分散システムにおける
動的負荷分散に関する研究

山井 成良

1993 年 8 月

内容梗概

本論文は著者が大阪大学大学院基礎工学研究科後期課程 (物理系専攻情報工学分野) 及び奈良工業高等専門学校情報工学教室において行った研究のうち，分散システムにおける動的負荷分散に関する研究をまとめたものである．

本論文では，いくつかの UNIX ワークステーションを Ethernet などのローカルエリアネットワーク (LAN) で相互接続することによって構成される典型的な分散システムを主たる対象とし，全体としての効率や性能の向上を図るための動的負荷分散手法や，その実装方法について述べる．

まず，本研究の対象である分散システムについて概説し，同システムが持つ機能や特徴を明らかにする．更に，主な負荷分散の方式について概説し，本研究で採用したプロセス発送 (process dispatching) 方式の妥当性について述べる．

負荷分散の性能は負荷の尺度に大きく左右される．従来の動的負荷分散手法の殆どが FCFS(First Come First Served) スケジューリング方式を対象にしているのに対し，本研究ではラウンドロビン (RR) スケジューリング方式を対象とし，新しいプロセスの割り当てによる平均応答時間の増加量を基準とした負荷の尺度を提案する．また，シミュレーションによりこの尺度が従来の尺度である残余仕事量の合計やプロセス数より優れていることを示す．

一方，動的負荷分散では通信量を抑えながら最新の負荷情報を収集する方法が問題となる．この問題に対して，本研究ではプロセス割り当てや終了通知などの通信の傍受による負荷情報の収集手法を提案する．また，シミュレーションにより従来の定期的同報通信方式や入札方式と比較し，本手法がこれらの方式より優れていることを示す．

最後に，耐故障性やネットワーク透過性を考慮した動的負荷分散機能の UNIX への実装手法について述べる．本研究では，耐故障性を実現する方法として，同報通信に対するローカルホストの応答時間を基準としたホスト選択手順を提案する．また，ネットワーク透過性を実現する方法として，REX を用いたコマンド遠隔実行機能のコマンドインタープリタ (シェル) への実装を提案する．また，実際に負荷分散機能をシェルに実装し，本手法が負荷分散の性能を低下させることなく耐故障性やネットワーク透過性を有していることを示す．

関連発表論文

1. 学術論文誌掲載論文

[1-1] 山井成良，下條真司，宮原秀夫：“マルチプロセッサ時分割システムにおける負荷分散アルゴリズム”，**電子情報通信学会論文誌 (D-I)**，Vol. J72-D-I, No. 2, pp. 75–82(1989).

[1-2] 山井成良，下條真司，宮原秀夫：“同報通信機能を持つ分散システムにおける負荷分散アルゴリズム”，**電子情報通信学会論文誌 (D-I)**, Vol. J75-D-I, No. 8, pp. 536–544(1992).

[1-3] 山井成良，若林進，下條真司，宮原秀夫：“UNIX におけるコマンド単位の負荷分散機能の設計と実装”，**電子情報通信学会論文誌 (D-I)** (投稿中).

2. 国際会議録掲載論文

[2-1] Nariyoshi Yamai, Shinji Shimojo and Hideo Miyahara : “A Process Dispatching Algorithm on Multiprocessor Time Sharing Systems,” *Proceedings of The 11th Annual International Computer Software and Applications Conference,* IEEE, pp. 681–686(1987).

[2-2] Nariyoshi Yamai, Shinji Shimojo and Hideo Miyahara : “A Process Dispatching Algorithm on Distributed Time Sharing Systems by Monitoring Network,” *Proceedings of 1989 Joint Technical Conference on Circuits/Systems, Computers and Communications,* IEICE, pp. 324–328(1989).

3. 研究会発表論文

[3-1] 若林進，山井成良：“UNIX における動的負荷分散の試み”，**第 18 回 jus UNIX シンポジウム論文集**，日本 UNIX ユーザ会，pp. 160–170(1991).

[3-2] 関努，若林進，山井成良：“UNIX における負荷の推定方法に関する考察”，**第 20 回 jus UNIX シンポジウム論文集**，日本 UNIX ユーザ会，pp. 12–21(1992).

目次

第 1 章

序論

1970 年代までの計算機システムは，高価なメインフレームを中心とした集中システム (centralized system) が主流であった．ところが，1980 年代に入ると，ハードウェア技術の急速な進歩に伴い，従来のメインフレームより遥かに価格性能比が優れたワークステーションが普及するようになった．これに伴い，Ethernet に代表されるローカルエリアネットワーク (Local Area Network, LAN) が普及し，UNIX 等のワークステーション用のオペレーティングシステム (Operating System, OS) にも LAN をサポートする機能が標準的に備えられるようになり，現在では多くの UNIX ワークステーションを LAN で相互接続してプロセッサ，記憶装置，周辺機器などの資源の共有によって全体の性能の向上を図る典型的な分散システム (distributed system) が急速に普及してきている．

一方，分散システムを利用するためのソフトウェアについても 1980 年代初めから大学や研究所を中心に研究が進められている．このうち OS 全体に関しては，ユーザにネットワークの存在を意識させない性質，すなわちネットワーク透過性 (network transparency) を持ち，分散システム全体を 1 つの集中システムと同様に扱うことができる分散オペレーティングシステム (distributed operating system, 分散 OS) が研究開発されている．これまでに開発された分散 OS の例としては，LOCUS[1], V-System[2], Amoeba[3], Mach[4], Sprite[5], ToM[6], Muse[7] などが挙げられる．また，ファイルシステムに限定した部分に関しては，ファイルが存在する計算機を意識させない性質，すなわち位置透過性 (location transparency) を持ち，分散システム内でファイルシステムの共有を行うことができる分散ファイルシステム (distributed file system) が開発されている．これまでに開発されてきた分散ファイルシステムの例と

しては，NFS(Network File System)[8], RFS(Remote File Sharing)[9], AFS(Andrew File System)[10] などが挙げられる．

分散システムを導入するもう一つの目的は性能の向上にある．複数の計算機から構成される分散システムでは，複数の計算機で同時に処理を行った方が1台の計算機で全ての処理を行うより効果的である．従って，分散システムを効率よく使用するためには，各計算機間で負荷の分散を行なうことが重要となる．これにより，一部の計算機だけに負荷が集中する状態を避け，プロセスの平均実行時間の短縮，システムのスループットの向上や資源の利用率の改善など，システム全体の性能の向上を図ることができる．

しかし，前記の分散OSの研究の主な目的はネットワーク透過性の達成であり，コマンド遠隔実行 (remote execution) 機能やプロセス移送 (process migration) 機能など負荷分散を行うための基本機能は備えているが，負荷分散アルゴリズムは組み込まれていないものが殆どである．また，分散OSのうち普及しているものはMachなどごく一部であり，更に分散OSを導入するためにはUNIXのソースライセンスを必要とするなど，現在の時点では現存のシステムで利用することが困難であることが多い．

従来，負荷分散の研究は分散OSの研究と並行して多くの研究者により進められており，システムの構成やモデルに対する種々の仮定を基に多くの手法が提案されてきている (例えば [12]–[24] など)．しかし，これらの手法には，現実的でない仮定を設けているためそのまま適用できないもの，あるいは負荷が集中する危険性があるものなど，分散システムに実装するには問題があるものが多い [25]．

そこで本研究では，UNIXワークステーションをLANで相互接続することによって構成される典型的な分散システムを主たる対象とし，実用性の面から関心を集めている動的負荷分散 (dynamic load balancing または adaptive load balancing) 方式を用いて，分散OSの導入や既存のOSの修正を行うことなく，分散システムの性能の向上をプロセスの平均応答時間の短縮によって実現することを主題とする．

動的負荷分散は，例えば実行中のプロセス数や資源の利用率など，システムの負荷情報を基に負荷の分散を図るものであり，分散システムへ適用する際にはどのような負荷情報をどのように収集するか，かつその情報を用いてどのように負荷分散を行うかが問題となる．更に，分散システムに実装する場合には，ネットワーク透過性や耐故障性などについても考慮する必要がある．本論文ではこれらの問題点に

