，以下にこれらプロトタイプシステムの主要なクラスのクラス図を図 5.3 示した．

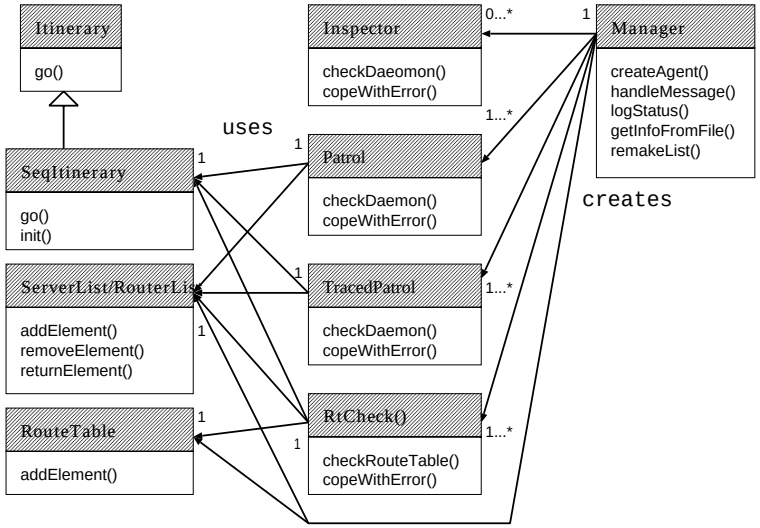

図 5.3: プロトタイプシステムのクラス図（概要）

5.4 移動部分の実装

移動するための基礎情報は全て ServerList，RouterList に格納される．ServerList は Vector クラスで巡回先とそれに対応したサービスデーモン及びデーモンバージョンを格納する．

またRouterListはVectorクラスで巡回先とそこから監視するサーバのリストを格納し，その各サーバへの経路情報も同時に格納する．また，これらのリストは全てSeqItineraryクラスによって管理される．全ての移動するエージェントはこのSeqItineraryに従って移動を行う．SeqItineraryクラスの移動に関する基本部分を図5.4示した．

```
public boolean go(){

  try{
    where2go    = (URL)destinations.firstElement();
    here4backup = here;

    //in case that there is the next destination
    if(where2go != null){
      try{
        here = (URL)destinations.firstElement();
        destinations.removeElementAt(0);

        //dispatch() in Itinerary.go() is called
        go(where2go);
      }

      //
      catch(Exception e){
        System.out.println("!!! [SeqItinerary] system error: " + e);
        here = here4backup;

        return false;
      }

    }
    return true;
  ..........
```

図 5.4: SeqItinerry クラスの移動部分

また，例外処理として移動できないサーバがあった場合はその障害報告を即時にメッセージ通信を行ってマネージャへ行い，そのサーバを省いて次のサーバへ移動する．

5.5 メッセージ通信

Agletsがエージェント間での通信手段として提供するMessageクラスを用いた．各メッセージはKindと呼ばれる識別子で区別され，各値はArgとValueの対で保持される（図5.5参照）．またこれらのメッセージはMessageHandlerクラスをエージェントに実装することで扱うことができる．今回の実装では主に障害発見時にメッセージ通信を行うために迅速な対応ができる非同期で一方向メッセージを用いる事とした．これらのメッセージをマネージャがエージェントから受取り処理を行う．表5.4に全メッセージ種類を記述した．

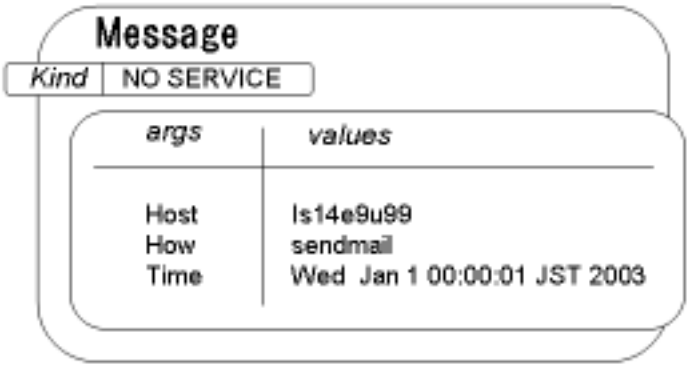

図 5.5: Message クラスの一例（サービス不在障害報告）

表 5.4: メッセージ通信の種類

定期巡回エージェント（通常・追跡モード）	
識別子	内容
USUAL	定期的な通常報告（正常情報を報告）.
NO SERVER	指定サーバへ移動できなかった場合の障害報告
NO SERVICE	指定サービスが存在しなかった場合の障害報告
NO RESPONSE	指定サービスが正常応答しなかった場合の障害報告
HEAVY WORK	CPU 高負荷時の障害報告
END TRACE	追跡モード終了．通常モードへの移行申請（追跡モード時のみ）
集中監視エージェント	
識別子	内容
NO SERVER	指定サーバへ移動できなかった場合の障害報告
NO SERVICE,again	前回に続き指定サービスが存在しなかった場合の障害報告
NO RESPONSE,again	前回に続き指定サービスが正常応答しなかった場合の障害報告
HEAVY WORK,again	前回に続き CPU 高負荷時の障害報告
ALL GREEN	全ての障害から復旧した場合の復旧報告
経路監視エージェント	
識別子	内容
USUAL RT	定期的な通常報告（正常情報を報告）.
NO PING	ICMP ECHO リクエストの応答が無かった場合の障害報告
WRONG ROUTE	指定した経路と実際の経路が違った場合の障害報告
TOO LATE	ICMP ECHO リクエストの応答にかかる時間が遅い場合の障害報告

第6章　評価実験

本章では前章で述べた評価システムを用いて，最初に基本性能測定として監視対象を増加させた場合の総通信量やマネージャ負担，時間を調べ，それらの結果より様々な観点より値を算出し，評価を行う．次に，遠く離れたサーバの監視時に生じる時間遅延の測定やモードの違いによる情報量，時間変化の測定を行う．また一定数の監視対象に対してエージェント数を増加させた場合の通信量や時間の変化を調べる．最後に比較実験として従来の SNMP 方式での通信量測定，及び評価を行う．

6.1　実験環境

特に記述が無い場合，以下の実験ではここで述べる環境下で実験を行った．監視システム及び監視対象として Sun Microsystems のワークステーション（表 6.1 参照）を使用した．また，エージェントによる監視作業は sendmail と http デーモンの調査及び CPU 平均負荷率の調査とした．マネージャは Ultra5*上で動作させ，監視対象として Ultra5，Enterprise250，Blade100 を選んだ．

Blade100，Enterprise250，Ultra5 の順に巡回を行った．実験にて 50 台監視を行うが，監視対象数が 4 以上の場合はこれらの三つの監視対象をそれぞれ複数回巡回することで代用した．これは 50 台管理するよりも 3 台管理する方が実験環境の保持がしやすいからである．負荷が少ない時を見計らってプロトタイプシステムを動作させ，実験した．

また予備実験としてこれらのワークステーション上の JavaVM で動作するプロトタイプシステムのエージェント動作時間の差はシステム全体の動作時間と比較して非常に小さなものであったことを確認し，一様な環境として実験に用いた．

計測方法について述べる．通信量の計測はマネージャと各監視対象上で OS 付属の snoop を用いて行った．マネージャで計測した値をマネージャが負担する通信量，その値と各監視対象上で得られた値の合計を総通信量とした．総時間の計測は上述したマネージャで snoop を用いて得られた，エージェントの送信から受信までの時間を用いた．また，エージェントが監視対象上で監視作業を行った時間は Aglets のコールバックメソッドを用いて onArrival() で到着してから onDispatching() で送出されるまでの時間をエージェント側で出力させたものを用いた．

表 6.1: 各ワークステーション性能

Workstation	CPU	Memory
ULTRA5	UltraSPARC-IIi 333MHz	128MB
ULTRA5*	UltraSPARC-IIi 400MHz	256MB
Enterprise250	UltraSPARC-II 400MHz	512MB
BLADE100	UltraSPARC-IIe 502 MHz	256MB

6.2 基本性能測定

基本性能として通常監視時にかかる通信量・時間についてを調べる．また，それらの結果から得られる各通信路や各ホストでの平均通信量や時間を算出する．

6.2.1 実験内容

はじめに，基本性能として通常モードで定期巡回エージェントが同一セグメント内に配置されたサーバを 1 台から 50 台まで変化させた場合の監視に必要な通信量，及び時間の測定を行った．次にそれらの結果より，得られる情報を算出し，その結果をグラフで示した．

6.2.2 実験結果

結果を図 6.1 に示した．総通信量は監視対象数にほぼ比例しており，1 台監視時に比べて 50 台監視時ではおよそ 40.36 倍の通信量がかかっている．監視対象数に影響を大きく受けることが判明した．グラフはほぼ線形増加であるものの，非常に緩やかに二乗曲線を描いている．

一方で，マネージャにかかる通信の負担を調べた結果，1 台監視時に比べて 50 台監視時では約 1.58 倍になっており，監視対象数に大きな影響を受けないことが判明した．

次に，監視時における時間測定の結果を図 6.2 に示した．この結果より，総時間・作業時間は監視対象数に対して線形増加することが確認できた．また，それら二つの時間差であるネットワークの移動にかかる時間も監視対象数に対して線形増加している．このネットワークの移動にかかる時間はおよそ監視作業時間の半分であることも確認した．