ついて考察し，現実の分散システムに適した新しい負荷分散手法を提案する．

第2章においては，まず本研究の対象である分散システムについて概説し，同システムが持つ機能や特徴を明らかにする．また，主な負荷分散方針について概説し，本研究で採用したプロセス発送方式 (process dispatch) による動的負荷分散の妥当性を示す．

第3章においては，動的負荷分散の性能に大きく影響を与える負荷の尺度について論じる．本章では，分散システムで用いられているラウンドロビン (Round Robin, RR) スケジューリング方式を対象とし，新しいプロセスの割り当てによる平均応答時間の増加量を基準とした負荷の尺度を提案する．また，シミュレーションにより，残余仕事量の合計やプロセス数を用いた従来の尺度と比較し，本尺度の有効性を示す．

第4章においては，ネットワークを流れる通信量に大きな影響を与える負荷情報の収集方法について論じる．本章では，まず殆どの LAN が持つ同報通信機能に着目し，通信量を抑えながら各計算機の負荷情報を遅滞なく収集する方法として，プロセス割り当てや終了通知などの通信の傍受による負荷情報の収集手法を提案する．更に，シミュレーションにより従来の定期的同報通信方式や入札方式と比較し，本手法の有効性を示す．

第5章においては，耐故障性やネットワーク透過性を考慮した動的負荷分散機能の UNIX への実装方法について論じる．本章では，耐故障性を実現する方法として，同報通信に対するローカルホストの応答時間を基準としたホスト選択手順を提案する．また，ネットワーク透過性を実現する方法として，REX を用いたコマンド遠隔実行機能のコマンドインタープリタ (シェル) への実装を提案する．

第6章結論においては，本研究で得られた結果と残された問題についてまとめている．

第 2 章

分散システムと負荷分散

本章では，まず本研究の対象である分散システムについて概説し，同システムが持つ機能や特徴を明らかにする．また，主な負荷分散方式について概説し，本研究で採用した方式の妥当性を示す．

2.1 分散システム

分散システムを対象にして負荷分散を行う場合，分散システムがどのような機能や特徴を持っているかによって負荷分散の方針に影響を与える．例えば，分散システムに同報通信機能がなければ負荷情報の収集は 1 対 1 通信で行わなければならず，またネットワークの通信速度が小さければ頻繁に負荷情報の収集を行うことができない．

そこで，本節では分散システムの構成や典型的な分散システムが持つ機能・特徴などを明らかにする．

2.1.1 分散システムの構成

分散システムは疎結合型マルチプロセッサシステムを含めることもあるが，一般には複数の計算機を通信ネットワーク (communication network. 以下，ネットワーク) によって相互接続したシステムを指す．ネットワークに接続されている計算機や端末などの装置はノード (node) と呼ばれる．今日分散システムとして最も広く用

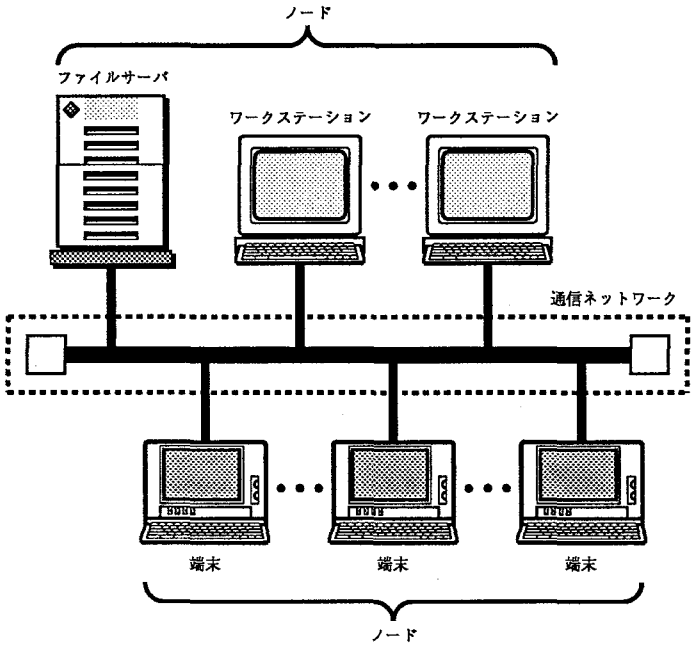


図 2.1: 典型的な分散システムの構成

いられているのは，ネットワークとして Ethernet などの LAN とノード計算機として UNIX ワークステーション (UNIX workstation) から構成されたものである．典型的な分散システムの構成を図 2.1 に示す．

LAN は，同一建物内あるいは同一構内において用いられるネットワークで，広域ネットワークや地域ネットワークなど他の種類のネットワークと比べると，近距離 (数 m ～数十 km) を高速 (数百 kbps ～数 Gbps) で通信する機能を持つ．現在最も普及している LAN である Ethernet は伝送媒体として同軸ケーブルなどを用いたバス型 LAN で，伝送速度は 10Mbps である．また，最近では光ファイバを用いたリング型 LAN でより高速 (伝送速度 100Mbps) な FDDI(Fibre Distributed Data Interface) が徐々に普及してきている．

2.1.2 分散システムの機能

本論文で対象としている分散システムは，典型的には OS として UNIX を用いており，種々の通信機能や資源の共有機能を備えている．ここでは，UNIX の機能のうち本研究と関連の深いものについて簡単に述べる．

(1) 基本通信機能

UNIX で標準的にサポートされているプロトコルとして，TCP/IP が挙げられる．TCP/IP では OSI 参照モデルのネットワーク層プロトコルとして IP(Internet Protocol)，トランスポート層プロトコルとして TCP(Transmission Control Protocol) 並びに UDP(User Datagram Protocol) プロトコルが用いられている．

TCP は信頼性のある双方向ストリーム型通信プロトコルで，パケットの喪失，重複などの通信エラーを修復する機能やフロー制御機能を持っている．一方，UDP はデータグラム型通信プロトコルで，信頼性は必ずしも保証されていないが，UDP では全てのノードに一斉にメッセージを送る同報通信 (ブロードキャスト，broadcast) 機能を利用することが可能である．この同報通信機能は，本研究では負荷情報の交換に用いられる．なお，同報通信機能と似た機能として，予めグループ化したノード群に一斉にメッセージを送るマルチキャスト (multicast) 機能があるが，この機能を標準的に利用できる UNIX は少ないため，本研究ではこの機能を用いていない．

(2) ファイルシステム共有機能

分散システムにおける資源の共有の１つとして，ファイルシステムの共有がある．現在 UNIX で用いられているファイルシステム共有機能として，NFS(Network File System)[8], RFS(Remote File Sharing)[9], AFS(Andrew File System)[10] などが挙げられる．このうち，特に NFS は最も普及しており，UNIX だけでなく MS-DOS や VMS など他の OS でもサポートされている．

例えば，NFS では遠隔マウント (remote mount) により他の計算機のディレクトリをローカルファイルシステム上にマウントしてディレクトリ単位でファイルの共有を行うことが可能で，これにより共有されたファイルを他のファイルと同様にパス名を用いてアクセスすることができるようになる．この様子を図 2.2に示す．また，RFS でも同様の方法によりファイルシステムの共有を行うことができる．

遠隔マウントによるファイルシステムの共有では，同じファイルシステムをノード毎に別々に遠隔マウントできるため，基本的にはファイルの位置透過性が確保されていないことになる．これは動的負荷分散を行うときに考慮すべき問題点である．

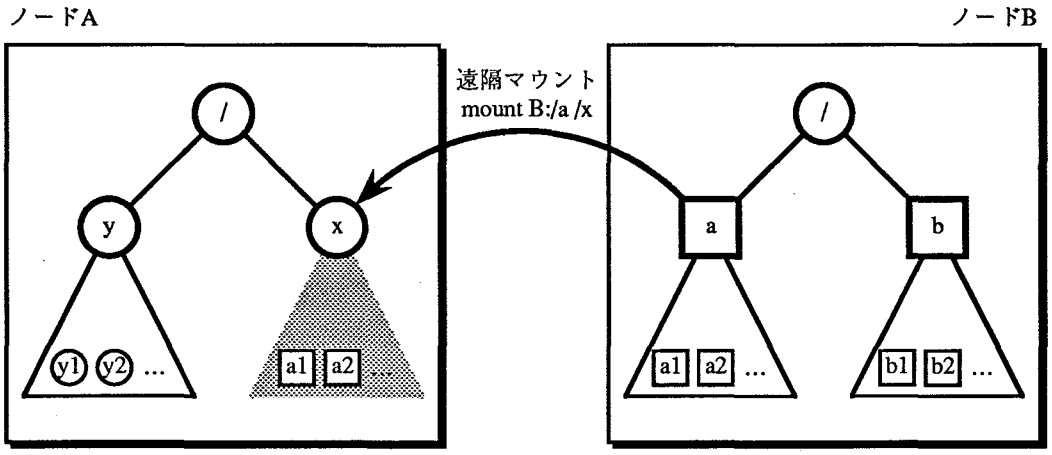


図 2.2: NFS によるファイルシステムの共有

(3) 遠隔実行機能

遠隔実行 (remote execution) は他の計算機 (リモートホスト (remote host)) 上でプログラムを実行することで，負荷分散を行う上で最も重要な機能である．標準的な UNIX では遠隔実行機能として，サーバでは rshd や rexecd，クライアントでは rsh，関数では rcmd や rexec が用意されている．

例えば，rsh を使ってコマンドを遠隔実行する場合，ユーザはコマンドラインにリモートホスト名とコマンド名，引数リストを指定する．すると，rsh は rcmd を用いて指定された計算機の rshd にコマンドの遠隔実行を依頼する．rshd はまず利用者認証を行い，これに成功すると依頼されたコマンドをホームディレクトリで実行する．コマンドの実行中，rsh と rshd の間には TCP によるバーチャルサーキット (virtual circuit) が設けられ，これを介してコマンドの標準入出力や標準エラー出力がやり取りされる．

コマンドの遠隔実行を行う場合，ネットワーク透過性が問題となる．すなわち，リモートホストでコマンドを実行しても自計算機 (ローカルホスト (local host)) での実行と同じ結果を得るためには，前述のファイルの位置透過性だけでなく，カレントディレクトリや環境変数の保存など，コマンドの実行環境を保存しなければならない．しかし，rsh など上記のプログラムや関数ではネットワーク透過性について考慮されていないため，ユーザは常にファイルのアクセスや実行環境を意識する必要がある．

```
struct rex_start {
        char    **rst_cmd;      /* list of command and args */
        char    *rst_host;      /* working directory host name */
        char    *rst_fsname;    /* working directory file system name */
        char    *rst_dirwithin; /* working directory within file system */
        char    **rst_env;      /* list of environment */
        u_short rst_port0;      /* port for stdin */
        u_short rst_port1;      /* port for stdout */
        u_short rst_port2;      /* port for stderr */
        u_long  rst_flags;      /* options */
};
```

図 2.3: REX のコマンド実行用プロトコル[1]

これに対して，最近では多くの UNIX で RPC(remote procedure call) サービスの１つとして REX[11] が利用できるようになってきた．REX では図 2.3に示すように，コマンド名や引数リストだけでなく，カレントディレクトリの計算機名，ファイルシステム名やマウントポイントからの相対パス名，及び環境変数リストなどを含むプロトコルが使われる．コマンドの遠隔実行は REX サーバ (rexd) で行われ，指定されたファイルシステムがマウントされていなければそれを自動的に NFS によりマウントし，カレントディレクトリと環境変数を設定した後に指定されたコマンドを実行する．この機能により，REX はコマンドの実行環境を保存し，カレントディレクトリを含むファイルシステムを参照する相対パス名に対する位置透過性を満足する．

なお，現在研究・開発されている多くの分散 OS では，実行中のプロセスを他の計算機に移動するプロセス移送 (process migration) 機能が実装されており，これを用いればより効果的な負荷分散が可能になると考えられる．しかし，この機能は現在のところ同一機種間に限られており，また標準的な UNIX ではこの機能は実装されていないため，本研究ではプロセス移送は用いない．

[1]SunOS 4.0.3 の/usr/include/rpcsvc/rex.h より引用

2.2 負荷分散手法

負荷分散の研究は古くから多くの研究者により進められており，種々の方針を基に多くのアルゴリズムが提案されてきている．しかし，これらのアルゴリズムには，現実の分散システムに適合しない仮定に基づく方針を用いているため，分散システムに実装するには問題があるものが多い．

そこで，本節ではこれらのアルゴリズムで用いられている負荷分散の方針について述べ，本研究で採用した負荷分散の方針の妥当性を示す．

2.2.1 負荷分散の目的

負荷分散は主に分散システムの性能向上を目的として行われるが，その性能評価基準は1つだけではない．代表的な性能評価基準としては，次のようなものを挙げることができる [39][40] .

1. プロセスの応答時間
2. プロセスの応答時間の分散 (variance)
3. システムのスループット (throughput)
4. 資源の利用率

これらの評価基準の多くは更に細分化することも可能である．例えば，1については，システムの全プロセスの平均応答時間の短縮を目的とするか (全体最適化)，あるいは負荷分散の対象となる個々のプロセスの応答時間の短縮を目的とするか (個別最適化) により細分化することができる [40] .

負荷分散を行う場合，ある程度まではこれらの評価基準のいずれの点からみても効果的であるが，一般には相容れないものが多い．従って，性能評価基準をどのように選ぶかは負荷分散の方針に影響を及ぼすことになる．本研究で対象としている分散システムはOSとしてUNIXを用いており，対話的に使われることが多いため，本研究では性能評価基準として主にシステムの全プロセスの平均応答時間を採用することにする．

2.2.2 代表的な負荷分散方式

負荷分散は，静的負荷分散 (static load balancing) と動的負荷分散 (dynamic load balancing または adaptive load balancing) の２つの方式に大別することができる．

このうち，静的負荷分散はプロセスの平均到着率や要求仕事量の分布など，予め与えられたプロセスの統計的な性質に基づいてノードの負荷を分散させる方式である．この方法はグラフ理論や待ち行列理論などを用いて数学的に解析できる場合が多く，システムの理論的な性能分析によく用いられている．代表的な静的負荷分散手法として，Optimal Static Load Balancing[22] が挙げられる．

一方，動的負荷分散はプロセッサやメモリの使用状況など，現在のシステムの状態に応じてノードの負荷を分散する方式である．この方法は静的負荷分散に比べて解析が難しく，主にシミュレーションにより性能評価が行われている．代表的な動的負荷分散手法として，JSQ(Join the Shortest Queue) 法 [12] が挙げられる．

現実のシステムでは，プロセスの統計的な性質は予測が困難であったり，状況によって変動したりすることが多い．そこで，負荷分散を現実のシステムに実装する場合には，実用性の面から動的負荷分散が注目されている．本研究においても，この理由により動的負荷分散を採用する．

2.2.3 動的負荷分散の方針

動的負荷分散は更にプロセス移送方式とプロセス発送 (process dispatch) 方式の２種類に分類することができる．このうち，プロセス発送方式は実行中のプロセスを移動させるのではなく，新しいプロセスを負荷の軽い計算機上に生成する方法による負荷分散方式である．2.1.2節で述べたように，標準的な UNIX ではプロセス移送機能は実装されていないため，本研究ではプロセス発送方式を用いることにする．