これらの結果をもとに，各通信路ごと，各監視対象ごとの平均通信量及び時間，マネージャ負担の全通信量に対する割合を算出した．

通信路ごとによる各平均値の算出

まず，各通信路で移動にかかった平均通信量を求めた（図 6.3 参照）．結果は監視対象数

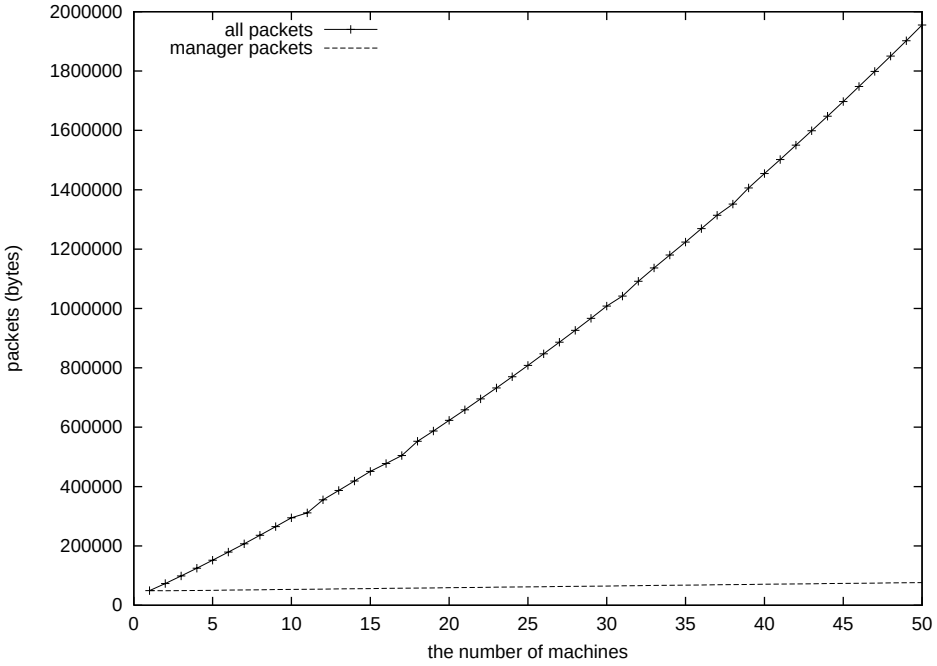

図 6.1: 基本性能 （総通信量／マネージャ通信量）

に対して線形増加している．また，監視対象が一つ追加された場合のサーバリストの増加は約 190bytes である．そして実装したプロトタイプシステムでは Vector クラスによってこのリストを保持しており，必要なくなった場合は消していく．そのため，監視対象に移動する度にリストは約 190bytes 短くなっていく．この結果とグラフの傾き（289.7）より各監視対象で格納している情報量の算出を行う．これより，求める格納情報量は約 480bytes であることが分かった．

各通信路で移動にかかった平均移動時間を求めた．その結果を図 6.4 に示す．結果のグラフは多少ばらつきが見られるが上述した各通信路での移動にかかった平均通信量と酷似した形となり監視対象数に対して線形増加した．平均通信量が平均時間に影響を与えるためである．また，線形近似式を求め，それを図中に記述した．監視対象が 1 台増加すると約 0.0012 秒遅れることが分かった．

監視対象ごとによる各平均値の算出

まず，各監視対象に対して監視にかかる平均通信量を調べた．その結果を図 6.5 に示した．この結果より 6 台〜17 台監視時が最も 1 台にかかる通信量が少なく，その区間では通信量はほぼ 3Kbytes を下回っている．これは総通信量がエージェントの通信路の数とエージェントが格納する情報と最初から保持している巡回リストに依存するためであると考えられる．

続いて，監視にかかった総時間に対して一つの監視対象で平均どれだけの時間が必要で

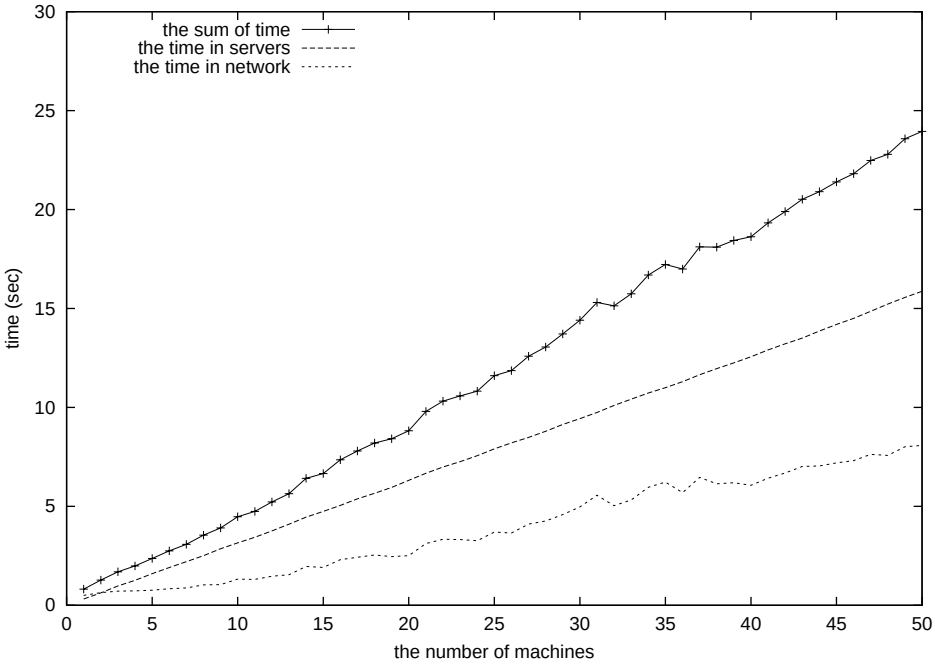

図 6.2: 基本性能 （総時間／エージェント作業時間）

あるかを求めた．その結果を図 6.6 に示した．監視対象数が 5 までは急激にその時間が減り，その減少は徐々に緩やかになる．そして，その後増加に転じていることが分かる．監視対象数 7〜13 が最も時間効率が良い．

最後に総時間に含まれる監視対象上で監視作業にかかった時間について，各監視対象での平均時間を算出した．結果を図 6.7 に示した．各監視対象の性能差があり，もっと離散した値が出ることも予想したがエージェントプログラムが比較的小さいことと高負荷時の測定ではなかったため，予想に反してほぼ一様の結果を得られた．また，この監視作業時間の平均値は 0.316 秒であった．

マネージャが負担する通信量の割合の算出

監視対象が 1 台の場合，全ての通信がマネージャへの経路であるため，マネージャ負担は 100 %となっている．しかしその後，監視対象が増えるにつれて総通信量におけるマネージャ負担の割合は減少し収束する傾向にある．監視対象 50 台時では 3.911 %に減少した．近似式は $y = 131.420x^{-0.886}$ であった．