一般に，動的負荷分散方式を設計する場合には次に示す項目が問題となると考えられる．

1. どんな負荷情報を収集するか
2. どのように負荷情報を収集するか

3. 誰が負荷分散を行うか

4. いつ負荷分散を行うか

5. どのプロセスを移動するか

6. どこへ移動するか

本研究ではプロセス発送方式を用いるため，3, 4, 5については，新しいプロセスが到着するとローカルホストがそのプロセスを最も負荷の軽い計算機に割り当てることになる．本論文では残りの項目のうち，1と6については第3章で論じる．また，2については第4章で論じる．

2.3 分散システムのモデル

これまでに述べた分散システムの機能や性質，並びに負荷分散の方針を基に，第3章や第4章で共通に用いられる分散システムの基本的なモデルを以下のように定める．

本研究で対象とする分散システムは図2.4に示すようにN台のホスト計算機(host computer. 以下，ホスト)及びこれらを互いに接続する通信ネットワーク(以下，ネットワーク)から構成されているものとする．このシステムにおいて個々のホストを$h_1, h_2, \ldots, h_N$と書き表わすことにする．

UNIXでは，スケジューリング方式として優先度付きマルチレベルフィードバック方式が用いられている．この方式の動作を正確に解析することは困難であるが，UNIXでは時間が経過するにつれて長時間実行中のプロセスの優先度を低く，長時間実行待ちのプロセスの優先度を高くする時効化(aging)機能を持っているため，特にCPU制約(CPU bound)のプロセスに対してはラウンドロビン(RR)スケジューリング方式と見なすことができる[26][27]．

そこで，各ホストを図2.5のようにモデル化する．ユーザにより生成された新しいプロセスはシステム内のいずれかのホストに到着する．各ホストにはプロセスディスパッチャ(Process Dispatcher, PD)と呼ばれる特別のプロセスが1つ存在し，新しいプロセスを負荷分散アルゴリズムに従って最も負荷の小さいホストに割り当てる役割を果たす．PDは新しいプロセスがホストに到着すると直ちに起動される．PD

起動後から新しいプロセスの割り当てが終了するまでの間は，他のプロセスは処理されない．各ホスト内では，プロセッサは割り当てられたプロセスをRRスケジューリングに従って処理する．すなわち，ホストは待ち行列中の先頭のプロセスを最大で単位時間量 (time quantum) だけ処理する．単位時間量分の時間が経過しても処理が終了しない場合には，現在のプロセスの処理を中断してそのプロセスを待ち行列の最後尾に移動させ，次のプロセスを処理する．ホストで実行されている各プロセスはその要求仕事量に等しい処理を受けた後，システムから去る．

第3章や第4章ではこの基本モデルを基に更に種々の仮定を設けているが，これらについては必要に応じて各章で述べることにする．

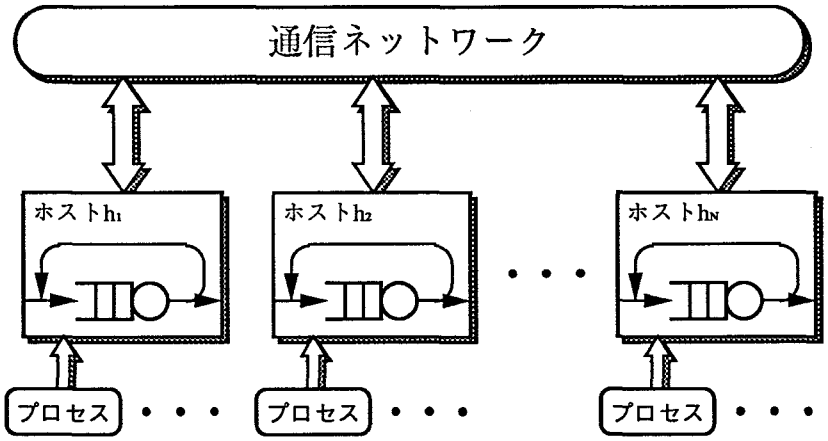


図 2.4: 分散システムのモデル

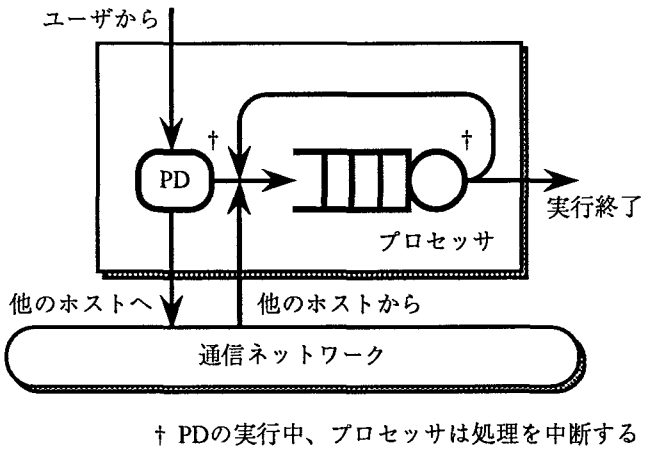


図 2.5: ホストのモデル

第 3 章

負荷の尺度

本章では，動的負荷分散の性能に大きく影響する負荷の尺度について述べる．本研究ではラウンドロビン (RR) スケジューリング方式を対象とし，プロセスの平均応答時間の短縮を目的とした新しい負荷の尺度を提案し，またシミュレーションにより本尺度の性能評価を行い，本尺度が従来の残余仕事量の合計やプロセス数による負荷の尺度より優れていることを示す．

3.1 負荷の尺度における問題点

動的負荷分散では，負荷の大きさに応じてプロセスを割り当てる計算機を決定するため，負荷の大きさの尺度が負荷分散の性能に大きく影響を与える．これまでに実行中のプロセス数，残余仕事量の合計，資源の利用率など多くの負荷の尺度が提案されてきたが，このうちどれが最も適切であるかは計算機システムの構成，OS のスケジューリング方式や負荷分散の目的などによって異なり，一概には定められない．また，負荷分散のオーバヘッドを減らすため，負荷の大きさの計算に要するコストも考慮する必要がある．

現実の計算機システムでよく用いられている OS のスケジューリング方式としては，FCFS(First Come First Served) 方式，ラウンドロビン (Round Robin, RR) 方式，フィードバック (FeedBack, FB) 方式などを挙げることができ，本研究の対象である UNIX システムでは 2.3節で述べたように RR 方式と見なすことができる．しかし，従来の研究 (例えば [12], [13] など) では大部分が FCFS 方式を対象としており，他のスケジューリング方式を対象としているものは少ない．

そこで，本研究ではRR方式を対象とし，プロセスの平均応答時間の短縮に適した負荷の尺度について検討する．

3.2 対象システムのモデル

3.2.1 計算機のモデル

本章では対象とする分散システムを2.3節で述べたようにモデル化し，更にこのモデルにおいて次のように仮定する．

1. プロセッサにおいてRRスケジューリングの単位時間量(time quantum)は十分小さく，プロセッサシェアリング(PS)スケジューリングと見なすことができる[29]．
2. PDを除く各プロセスは互いに独立であり，PD実行時を除いて，入出力を待つ，他のプロセスとの同期をとる，などのために実行を中断することはないものとする．
3. 新しいプロセスが到着した時点において，PDは各プロセスの要求仕事量の分布を知ることができるものとする．これについては3.2.2節で詳しく述べる．
4. 各プロセスの処理仕事量(finished work:既に受けた仕事量)は通信機能によりOSが管理しており，PDが自由に利用できるものとする．

なお，本尺度は，上記3で各プロセスの要求仕事量の分布の代わりに平均要求仕事量だけを求めることができる場合にも適用可能である．

3.2.2 要求仕事量の推定

本尺度では，後述するようにプロセッサの負荷の大きさの計算に各プロセスの要求仕事量の分布(求められない場合には，その平均でも可)を用いる．これらを得るために，本論文ではすべてのプロセスを例えば対応するコマンド名をもとにいくつかのグループに分類し，各グループ毎にプロセスの要求仕事量の統計をとることが

できるものと仮定する．現実の多くのシステムではこのような統計情報 (アカウント情報, accounting information) を課金のために自動的に収集・集計する機能が利用可能であるため，この仮定を満足させることは困難ではない．

これらの統計情報が得られると，PD は各プロセスの属するグループを知ることにより，そのプロセスの要求仕事量の分布 (あるいは平均) を経験的に求めることができる．例えばUNIX システムでは表3.1に示すようなコマンド毎の平均CPU 時間などを容易に得ることができる．

現実のシステムでは，負荷の大きさの計算において各プロセスの平均要求仕事量を求めるためのオーバヘッドの大きさが問題となる．しかし，例えばシステムを起動するときに集計結果から各グループに対する平均要求仕事量を求め，この表をメモリ上に置けば，このオーバヘッドは十分小さくすることができると考えられる．

表 3.1: UNIX システムにおける統計情報の一例

コマンド	平均 CPU 時間
find	2.74 (秒)
tar	1.73
rnews	1.16
nroff	0.73
⋮	⋮

3.3 記号の定義

次に示す各記法はある時刻 t におけるホスト並びにプロセスの状態を表現するために使われる．なお，特に断わらない限り，引数 t は省略する．

N_i: h_iで実行中のプロセス数.

P_{ik}: h_iで実行中の個々のプロセス．$1 \leq k \leq N_i$.

W_{ik}: P_{ik}の要求仕事量を表わす確率変数.

w_{ik}: P_{ik}の処理仕事量．P_{ik}が，その起動から時刻 t までに受けた仕事量.

U_{ik}: 時刻 t における P_{ik}の残余仕事量 (unfinished work) を表わす確率変数. $U_{ik} = W_{ik} - w_{ik}$.

e_{ik}: P_{ik}の起動から時刻 t までの時間. 以後, これを P_{ik}の経過時間 (elapsed time) と呼ぶ.

r_{ik}: 時刻 t 以降に到着するプロセスがないと仮定した場合の時刻 t から P_{ik} の終了までの時間を表わす確率変数. 以後, これを P_{ik}の暫定残余時間 (tentative remaining time) と呼ぶ.

R_{ik}: 時刻 t 以降に到着するプロセスがないと仮定した場合の P_{ik}の応答時間を表わす確率変数. 以後, これを P_{ik}の暫定応答時間 (tentative response time) と呼ぶ. $R_{ik} = e_{ik} + r_{ik}$.

3.4 プロセスの性質を考慮した負荷の尺度

3.4.1 負荷の定義

本節では, 新しいプロセスが時刻 t にシステム内のいずれかのホスト h_iに到着した場合を考え, このときの h_iの負荷の定義を行なう.

新しいプロセスをホスト h_iに割り当てた場合, PS スケジューリングの性質によりプロセス側から見た h_iの処理能力は低下し, その結果, h_i内の各プロセス P_{ik}の暫定応答時間が増加する.

平均応答時間をできるだけ小さくする方法として, 新しいプロセスをシステム内の全プロセス (新しいプロセスを含む) の応答時間の合計が最小になるように割り当てる方法が考えられる. しかし, プロセスの応答時間は将来に到着するプロセスの影響を受けるため, これを現時点で推定することは困難である. そこで, 本研究ではシステム内の全プロセスの暫定応答時間の合計 (以後, システムの総暫定応答時間と呼ぶ)

$$\sum_i \sum_k R_{ik} = \sum_i R_i$$

但し, (3.1)

$$R_i = \sum_k R_{ik} \qquad (h_i\text{の総暫定応答時間})$$

に着目する．もし，各プロセスの残余仕事量が既知であれば，この量を正確に求めることができるが，現実のシステムでは不可能である．本研究では，各プロセスの要求仕事量の分布からシステムの総暫定応答時間の期待値 $E[\sum R_i]$ を求め，新しいプロセスを $E[\sum R_i]$ が最小になるように割り当てる方法を用いる．

ここで，一般性を失うことなく新しいプロセスが h_i に割り当てられたと仮定し，その場合の h_i の総暫定応答時間 R_i の変化に注目する．h_i の総暫定応答時間 R_i は

$$\begin{aligned} R_i &= \sum_k R_{ik} \\ &= \sum_k (e_{ik} + r_{ik}) \\ &= \sum_k e_{ik} + \sum_k r_{ik} \\ &= e_i + r_i \end{aligned} \tag{3.2}$$

但し，

$$\begin{aligned} e_i &= \sum_k e_{ik} \qquad (h_i\text{の総経過時間}) \\ r_i &= \sum_k r_{ik} \qquad (h_i\text{の総暫定残余時間}) \end{aligned}$$

と表わせる．このうち，r_i は後述するように新しく到着したプロセスの影響を受けて増加する．一方，e_i はこの影響を受けず，変化しない．また，明らかに他のホストの総暫定応答時間も新しいプロセスの影響を受けない．従って，新しいプロセスの割り当てにより影響を受けるのは r_i だけである．その増加量を Δr_i とすると，これは同時に新しいプロセスを h_i へ割り当てたことにより生じるシステムの総暫定応答時間の増加量 ΔR_i も表わす．

新しいプロセスの割り当てにおいて，割り当て後の総暫定応答時間を最小にすることは割り当てによる総暫定応答時間の増加量を最小にすることと等価である．そこで，本研究では h_i の負荷の大きさ L_i を h_i の総暫定残余時間の増加量の期待値に等しく，

$$L_i = E[\Delta r_i] \tag{3.3}$$

と定義する．

3.4.2 負荷の導出

(1) 要求仕事量が一般分布に従う場合の負荷

各プロセスの要求仕事量が既知であると仮定し，式 (3.3) からその場合の h_i の負荷 L_i を導出する．

まず，新しいプロセスが割り当てられる直前における h_i の状態を考える．

h_i 内の任意のプロセス P_{ik} に着目すると，その暫定残余時間 r_{ik} は

$$\begin{aligned} r_{ik} &= \frac{1}{\mu_i}(P_{ik}\text{の終了までに}\ h_i\ \text{で行われる仕事量}) \\ &= \frac{1}{\mu_i}\sum_{m=1}^{N_i}(P_{ik}\text{の終了までに}\ P_{im}\ \text{が受ける仕事量}) \end{aligned} \tag{3.4}$$

と表わせる．ここで，μ_i はホスト h_i の処理能力である．PS スケジューリングでは一定の時間にホストが各プロセスに対して行なう仕事量は等しいから，式 (3.4) の「P_{ik} の終了までに P_{im} が受ける仕事量」は，2 つのプロセス P_{ik}, P_{im} の残余仕事量 U_{ik}, U_{im} の間の関係により次のようになる．

1. $U_{ik} < U_{im}$ ならば，P_{ik} が先に終了し，P_{im} が受ける仕事量は U_{ik} に等しい (図 3.1(a))．
2. $U_{ik} > U_{im}$ ならば，P_{im} が先に終了し，P_{im} が受ける仕事量は U_{im} に等しい (図 3.1(b))．
3. $U_{ik} = U_{im}$ ならば，P_{ik} と P_{im} は同時に終了し，P_{im} が受ける仕事量は $U_{ik}(= U_{im})$ に等しい (図 3.1(c))．

これらをまとめると，「P_{ik} の終了までに P_{im} が受ける仕事量」は $\min(U_{ik}, U_{im})$ と書ける．

これを用いると，r_{ik} は

$$r_{ik} = \frac{1}{\mu_i}\sum_{m=1}^{N_i}\min(U_{ik}, U_{im}) \tag{3.5}$$

となる．また，h_i の総暫定応答時間 r_i は式 (3.3) より

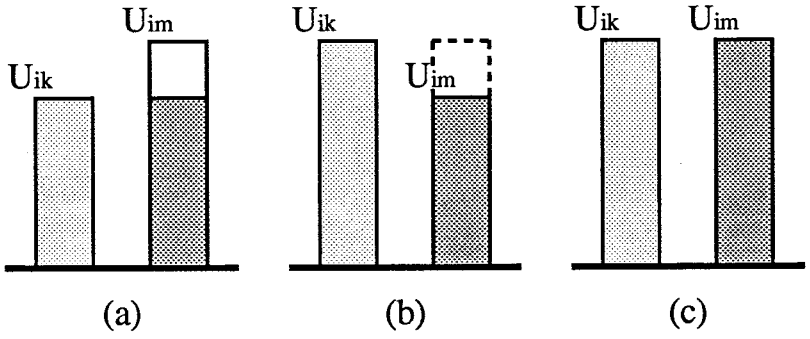


図 3.1: 時刻 t から P_{ik} の終了までに P_{im} の受ける仕事量
(a) $U_{ik} < U_{im}$　(b) $U_{ik} > U_{im}$　(c) $U_{ik} = U_{im}$

$$\begin{aligned} r_i &= \sum_{k=1}^{N_i} r_{ik} \\ &= \frac{1}{\mu_i} \sum_{k=1}^{N_i} \sum_{m=1}^{N_i} \min(U_{ik}, U_{im}) \end{aligned} \tag{3.6}$$

となる.