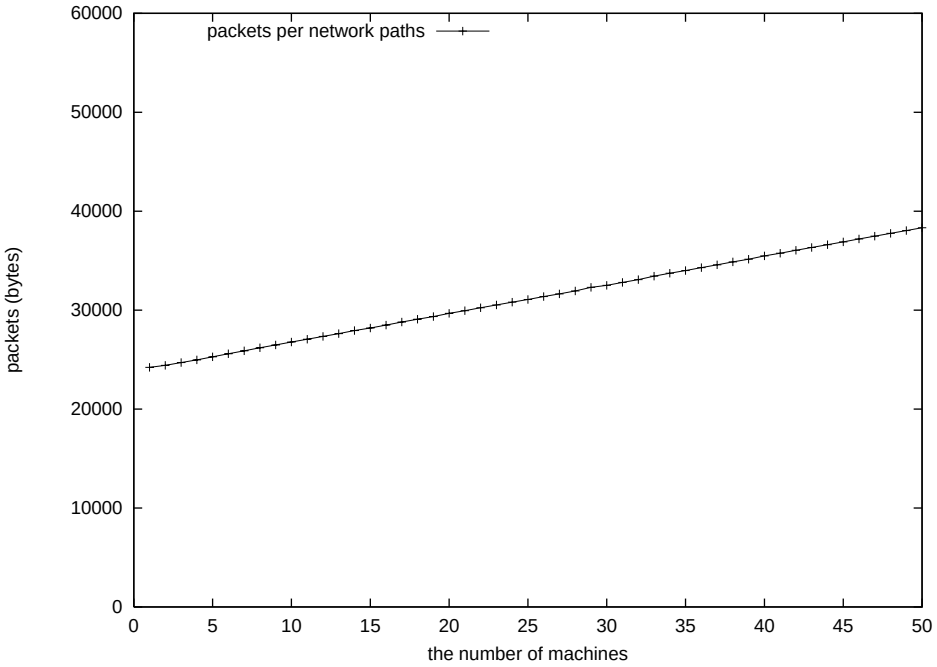

図 6.3: 各通信路での平均移動量

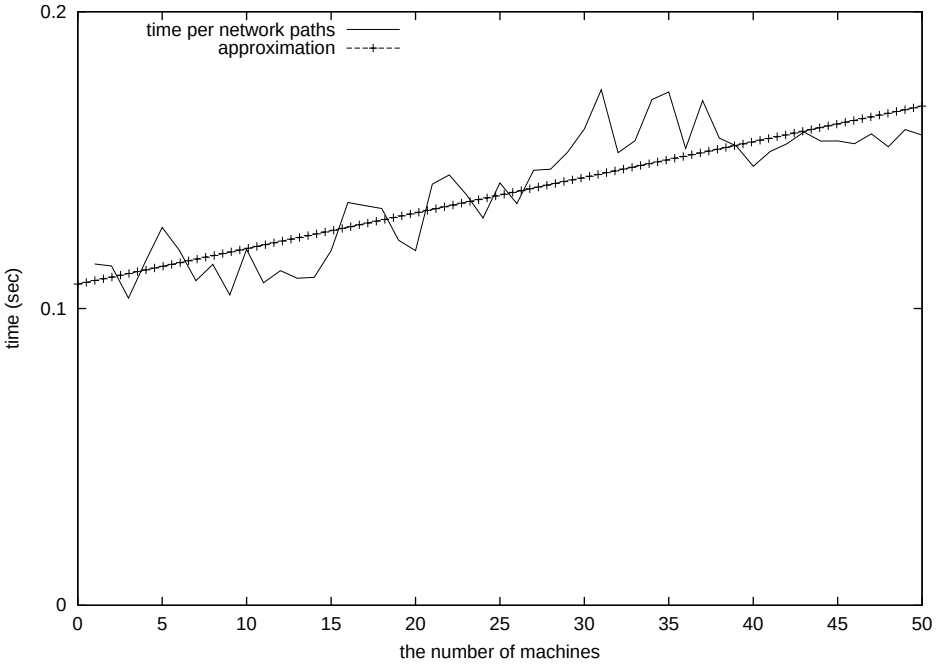

図 6.4: 各通信路での平均移動時間

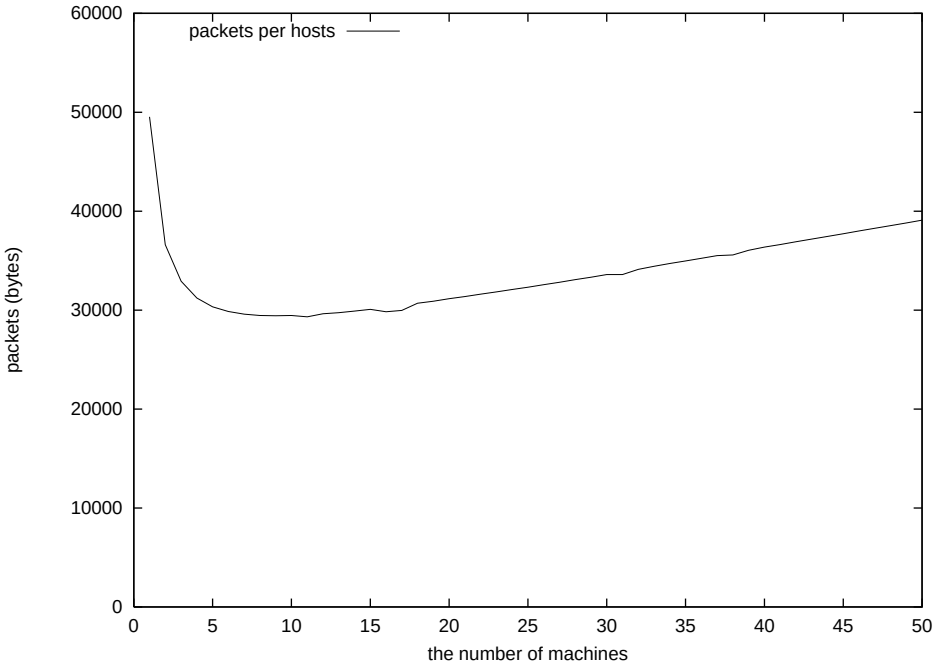

図 6.5: 各監視対象にかかる平均通信量

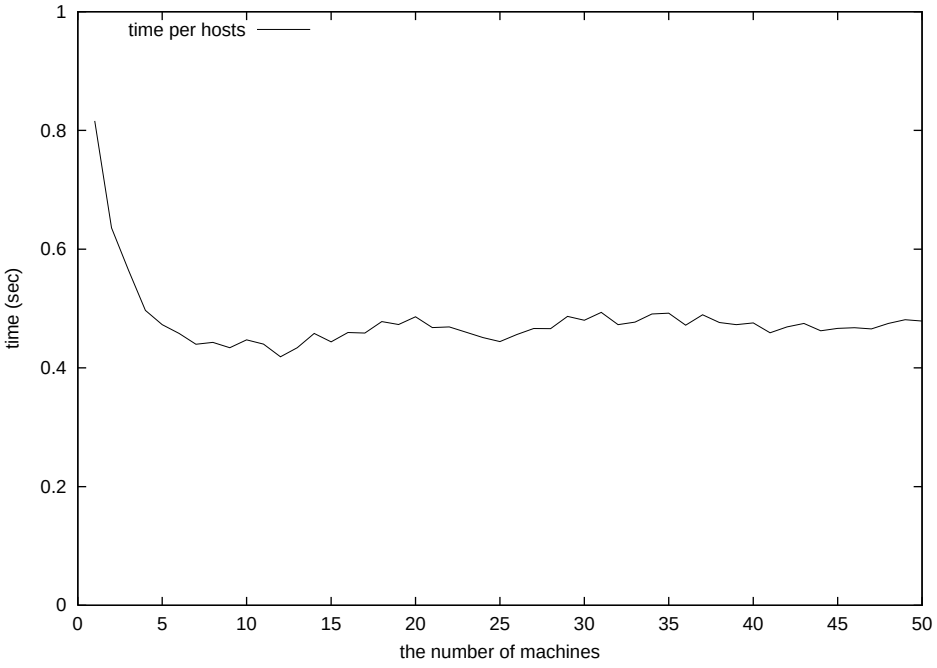

図 6.6: 各監視対象にかかる平均総時間

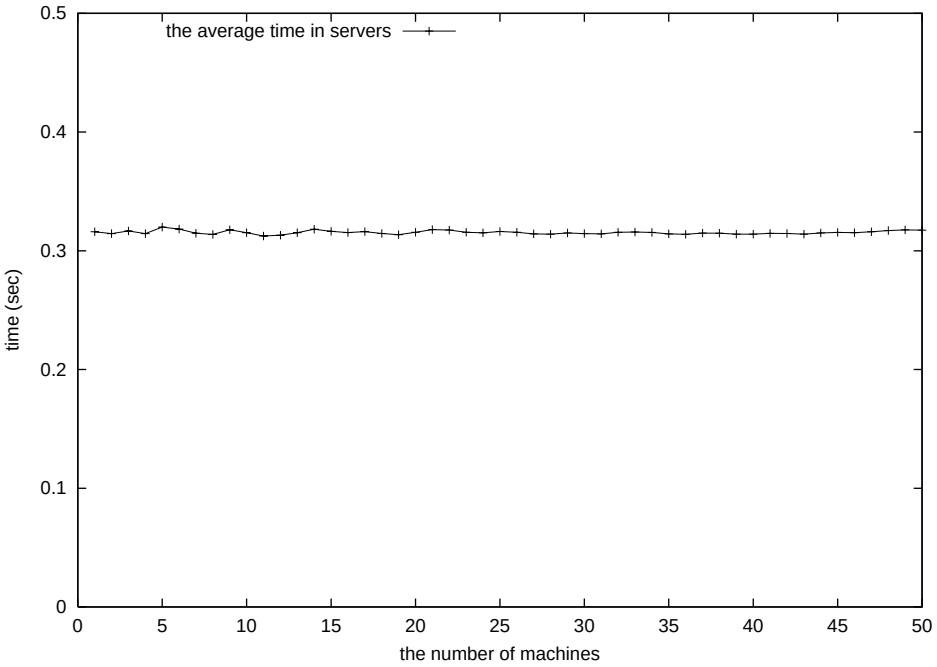

図 6.7: 各監視対象にかかる平均監視時間

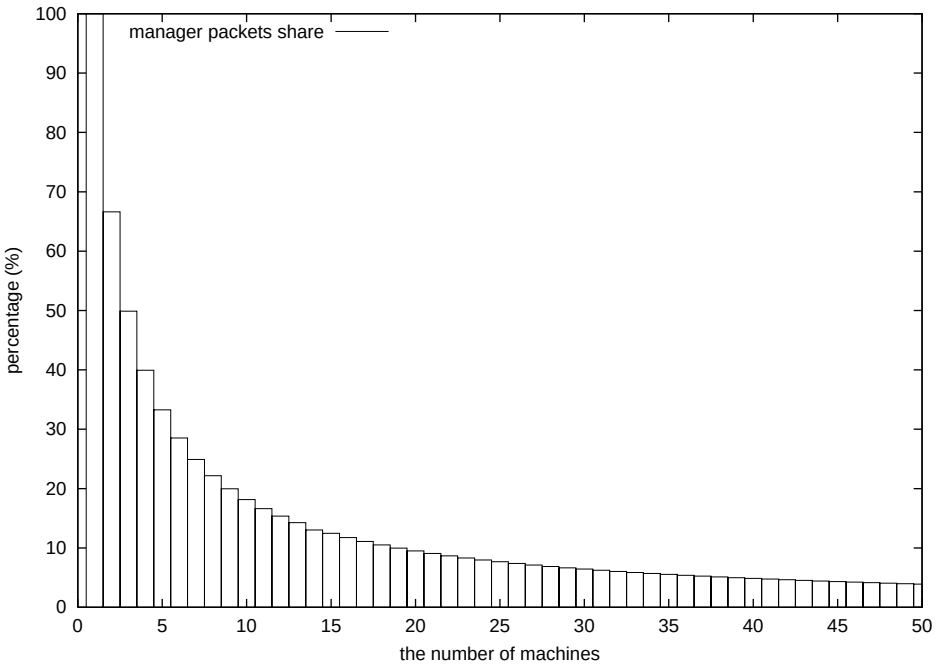

図 6.8: 全体通信量に対してマネージャ負担が占める割合

6.3 監視対象の距離差が与える時間遅延測定

監視対象の距離の違いが与える時間遅延への影響を調べる．

6.3.1 実験内容

距離差による時間遅延の測定には，近くの監視対象の監視にかかった時間と遠くの監視対象を含めた場合の監視にかかった時間との比較をそれぞれの監視対象を 1 台〜50 台にして行った．近くの監視の測定実験では全ての監視対象はマネージャと同一セグメント上にある．また，遠くの監視の測定実験ではマネージャと監視対象の一つである Enterprise250 だけが同一セグメント上にあり，残る Ultra5 と Blade100 の 2 台は他のサブネットワークに存在する．これらの監視対象を Blade100，Enterprise250，Ultra5 の順に巡回させた．Ultra5 の次の巡回先は Blade100 とし，監視対象数繰り返しループさせた．よって，エージェントの監視対象間の移動は最小で 5hops であり最大で 7hops である．この実験環境は図 6.9 の通りであり，非常に高性能なスイッチによって構成された高帯域な高速ネットワークである．

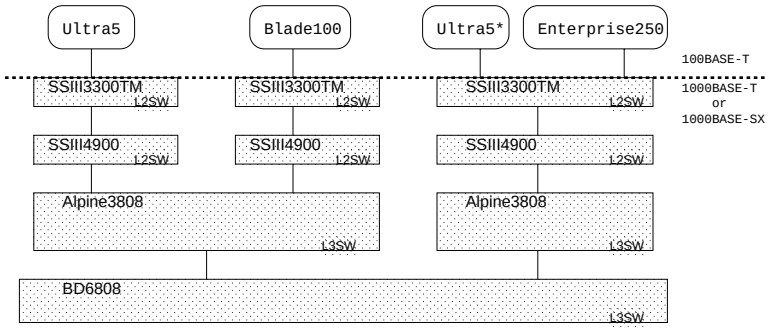

図 6.9: 実験環境ネットワーク概略図

6.3.2 実験結果