次に，新しいプロセスが h_i に割り当てられた場合を考え，各プロセスの暫定残余時間 r_{ik} 並びに h_i の総暫定残余時間 r_i の変化を調べる.

新しいプロセスが h_i に割り当てられると，PS スケジューリングの性質により個々のプロセスに対する処理能力は低下し，その結果，各プロセスの暫定応答時間 r_{ik} は増加する．新しいプロセスを P_{i0} と表わすと，P_{ik} の暫定残余時間 r_{ik} の増加量 Δr_{ik} は P_{i0} の実行により P_{ik} が待たされる時間の合計に等しく，図 3.1 の場合と同様の考察により，

$$\begin{aligned} \Delta r_{ik} &= (P_{i0}\text{の実行中に } P_{ik}\text{が待つ時間の合計}) \\ &= \frac{1}{\mu_i} \sum_{m=1}^{N_i} (P_{ik}\text{の終了までに } P_{i0}\text{が受ける仕事量}) \\ &= \frac{1}{\mu_i} \min(U_{i0}, U_{ik}) \\ &= \frac{1}{\mu_i} \min(W_{i0}, U_{ik}) \end{aligned} \tag{3.7}$$

と表わせる[1]．ここで，W_{i0} は P_{i0} の要求仕事量で，同時に現時点における P_{i0} の残余

[1]文献 [14] にも同様の式が示されている.

仕事量U_{i0}に等しい．また，h_iの総暫定残余時間r_iの増加量Δr_iは，

$$\begin{aligned}
\Delta r_i &= (P_{i0}\text{割り当て後の } r_i) - (P_{i0}\text{割り当て後の } r_i) \\
&= \frac{1}{\mu_i}\sum_{k=0}^{N_i}\sum_{m=0}^{N_i}\min(U_{ik}, U_{im}) \\
&\quad -\frac{1}{\mu_i}\sum_{k=1}^{N_i}\sum_{m=1}^{N_i}\min(U_{ik}, U_{im}) \\
&= \frac{1}{\mu_i}\left(W_{i0} + 2\sum_{k=1}^{N_i}\min(W_{i0}, U_{ik})\right) \\
&= \frac{W_{i0}}{\mu_i} + 2\sum_{k=1}^{N_i}\Delta r_{ik} \qquad (3.8)
\end{aligned}$$

と表わせる．

次に，式(3.8)の期待値

$$E[\Delta r_i] = \frac{E[W_{i0}]}{\mu_i} + 2\sum_{k=1}^{N_i}E[\Delta r_{ik}] \qquad (3.9)$$

を求める．

ここで，一般に確率変数$X_1, \ldots, X_i, \ldots$が互いに独立であるならば，

$$E[\min_i X_i] = \int_0^\infty \prod_i P[X_i > w]\, dw \qquad (3.10)$$

が成り立つ[28]．この式を用いて式(3.7)の期待値を表わすと，

$$\begin{aligned}
E[\Delta r_{ik}] &= \frac{1}{\mu_i}E[\min(W_{i0}, U_{ik})] \\
&= \frac{1}{\mu_i}\int_0^\infty P[W_{i0} > w]\, P[U_{ik} > w]\, dw \qquad (3.11)
\end{aligned}$$

となる．式(3.11)においてU_{ik}はプロセスP_{ik}の残余仕事量であるから，時刻tにおける$P[U_{ik} > w]$は次のように書ける．

$$\begin{aligned}
P[U_{ik} > w] &= P[W_{ik} > w + w_{ik} \mid W_{ik} > w_{ik}] \\
&= \frac{P[W_{ik} > w + w_{ik}]}{P[W_{ik} > w_{ik}]} \qquad (3.12)
\end{aligned}$$

ここで，w_{ik}はプロセスP_{ik}の処理仕事量である．

式(3.12)を式(3.11)に代入すると，次の式を得る．

$$\begin{aligned} E[\Delta r_{ik}] &= \frac{1}{\mu_i} E[\min(W_{i0}, U_{ik})] \\ &= \frac{1}{\mu_i} \int_0^\infty P[W_{i0} > w] \frac{P[W_{ik} > w + w_{ik}]}{P[W_{ik} > w_{ik}]} \, dw \end{aligned} \tag{3.13}$$

更に，式(3.13)を式(3.9)に代入すると，

$$E[\Delta r_i] = \frac{1}{\mu_i} \left(E[W_{i0}] + 2 \sum_{k=1}^{N_i} \int_0^\infty P[W_{i0} > w] \frac{P[W_{ik} > w + w_{ik}]}{P[W_{ik} > w_{ik}]} \, dw \right) \tag{3.14}$$

を得る．式(3.3)で定義したように，この式は各プロセスの要求仕事量が一般分布に従う場合のh_iの負荷の大きさL_iを表わす．

(2) 要求仕事量が指数分布に従う場合の負荷

残念ながら，現実のシステムでは，各プロセスの要求仕事量の平均を知ることは容易であるが，その分布を知ることは困難である場合が多い．また，例え分布を知ることが可能でも，一般には式(3.14)を用いて各ホストの負荷を計算するにはかなりの時間が必要となり，実用的ではないと考えられる．

そこで，特別な場合として，各プロセスの平均要求仕事量$\overline{W}_{ik}$だけが3.2.2節で述べた統計から求められる場合を考え，このとき，要求仕事量W_{ik}がパラメータ$1/\overline{W}_{ik}$の指数分布に従うものと見なす．すると，

$$P[W_{ik} > w] = \exp\left(-\frac{w}{\overline{W}_{ik}}\right) \tag{3.15}$$

と表わせる．また，指数分布の無記憶性から，

$$\begin{aligned} P[U_{ik} > w] &= P[W_{ik} > w] \\ &= \exp\left(-\frac{w}{\overline{W}_{ik}}\right) \end{aligned} \tag{3.16}$$

が成り立つ．式(3.15), (3.16)を式(3.11)に代入すると，

$$
\begin{aligned}
E[\Delta r_{ik}] &= \frac{1}{\mu_i}\int_0^\infty \exp\left(-\frac{w}{\overline{W}_{i0}}\right)\exp\left(-\frac{w}{\overline{W}_{ik}}\right)dw \\
&= \frac{1}{\mu_i}\int_0^\infty \exp\left(-w\left(\frac{1}{\overline{W}_{i0}}+\frac{1}{\overline{W}_{ik}}\right)\right)dw \\
&= \frac{\overline{W}_{i0}\overline{W}_{ik}}{\mu_i(\overline{W}_{i0}+\overline{W}_{ik})}
\end{aligned}
\tag{3.17}
$$

を得る．更に，式(3.17)を式(3.9)に代入すると，

$$
E[\Delta r_i] = \frac{1}{\mu_i}\left(\overline{W}_{i0}+2\sum_{k=1}^{N_i}\frac{\overline{W}_{i0}\overline{W}_{ik}}{\overline{W}_{i0}+\overline{W}_{ik}}\right) \tag{3.18}
$$

を得る．この式は各プロセスの要求仕事量が指数分布に従うと見なした場合のh_iの負荷の大きさ$L_i = E[\Delta r_i]$を表わす．

なお，この式では，式(3.14)とは異なり，指数分布の無記憶性により各プロセスの処理仕事量w_{ik}を用いる必要がない．

3.5 評価と考察

3.5.1 性能評価の概要

3.4節で示した負荷の尺度(以下，MIR(Minimum Increment of Response time)と表記する)の有効性を評価するため，4台及び10台のホストを相互接続した分散システムについてシミュレーションを行なった．なお，このシミュレーションでは各プロセスの平均要求仕事量だけが求められる場合を想定し，各ホストの負荷は，式(3.18)を用いて計算した．また，次の2つの負荷の尺度についても同様のシミュレーションを行い，性能を比較した．

- プロセス数
  負荷の大きさは$L_i = N_i$で表される．以下，JSQ(Join the Shorest Queue)と表記する．

- 残余仕事量の合計
  負荷の大きさは$L_i = \sum U_{ik}$で表される．シミュレーションでは，各プロセス

の要求仕事量は指数分布すると仮定し，$L_i = \sum U_{ik} = \sum W_{ik}$とした．以下，LUW(Least Unfinished Work) と表記する．

各ホストの処理モデルは3.2節で述べたものを用いたが，以下の実験では更に次のように仮定した．

1. すべてのホストは同じ処理能力を持っており，$1 \leq i \leq N$のすべてのiに対して，$\mu_i = 1$であるとする．この仮定より，以降の議論では仕事量の単位として秒を用いる．

2. 各ホストへのプロセスの到着間隔 (interarrival time) は指数分布 (exponential distribution) あるいは2ステージ超指数分布 (2 stage hyperexponential distribution) に従うものとし，すべてのホストに対するプロセスの到着分布は等しいものとする．

3. PD が各ホストの負荷の大きさを計算するために要するオーバヘッドは，負荷の尺度，プロセスの到着分布，各ホストの状態などによらず一定であるとする．

4. いずれの尺度においても負荷の大きさが最小であるホストが2つ以上あった場合には，それらのうちの1つを無作為に選び，そのホストに新しいプロセスを割り当てるものとする．

式 (3.18) の導出では，コマンド名により分類した各グループに属するプロセスの要求仕事量が指数分布に従うものと仮定したが，実際のシステムではこの仮定が成り立つことは稀である．そこで，現実のシステムのふるまいに近いシミュレーションを行なうため，シミュレーションに先立ち実際に稼働中のシステムでコマンド毎の要求仕事量の分布を約3カ月間調査した．調査の対象としたシステムは VAX-11/750 上で稼働している 4.2 BSD UNIX で，主にプログラム開発のために使われているものである．分布の集計は，このシステムの統計情報収集プログラム acct 並びに集計プログラム sa(後者はこの目的のために多少の変更を加えた) を用いて行なった．その後，集計結果をもとに，全コマンドを要求仕事量の平均や分散係数に基づき9グループに分類し，各グループに属するプロセス数の合計を求めた．その結果を表3.2に示す．

表3.2に基づき，シミュレーションで用いるプロセスの特性を表3.3に示すように決定した．この表はシステムに到着する新しいプロセスが各グループに属する確率

を示し，すべてのシミュレーションで共通である．なお，システム全体から見た要求仕事量の平均$\overline{W}$は0.0896秒で，その二乗分散係数C_W^2は10.6である．

シミュレーションでは，ホスト数Nを4, 10の2通り，各ホストへのプロセス全体の平均到着率$\overline{\lambda}$を$0.1/\overline{W}$, $0.3/\overline{W}$, $0.5/\overline{W}$, $0.7/\overline{W}$, $0.8/\overline{W}$の5通り，到着間隔の分散係数C_aを1, 3, 5の3通りに変え，これらを組み合わせた計30通りについて各尺度を用いた場合の平均応答時間を測定した．また，到着間隔の分散係数C_aが3のときの各場合については応答時間の分散についても測定した．すべてのシミュレーションを通して，RRスケジューリングの単位時間量並びに各ホストの負荷の大きさの計算に要するオーバヘッドはいずれも0.001秒とした．それぞれのシミュレーションでは50回の独立試行を行ない，各試行ではプロセスの応答時間を開始10秒後から200秒後まで集計した．

3.5.2 実験結果と考察

まず，プロセスの平均応答時間については，それぞれの負荷の尺度に対する平均応答時間を表3.4に，JSQに対する他の尺度の平均応答時間の比率を図3.2に示す．また，到着間隔の分散係数C_aが3のときのプロセスの応答時間の分散については，それぞれの負荷の尺度に対する分散を表3.5に，JSQに対する他の尺度の分散の比率を図3.3に示す．

これらの図表より，平均応答時間，応答時間の分散のいずれについてもJSQよりもLUWの方が，またLUWよりも本尺度の方が優れており，本尺度はJSQと比較すると平均応答時間については最大で5%程度，応答時間の分散については最大で20%程度改善されていることがわかる．また，いずれの場合でも平均到着率がある程度大きい時に本尺度はJSQと比べて高い改善率を示していることがわかる．

この現象の理由は次のように推測できる．

平均到着率が比較的小さい($N=4$では$\overline{\lambda} \leq 0.3/\overline{W}$，$N=10$では$\overline{\lambda} \leq 0.5/\overline{W}$)場合には，新しいプロセスが到着したとき，実行するプロセスのないホストが存在する確率が比較的高い．この場合，いずれの尺度を用いてもプロセスのない同一のホストに新しいプロセスを割り当てるため，負荷の尺度の違いが殆ど平均応答時間や応答時間の分散に反映しないと考えられる．

表 3.2: ある 4.2 BSD UNIX システムのプロセス特性

プロセス数		平均要求仕事量 (秒)		
		小	中	大
		$E[W] \leq 0.01$	$0.01 < E[W] \leq 0.1$	$0.1 < E[W]$
分	$C_W^2 \leq 0.5$	27966	26773	5013
	$0.5 < C_W^2 \leq 1.5$	201	8603	33535
布	$1.5 < C_W^2$	151430	31978	1495

W : コマンドの要求仕事量
C_W^2: コマンドの要求仕事量の二乗分散係数

表 3.3: シミュレーションで用いたプロセス特性

確率 (%)		平均要求仕事量 (秒)		
		小	中	大
		$E[W] = 0.005$	$E[W] = 0.05$	$E[W] = 0.5$
分	$D\,(C_W = 0)$	10	9	2
	$M\,(C_W = 1)$	0	3	12
布	$H_2\,(C_W = 3)$	52	11	1