結果を図 6.10 に示した．50 台監視時において 1.197 秒の遅延を確認した．一台監視対象が増えると 0.024 の遅延が発生することが分かった．高性能なネットワーク機器で構成されたネットワークにおいては距離の差による時間遅延はあまり無いことが確認できた．

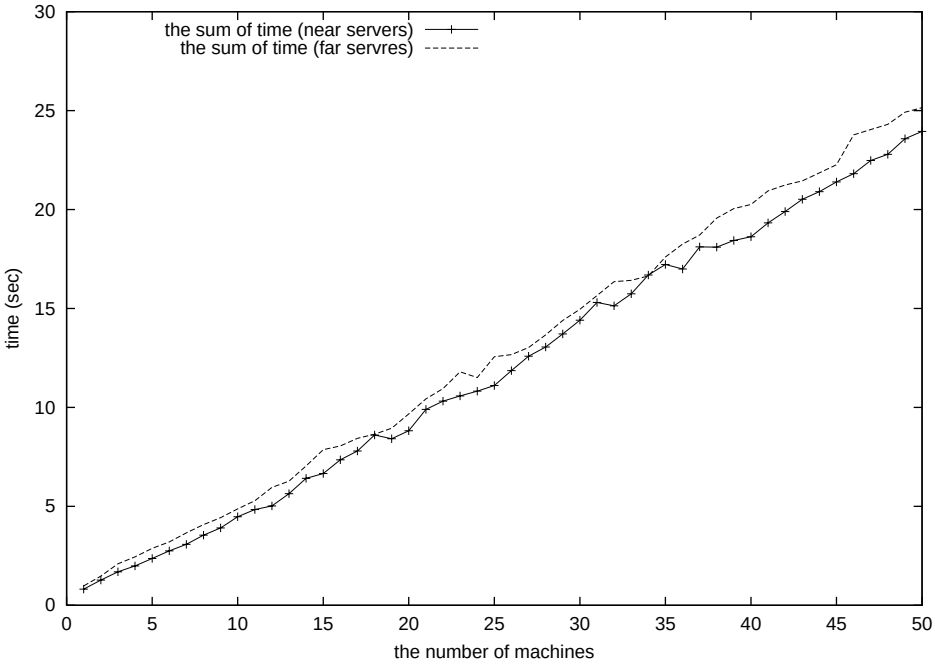

図 6.10: 監視対象までの距離差による時間遅延

6.4 モードの違いによる通信量及び時間比較

モードの違いによる通信量と時間の変化を調べる.

6.4.1 実験内容

通常モードと追跡モードでそれぞれ，監視対象を 1 台〜50 台にして監視業務を行い，その時の総通信量，マネージャ通信量，総時間について比較を行った.

6.4.2 実験結果

図 6.11 にモードの違いによる総通信量とマネージャ通信量の比較結果を示した．また各モードでの総通信量の差とマネージャ負担の差を算出し，その線形近似式を求めた．その結果は表 6.2 の通りである.

求めた式より位置登録にかかる 1 メッセージの通信量はおよそ 4Kbytes であると判明した．また，追跡モードは通常モードのエージェントよりも実装におよそ 1.7Kbytes 多くかかっている．よって追跡モード時では通常モード時と比較して総通信量は監視対象が一台増加するごとにおよそ 5.7Kbytes 増加することになる.

表 6.2: 総通信量差，マネージャ通信量差の近似式

総通信量差	$y = 5963.9x + 2125.5$
マネージャ通信量差	$y = 4268x + 3683.1$

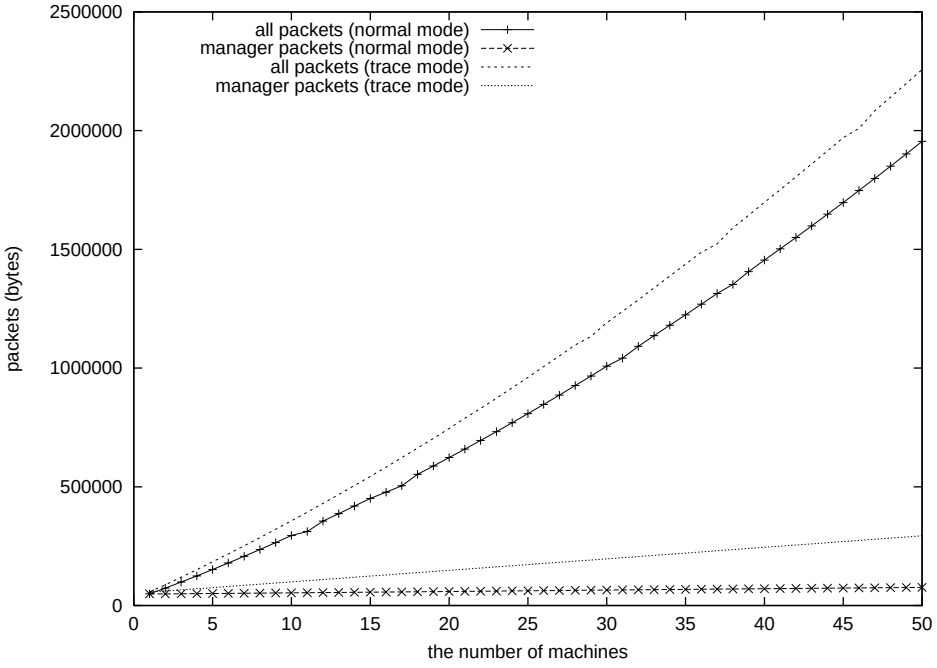

図 6.11: モードの違いによる通信量差 （総通信量）

モードの違いによる時間遅延測定の実験結果を図 6.12 に示した．非同期一方向メッセージを用いて，マネージャとの通信を行っているためにモードの違いによる時間差はほぼ無い事が確認できた．

6.5 エージェント数が与える影響調査

これまでは一つのマネージャに対して一つのエージェントを管理させていたが，今回は複数のエージェントを一つのマネージャでそれらの管理を行い，その時の通信量や時間の変化を測定した．

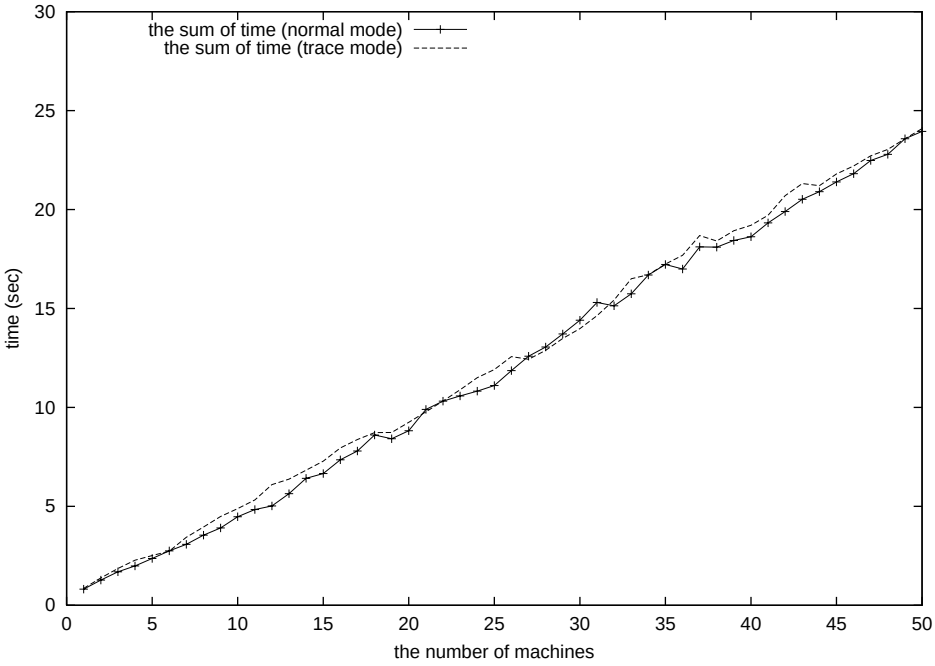

図 6.12: モードの違いによる時間遅延

6.5.1 実験内容

通信量変化の測定には 64 台監視時に通常モードの定期巡回エージェントの数を 1，2，4，8，16，32，64 と変化させた場合にかかった総通信量及びマネージャが負担する通信量について測定した．

次に時間変化の測定には 32 台監視時に実験用の移動だけを行う通常モードの定期巡回にエージェントの数を 1，2，4，8，16，32 と変えた場合にかかった総時間について測定した．この際に動作する実験用エージェントは新たに情報を格納することはなく常に一定量である．PC クラスタシステムを用いて全ての作業行った（表 6.3 参照）．

表 6.3: PC クラスタシステム構成

Best Systems HPC2000	
CPU	Pentium3 (1GHz) X 32
Memory	512MB X 32
Disk	40GB X 32
Interconnection	Myrinet-2000
OS	Linux, SCore

6.5.2 実験結果

通信量変化の結果を図 6.13 に示した．エージェント数に比例してマネージャへの通信が集中してしまう．結果として，エージェント数 8 の場合が最も総通信量は低減できた．総通信量が格納する情報とサーバリストのサイズが影響を与えるためである．

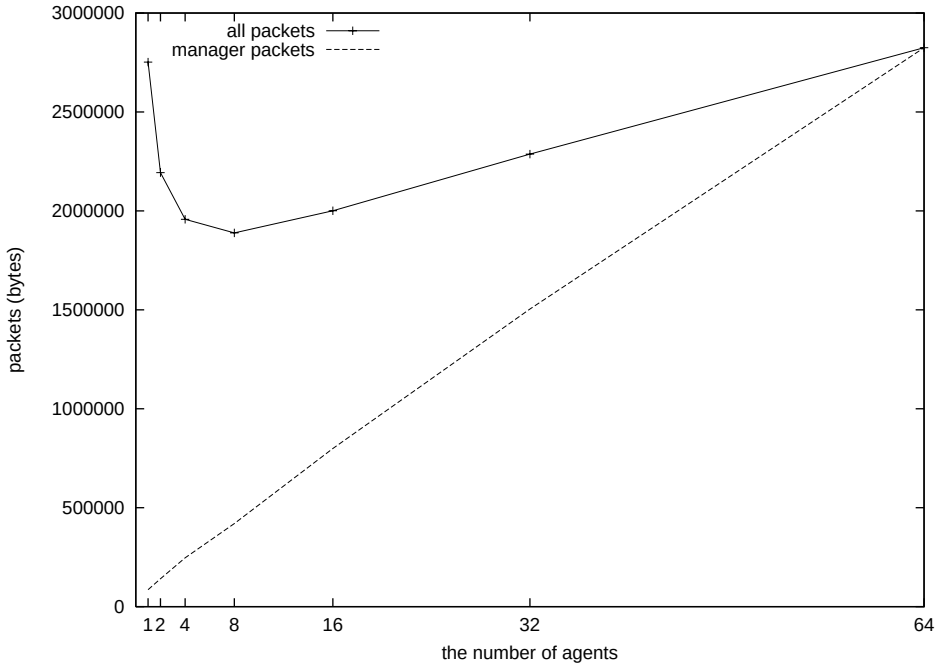

図 6.13: 64 台監視時のエージェント数による通信量変化

時間変化の結果を図 6.14 に示した．エージェント数が 4 まではほぼ反比例して総時間は減っている．しかしその後，エージェント移動経路の総和の増加とマネージャ負担による影響でその傾向は無くなり，エージェント数 16 ではほぼエージェント数 1 の時とほぼ同時間監視にかかり，エージェント数 32 の時にはほぼエージェント数 1 の時の総時間の 2 倍が必要となっている．

6.6 従来方式との通信量比較

既存の従来方式による実際のサーバ監視にかかる通信量と提案方式との比較を行う．

6.6.1 実験内容

ポーリングベースの SNMP と TCP を用いた監視システムである TWSNMP MAP version 1.0.2 を用いた．これは数個の MIB と TCP ベースの通信によってサーバの監視を行うこと

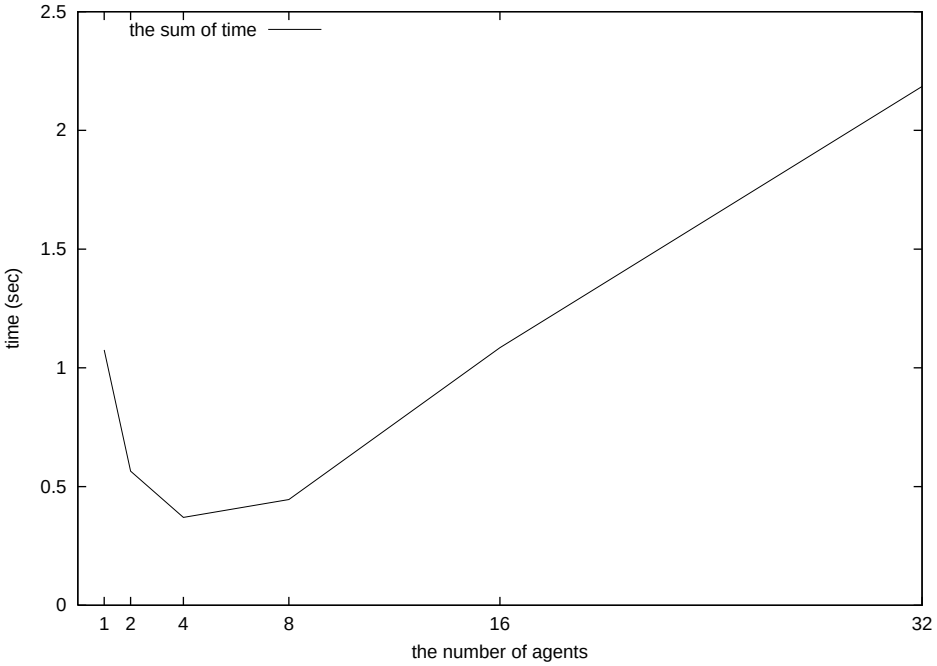

図 6.14: 32 台監視時のエージェント数による総時間変化

が可能な比較的小規模で簡単な監視を行うための監視マネージャである．

プロトタイプシステムと同じく，http サービスの監視，sendmail サービスの監視，CPU の負荷率の調査を監視項目とした．ただし，CPU 負荷率を調査する機能が無かったため，これらの値は MIB オブジェクトを三つ取得して得られたと仮定して最後にこの通信量を付加した．この時は実際に net-snmp の CPU 負荷率が分かる独自 MIB オブジェクトを三つ取得した時にかかった通信量（471bytes）を用いている．

その次に，ポーリングベースの SNMP を用いた監視ソフトであるマネージャ通信について Sun Management Center 3.0（以下，SunMC3.0）のデフォルト設定で必要な通信量を調べ，実運用されている製品化されたシステムにおいてどれだけの通信が行われているかを確認する．そして，その結果より考察を行った．

6.6.2 実験結果

TWSNMP MAP による監視を行った結果，通信量は 1550bytes となった．それらの内訳を表 6.4 にまとめた．

また，通信量と監視対象数は比例関係にある．これより，50 台まで監視した場合までの値を算出し，提案方式とのマネージャ通信量の比較を図 6.15 に示した．

今回の監視のような調査項目が少なく小さな規模での監視においては従来方式の方が有効であることが確認できた．これはモバイルエージェントの情報の集約の機能があまり

表 6.4: 通信量の内訳

監視項目	プロトコル	通信量
sendmail	TCP	511bytes
http	TCP	412bytes
ping	ICMP	156bytes
CPU 負荷率 [1]	SNMP	471bytes

発揮できなかったためである．しかしながら，今回の少ない監視項目においても 1 台監視対象が増えることによるマネージャ通信量の増加割合は提案方式の方が少ないことは確認でき，50 台以上の監視では提案方式の方がマネージャ通信量の低減ができた．

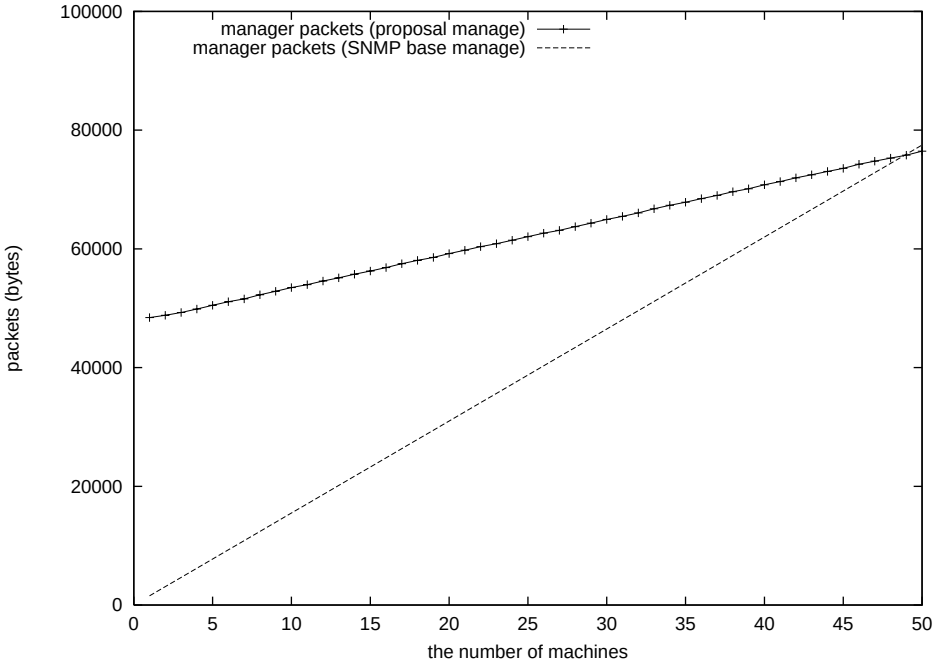

図 6.15: SNMP と TCP ベースによる監視方式と提案方式とのマネージャ通信量比較

デフォルトでは 120 秒ごとにおよそ 7Kbytes の通信を行って監視を行っていることを確認した．また，SunMC3.0 では基本情報として標準 MIB-II から約 10 個のオブジェクト，独自の作り込み部分とベンダー MIB により約 40 個のオブジェクトを取得している．このように商用の監視システムにおいては比較的多くの MIB オブジェクトが利用されていることが確認できた．この時かかったマネージャ通信量を監視対象を 1 台〜50 台までの時を算出して図 6.16 に示した．また，比較対象に MIB10 個，20 個取得にかかるマネージャ

[1]計測では FreeBSD で net-snmp を用いて計測を行ったが Solaris の net-snmp でも同等である

通信量を加えた．この際，一つの MIB オブジェクト取得にかかる通信量は実際に JAIST 内のワークステーション (Ultra5) に対して MIB-II の system グループの全オブジェクトを取得した時にかかった通信量の平均を採った値（176bytes）を利用している．監視内容が違うために，今回のプロトタイプシステムと SunMC3.0 との直接比較を行うことはできないが，データ移動が不要な提案方式は監視対象の増加に対するマネージャ通信の増加の割合は仮に，監視内容が増加してもあまり影響を受けない部分である．

今回のプロトタイプシステムでは実装までは至らなかったために詳細な比較を行うことができなかったが，大規模なサーバ向けの監視においてはよりネットワーク機器と同等以上の情報が必要であると考える．例えば，大規模ファイルサーバのディスクパーティション情報やクラスタ構成などで冗長化されたサーバのネットワークインターフェース情報，またはサーバ監視において重要なログファイル情報などが挙げられる．このように非常に多くの情報を必要とする場合，従来の MIB に頼った純粋な SNMP 監視方式では通信量の増加が著しいため提案方式がより有効であると考えられる．

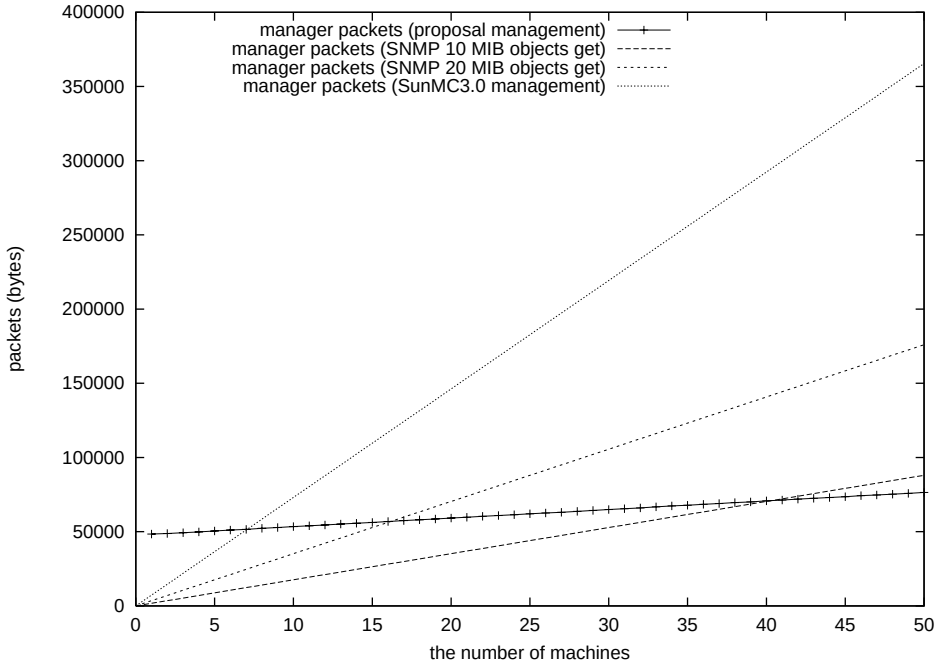

図 6.16: SunMC3.0 と提案方式とのマネージャ通信量比較

第7章　議論

本章では前章で得られた実験結果より考察を行い，通信量，通信の分散化，監視時間，管理性についての議論を進める．

7.1　通信量について

提案方式を用いた場合に必要な総通信量について考えてみる．総通信量は以下の要素に依存している．

- エージェントの数
- エージェント自身の移動に必要な通信量
- エージェントが新たに格納する情報量
- 監視対象数

また，総通信量を求めるために必要となる総通信路は総監視対象数とエージェント数の和であり，監視にかかる総通信量はその総通信路とベースエージェントの移動にかかる通信量の積と格納した情報量が移動にかかった通信量の和で求めることができる．一般的にαをエージェントの1移動にかかるベースの通信量，βを格納する平均情報量とすると，n台の監視時にかかる総通信量は下式の通りである．

$$\begin{aligned} P_n &= \alpha \times (n+1) + \beta \times \sum_{i=1}^{n} i \\ &= \alpha \times (n+1) + \beta \times \frac{n(n+1)}{2} \end{aligned} \tag{7.1}$$

よって，この時1台にかかる通信量は下式の通りである．

$$\begin{aligned} P_n/n &= \frac{\alpha \times (n+1) + \beta \times \frac{n(n+1)}{2}}{n} \\ &= \alpha \times \frac{(n+1)}{n} + \beta \times \frac{(n+1)}{2} \end{aligned} \tag{7.2}$$

これらの式より監視にかかる総通信量は格納する情報量に二乗増加することが分かる．よって，格納する情報を選別し，少なくすることが最も有効に総通信量を低減することが可能であると推測できる．また，プロトタイプシステムにおいてはサイズ可変のサーバリストを保持し，移動する度に不要な要素を削除するためリストが短くなる．これを考慮して，αをエージェントの1移動にかかるベースの通信量，βを格納する平均情報量，γをリストの1監視対象に対する要素の平均サイズとすると，n台の監視時にかかるプロトタイプにおける総通信量を求める式は次の通りである．

$$P_n \quad = \quad \alpha \times (n+1) + (\beta + \gamma) \times \frac{n(n+1)}{2} \tag{7.3}$$

今まではエージェント数が1の場合について述べてきたが，ここでエージェントが複数存在する場合について考察する．一般的にm個のエージェントが監視を行う場合，αをエージェントの1移動にかかるベースの通信量，βを格納する平均情報量とすると，n台の監視時にかかる総通信量は下式の通りである．この時，効率を考えエージェントが巡回するサーバ数はそれぞれ等しいと仮定する．

$$P_{mn} \quad = \quad \alpha \times (n+m) + \beta \times \frac{n(\frac{n}{m}+1)}{2} \tag{7.4}$$

この結果より，エージェント数を増やすという試みは上式の左項を増やし，右項を減らす意味を持つ．上式より総通信量を考慮した最も最適なエージェント数を導き出す事ができる．

他の従来方式と監視対象数nの場合について総通信量に関して比較を行う．SNMP方式では取得するMIBオブジェクト数と監視対象数に比例した通信量となる．また，一つのMIBオブジェクトの取得にかかる通信量は非常に小さい．MIBオブジェクトを一つ取得するのにかかる通信量を176bytesとし，取得するオブジェクト数をxとすると，総通信量は$176 \times x \times n$となる．MIB数に依存するが，取得するMIB数が少ない場合非常に通信量は小さい．また，常駐エージェント方式は通常時ほとんど通信が行われないため総通信量は非常に小さい．提案方式のエージェントを全て監視対象上で常駐させた場合を仮定すると，通常時はマネージャからエージェントへの状態把握のためのポーリングにメッセージ通信でおよそ4Kbytesの通信が必要なだけである．この値は監視対象数に比例するので監視対象数をnとすると，$4000 \times n$で表すことができる．一方でプロトタイプのシステムの実際の値を入れた総通信量の概算は$24000 \times (n+1) + (480+190) \times \frac{n(n+1)}{2}$となり，結果は$335n^2 + 24335n + 24000$となる．これより提案方式の総通信量の方が大きいことが分かる．今回の実装ではあまり細部で通信量を減らす試みを行わなかったために格納する情報量やリストの大きさが比較的大きいことが影響している．

しかしながら，通信が一カ所に集中する従来方式と違い，提案方式においては総通信量は監視対象となるネットワークシステム全体を流れるパケットの総流量であり，その意味あいが変わり，あまり重要ではないと認識している．

7.2 通信の分散化

まず，監視に必要な通信路の数について考える．従来方式では，通信路の数は総監視対象数 $\times 2$ で表すことができる．一方，提案方式では総監視対象とエージェント数の和で表すことができる．総通信路が少ないということはネットワークを利用する回数が少ないということである．同じ数の監視対象を監視した時において，提案方式では従来方式と比較してネットワーク利用の回数が監視対象数の影響に大きな影響を受けない．その恩恵としてネットワークのパケット送受信にかかる通信機構のオーバーヘッドを軽減することが挙げられる．

次に監視システムが負担する通信量について考える．一般的には α をエージェントの 1 移動にかかるベースの通信量，β を格納する平均情報量とすると，n 台の監視時にかかるマネージャが負担する通信量は下式の通りである．

$$PM_n \quad = \quad 2 \times \alpha + \beta \times n \tag{7.5}$$

先に述べた総通信量と比べてマネージャが負担する通信量は格納情報量分だけが監視対象数に比例して増加するだけある．

また，巡回リストが増減するよう実装したプロトタイプシステムにおいて，α をエージェントの 1 移動にかかるベースの通信量，β を格納する平均情報量，γ をリストの 1 監視対象に対する要素の平均サイズとすると，n 台の監視時にかかるマネージャが負担する通信量は下式の通りである．

$$PM_n \quad = \quad 2 \times \alpha + (\beta + \gamma) \times n \tag{7.6}$$

方式別に監視対象数が n の時にかかるマネージャ通信量を概算して比較を行う．SNMP 方式では議論した総通信量がそのままマネージャ負担になるため，マネージャ通信量は $176 \times x \times n$ となり，結局は MIB オブジェクト数に依存する．また，提案方式を常駐エージェント型方式として運用した場合のマネージャ通信量も上述した総通信量と等しく，$4000 \times n$ となる．提案方式のプロトタイプシステムの場合，マネージャ通信量は $2 \times 24000 + (480 + 190) \times n$ となり，結果は $670n + 48000$ である．監視対象 15 台以上の場合は常駐型よりも移動型エージェント方式がマネージャ通信を抑えることができた．この結果より大規模なネットワークシステムでの監視業務であれば従来方式と比べて通信の分散により監視システムが負担する通信量を軽減することが分かった．

モバイルエージェントを用いるためにネットワーク中を分散して通信を行う事ができ，従来方式で生じる監視マネージャ周辺での通信の局所化を防ぐことが可能である．特に，監視対象として想定している大規模で分散したネットワークシステムでの運営により大きな恩恵を受ける．従来方式ではネットワークの構成に依らず，一点に通信が集中してしまいそこが通信のボトルネックとなり結果としてネットワーク全体の大きさを決定してい

た．しかし，提案方式では監視対象の配置や，それらの巡回順番によって，自由に通信路を選ぶことが可能である．サーバ群を比較的まとめて配置すればそのセグメントに通信は集中するが，それらのサーバを分散して配置すればその分だけ通信も分散することが可能である．集中監視を目指しながらネットワーク構成によって監視にかかる通信を分散できることが大きな特徴である．またこれによって監視システムがネットワークシステム全体のスケーラビリティの低下を防ぐことも可能である．

ここで少しマネージャの負担といった観点から通信量ではなく，CPU の負荷について考える．通信の負担は確かに提案方式では軽減されているが，一方でプラットフォーム非依存な Java で実装されているため，単純な仕組みである SNMP と比べてより CPU の負荷は一時的に高くなる．しかしながらサーバと想定するアーキテクチャは強力な計算資源であると考えられるためあまり問題にはならないと考えられる．また，現実に Java によって実装された商用のディスクアレイの監視ソフトウェアやサーバ監視ソフトウェアが広く使われている．

7.3 監視時間

正常情報の取得及び通知

正常情報の取得からシステムがそれを検知するまでの時間遅延を調べる．監視対象数を n，その時の 1 移動にかかる平均移動時間を α とすると，その時間遅延の期待値は次の通りである．

$$\begin{aligned} K_n &= \frac{\alpha}{n} \times \sum_{i=1}^{n} i \\ &= \frac{\alpha(n+1)}{2} \end{aligned} \tag{7.7}$$

35 台の監視対象がある場合について考えてみる．先の実験結果（図 6.4）より，35 台監視時において 1 移動にかかる平均時間は 0.15 秒であった．このため，監視対象全てを巡回するのに 5.4 秒かかることになる．この時，1 台目の監視対象の情報が取得から検知されるまでの遅延が最も大きく，5.25 秒である．また，この時の期待値 K_{35} は 2.7 である．ポーリング間隔を 5 分と考えると，ポーリング間隔に対する 35 台監視時の時間遅延の期待値の割合は 0.9 %であった．

正常情報であるために少しの時間遅延は許され，例に挙げた通り，遅延時間に対して一般的にポーリング間隔が非常に長い時間に設定されている場合が多く，この正常情報の遅延による影響はほとんどない．

障害情報の取得及び通知

ネットワークシステムの監視ではポーリング間隔によって情報を取得する．このため，ポーリング間隔により最初の障害検知が行われる．これは従来方式においても，提案方式

においても同様である．また，障害情報を取得後，マネージャへそれを通知するプロセスにおいて提案方式ではその情報を格納して移動していくのではなく，取得後すぐにメッセージ通信による通知を行う．このために従来方式と同等であると考えられる．方式による通信量の僅差による時間遅延が生じるがポーリング間隔に対して十分小さいために，ここでは同等と見なすことができる．例として提案方式のメッセージによる通信量を 4Kbytes，従来の SNMP 方式のメッセージを 176bytes，ポーリング間隔を 5 分，ネットワークを 100Mbps として考える．この時提案方式のメッセージ通信にかかる時間は 4.00×10^{-3} であり，SNMP 方式の通信にかかる時間は 176.00×10^{-6} である．これらの時間差は 3.82×10^{-3} 秒である．この時間のポーリング間隔に対する割合は 0.01 %未満であり，大きな影響を与えるような値でないことが分かる．障害情報の取得，通知にかかる時間はポーリング間隔に依存した値であるため，提案方式も従来方式も同等である．

厳密なタイミングでの監視の動的変更

重大な障害が起き，非常に厳密なタイミングでの監視を必要とするサーバなどの監視ではより，ポーリング間隔を短くすることでその障害変化を通常時よりも迅速に検知し，障害状況を把握することが可能である．通常，重点的に監視を行いたいサーバにはポーリングによるネットワーク上のトラフィックが必要でない常駐エージェントによる監視がより望ましい．提案方式では集中監視エージェントといった常駐エージェントが存在し，それを自在に設定配置できる仕組みがある．マネージャ側で一元管理されたポリシーをもとに必要に応じて動的に常駐エージェントを生成することでポーリング間隔なども動的に変化させることが可能である．

同時刻における並列監視作業

同時刻における複数の監視対象への監視業務が必要な場合が考えられる．現在，別々の監視対象に対して同時性を保った監視を行う機構は実装されていない．しかし，例えばマーケティングやネットワークシステムの構成管理を目的に統計分析などで同時性を保った監視が行われることがある．このような分野においては複数箇所の同時性を保った監視が重要であると考えられる．現在の実装ではメッセージ通信をマネージャとエージェントとの協調作業に用いているが，これをエージェント同士間の協調作業にも用いることで対処できる．複数のエージェントがそれぞれ担当の監視対象に移動し，その後，メッセージによって一斉に動作を開始する絶対時刻を知らせる．これによって同時性を保った複数監視が実現できると考えられる．その他にも常駐エージェントである集中エージェントを配置し，絶対時刻で同期を取りながら作業を行う方法を用いることによって同時性を保った複数監視の実現が可能である．

しかし，障害はイベントであり，起きたかどうかが重要であるため，サーバの障害監視といった意味においては厳密な時刻を要求することにあまり重要ではない．例えば，同時性を保つ監視に関連して連鎖的な障害の監視についても検討する．連鎖的な障害とはある一つの障害が起ると，それに関連する全てのサーバにその障害の影響が伝搬することを言う．このような場合において障害伝搬の速度が非常に遅くない限り障害の発生した順番を

突き止めるのは難しい．ポーリング間隔で監視を行っているため，同時に監視を行っていたとしても同時に分かるだけである．結局，マネージャに集められた障害情報を管理者が後に解析することによって障害の発生の流れが分かる．

7.4 管理性

提案方式は集中管理されるシステムでの運用を想定して設計してきた．複数の監視システムが構成されてしまう既存の SNMP 方式などと比較して実際の管理業務に適したシステムを管理者へ提供し，円滑な監視業務が可能となる．また，マネージャを単一にすることで監視システム同士の統合を無くすことによって監視システムの持つ情報の統合にかかる通信量を低減できる．そして，各監視属性やサブネットごとの運営ポリシーなどを一カ所で管理し，それをエージェントに持たせることで情報の集約を行った．

また，複雑なサーバ監視を行う場合，各サーバに特別なプログラムを設置する従来の監視方式と比べて，提案方式は監視プログラムの総数が少ないという大きな特徴がある．これにより次の利点がある．

- 初期導入コストの低減

  マネージャとして動作させるホストに対して全ての監視プログラムを設置するだけで全ての監視業務が行える．各監視対象に対してモバイルエージェントの実行環境を提供する以外に特別な作業は不要である．

- チューニング，メンテナンスコストの低減

  マネージャ側で監視プログラムのバージョンアップや修正作業を行うだけでシステム全体へ影響を及ぼすことが可能である．

また，モバイルエージェントを用いることによって常駐エージェントだけを用いた方式と比べて，次の点で優れている．

- 監視システムの把握

  モバイルエージェントがマネージャに戻ることによって，マネージャは各エージェントの状況を把握することができる．そして，マネージャが容易にそれらのエージェントを集中管理することが可能である．

- 変化への柔軟な対応

  監視業務においてその時の状況に合わせた対処を簡単に行うことができる．昼夜で監視の内容を変え，通信量の多い時間帯は比較的管理パケットのかからない軽度の調査を行う監視エージェントにしたり，逆に，通信量の少ない時間帯に合わせて重度の調査を行う監視エージェントにすることが可能である．また，必要に応じて重要な監視対象に対して常駐エージェントを設置して回るなどの処理を行うことも可

能である．極端な例を挙げれば，既存方式では管理コストの面から非常に困難な，全く機能の異なったエージェントを状況に応じて動作させることを容易に行うことができる．

第8章　おわりに

本章では今まで述べてきた事のまとめと今後の課題を述べる．

8.1　まとめ

近年のネットワークシステムの大規模化・複雑化に伴い，従来の SNMP を用いた監視方式では通信量や管理コストの面などで様々な問題が生じてきた．また現在では複雑な監視業務を行う場合，SNMP 方式だけでは困難であり，常駐型エージェント方式と組み合わせて監視を行っている．常駐エージェントが結果を MIB オブジェクトとして格納し，それを SNMP を用いて通知する方法が一般的である．このような監視方式はネットワークトラフィックの低減を実現する一方で，常駐エージェントを全ての監視対象に設置・管理を行うことは管理性において大きな問題がある．そこで，本研究では分散オブジェクト技術であるモバイルエージェントに注目し，普段は可能な限り通信量を低減しつつエージェントを巡回させ，状況に応じて動的に動作を変え，必要箇所には常駐エージェントを生成し，柔軟な監視を行う方式を提案した．モバイルエージェントを用いた提案方式によって得られる利点である，情報の集約，通信の分散，管理性，柔軟な対応について既存方式と比較してその有効性を示した．

情報の集約

エージェントがネットワークを移動し監視対象上で様々な値を自律的に取得し，それらの値から結果を判断することで情報の集約を行った．現在，これらの管理情報は増加の一途を辿っており，大規模なシステムにおいては重大な問題である．モバイルエージェントの自律性を活かし，その場で判断させることによって大規模な通信が必要な場合においてネットワークトラフィックを低減させることが可能であり，増加傾向にある管理情報を効率良く取得するための方式であることが示せた．

通信の分散化

従来の監視方式では全ての監視対象が監視マネージャと 1 対 1 の通信路を持っている．このために局所的な通信の集中が起こり，結果として，システム全体のスケーラビリティを決定した．しかし，分散オブジェクト技術であるモバイルエージェントを用いることによってその通信の流れを制御でき，従来問題となっていた監視システム周辺の局所的な通信の集中を低減することを示した．

管理性

管理性の向上にはモバイルエージェントの与える影響は非常に大きい．本研究では管理コストの低減に有効な集中監視について注目してきた．そこでこの形態にあった単一マネージャによる監視システムの設計を行い，現実の管理形態により適した監視システムを実現した．また，全ての監視プログラムはマネージャ上に蓄えられ，変更が生じた場合においてもその一カ所だけを管理することで全体に影響を及ぼすことが可能である．

柔軟な対応

モバイルエージェントが移動することによって一元的にマネージャでエージェントの動作・状況を把握し，監視業務の変更などが柔軟に行えることを示した．また，従来の監視はネットワーク機器単体のものが多かったが，その機構ではサーバの監視を効率よく組織的に行うことはできなかった．クライアント側からのサーバ周辺の経路状態の監視や，ログの判断など複雑な業務においてもエージェントはその威力を発揮できる．そして，常駐エージェントだけを用いるのではなく，モバイルエージェントを用いてその常駐エージェントの配置に行った．初めから常駐エージェントを配置すると，マネージャがそれらの独立して動くエージェントを把握するのが難しく，監視業務内容の変更などを伝えるためにはコストがかかる．そこで，その状況に応じて，普段はモバイルエージェントが比較的軽度な監視を行い，マネージャの意図を反映して動的に最適な監視に変化させる方法を示した．

上述してきた利点によって本研究は特に，集中管理されている大規模分散ネットワークシステムにおいて非常に大きな恩恵をもたらすことが確認できた．今後もさらにこのネットワークシステムは大規模分散化，複雑化しその管理業務にかかるコストは増加する傾向にある．このような状況下において，本研究は有効な管理法の一つであると考えられる．

8.2　今後の課題

実運用するためには，状況に応じた柔軟な管理者への通知のシステムの実装が必要である．監視対象ネットワークに最適なエージェント数の自動導出及び，監視時のネットワーク状況に応じて動的にリストを生成，変更を行うことでより有効に本研究の利点が活かせるようになると考える．また，監視業務だけでなくサービスプロセスの自動アップデートやライセンス管理などへの応用も考えられる．

謝辞

本研究を進めるにあたって終始御指導頂いた指導教官の敷田幹文助教授に心から感謝致します．また，ゼミを通して有益な助言をして頂いた大西健治氏，坂下幸徳氏，そして敷田研究室の後輩の皆様に感謝致します．評価実験を行うにあたって，快く計算機の使用を認めてくれた友人達にも深く感謝致します．

参考文献

[1] 敷田幹文, 井口寧, 三輪信介, 丹康雄, 松澤照男. 大規模高可用性サーバの設計と運用. 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, pp. 57–62, 2001.

[2] 磯川弘実, 萱島信, 寺田真敏, 伊藤武. インターネットサービスの特徴を考慮したサーバ稼働監視システムの実装. 情報処理学会 DSM 研究会, Vol. 9, No. 1, pp. 1–6, 1998.

[3] 犬束敏信, 藤崎智宏, 蔭山克禎. 分散協調管理プラットフォームの実現. 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, pp. 25–30, 2000.

[4] 泉裕, 上原哲太郎, 國枝義敏. 柔軟な管理情報の共有によるトラブルチケットシステムの構築. 情報処理学会 DSM 研究会, Vol. 20, No. 10, pp. 55–60, 2000.

[5] 長田智和, 谷口祐治, 玉城史朗. 大規模分散ネットワーク運用管理システムの提案. 情報処理学会 DSM 研究会, Vol. 20, No. 6, pp. 31–36, 2000.

[6] 斎藤明紀. 教育用大規模計算機システムにおける管理の省力化手法. 情報処理学会論文誌, Vol. 41, No. 12, pp. 3198–3207, 2000.

[7] 久長穣, 北上悟史, 渡邊孝博, 井上裕二. 複数 vlan の動的切り替えネットワークの構築について. 情報処理学会 DSM 研究会, Vol. 22, No. 7, pp. 39–44, 2001.

[8] Andrew S. Tanenbaum. コンピュータネットワーク. ピアソン・エデュケーション, 第 3 版, 1999.

[9] 日立製作所. JP1 Version 6 JP1/Integrated Manager - Console マニュアル, 2001.

[10] 日立製作所. JP1 Version 6 JP1/Base マニュアル, 2001.

[11] 三好優, 釜洞健太郎, 朴容震, 浦野義頼, 富永英義. モバイルエージェントによる大規模・分散型ネットワーク管理法の一提案. 情報処理学会 DPS 研究会, Vol. 97, No. 11, pp. 61–66, 2000.

[12] 知念眞也, 長田智和, 谷口祐治, 玉城史朗. 分散オブジェクト技術を用いたネットワーク監視システムの設計. 情報処理学会 DSM 研究会, Vol. 20, No. 7, pp. 37–42, 2000.

[13] IBM 東京基礎研究所. Aglets project. http://www.trl.ibm.com/aglets/.

[14] Danny B. Lange and Mitsuru Oshima. *Programming and Deploying JAVA Mobile Agents with Aglets*. Addison-wesley, 1998.