W : コマンドの要求仕事量
C_W: コマンドの要求仕事量の分散係数
D : 一定分布
M : 指数分布
H_2 : 2 ステージ超指数分布

表 3.4: 各尺度に対するプロセスの平均応答時間

(a) ホスト数が 4 台の場合

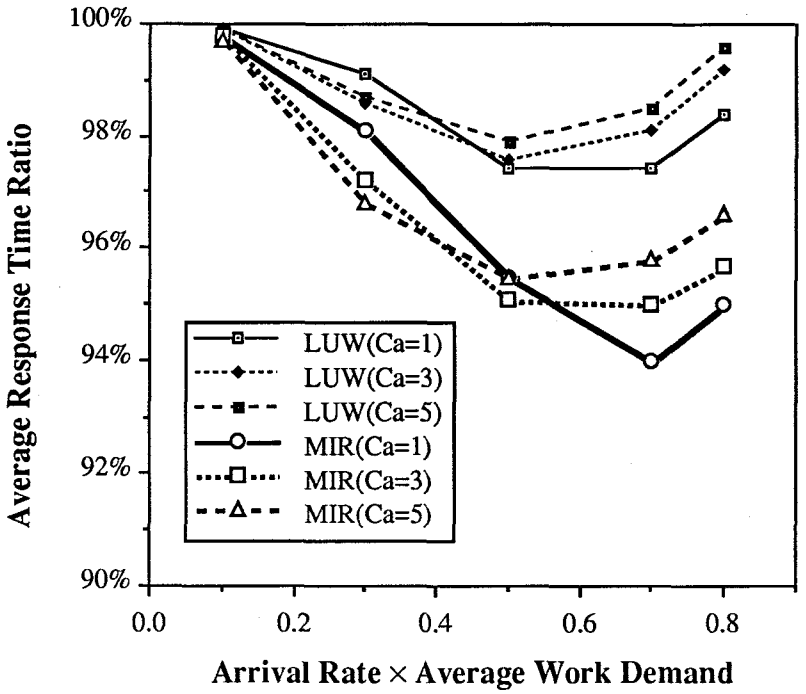
応答時間 (sec) 上段: JSQ 中段: LUW 下段: MIR		平均到着率 × 平均要求仕事量				
		0.1	0.3	0.5	0.7	0.8
到着分布	$C_a = 1$	0.0970	0.0990	0.1133	0.1525	0.2065
		0.0969	0.0981	0.1104	0.1485	0.2032
		0.0968	0.0970	0.1082	0.1434	0.1961
	$C_a = 3$	0.0953	0.1013	0.1233	0.1833	0.2499
		0.0952	0.0999	0.1204	0.1798	0.2478
		0.0951	0.0985	0.1172	0.1741	0.2391
	$C_a = 5$	0.0966	0.1028	0.1325	0.2295	0.3342
		0.0965	0.1015	0.1297	0.2268	0.3329
		0.0963	0.9955	0.1266	0.2198	0.3228

C_a: 到着間隔の分散係数

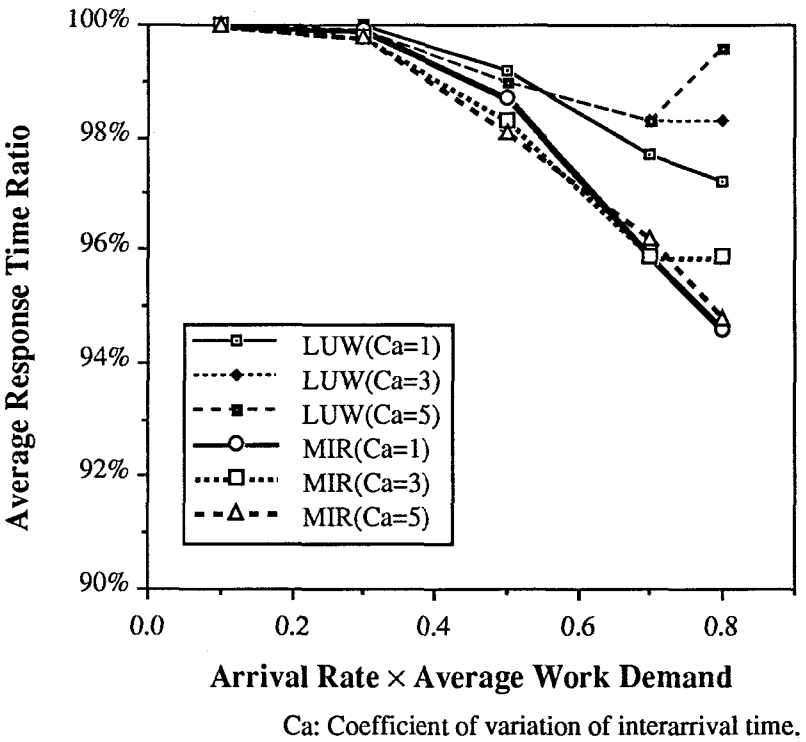
(b) ホスト数が 10 台の場合

応答時間 (sec) 上段: JSQ 中段: LUW 下段: MIR		平均到着率 × 平均要求仕事量				
		0.1	0.3	0.5	0.7	0.8
到着分布	$C_a = 1$	0.0936	0.0934	0.0958	0.1111	0.1332
		0.0936	0.0933	0.0951	0.1086	0.1295
		0.0936	0.0933	0.0946	0.1065	0.1260
	$C_a = 3$	0.0943	0.0925	0.0978	0.1205	0.1522
		0.0943	0.0923	0.0968	0.1185	0.1496
		0.0943	0.0923	0.0961	0.1156	0.1459
	$C_a = 5$	0.0943	0.0925	0.0991	0.1325	0.1877
		0.0943	0.0924	0.0981	0.1303	0.1870
		0.0943	0.0923	0.0972	0.1274	0.1779

C_a: 到着間隔の分散係数



(a) ホスト数が 4 台のとき



(b) ホスト数が 10 台のとき

図 3.2: JSQ に対する他の尺度の平均応答時間の比率

表 3.5: 各尺度に対するプロセスの応答時間の分散

分散 (sec^2) 上段: JSQ 中段: LUW 下段: MIR		平均到着率 × 平均要求仕事量				
		0.1	0.3	0.5	0.7	0.8
ホスト数	$N = 4$	0.083	0.107	0.157	0.394	0.753
		0.083	0.102	0.149	0.363	0.718
		0.083	0.095	0.131	0.328	0.665
	$N = 10$	0.092	0.083	0.096	0.162	0.269
		0.092	0.083	0.092	0.145	0.238
		0.092	0.082	0.088	0.126	0.212

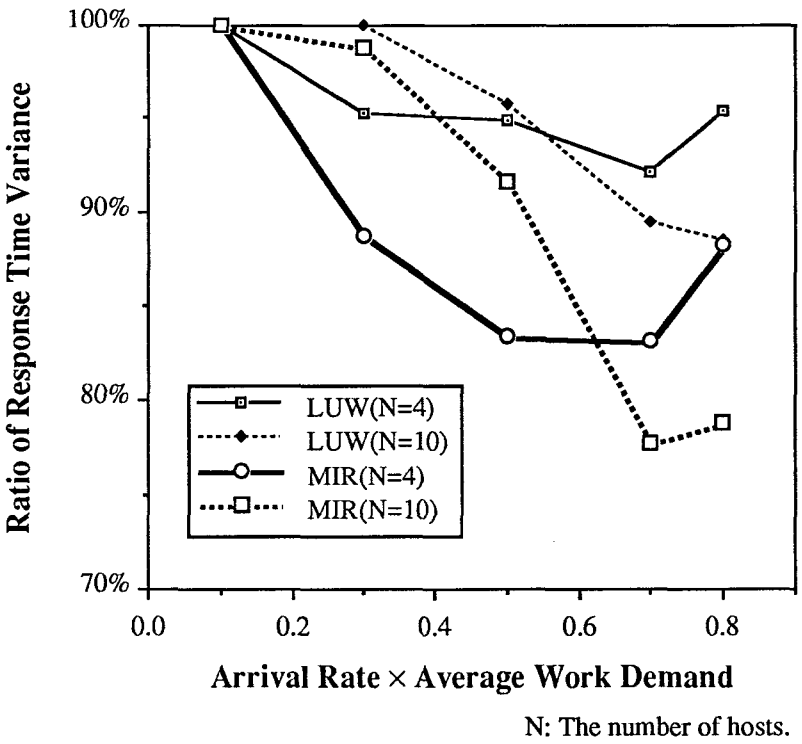


図 3.3: JSQ に対する他の尺度の応答時間の分散の比率

ホスト数が10台で平均到着率が比較的小さいときには全般的に3つの負荷の尺度の違いはホスト数が4台のときと比べて小さいが，これはホスト数が多いとプロセスのないホストが存在する確率が高くなるためと思われる．

一方，平均到着率が比較的大きい場合 ($N = 4$ では$\overline{\lambda} \geq 0.5/\overline{W}$，$N = 10$ では$\overline{\lambda} \geq 0.7/\overline{W}$) には，プロセスのない計算機が存在する確率が低くなるため，同じ状況においても負荷の尺度により選択されるホストが異なる場合が多くなり，負荷の尺度の違いが比較的大きく応答時間の平均や分散に影響したと考えられる．

なお，ホスト数が4台のときには$\overline{\lambda} = 0.7/\overline{W}$付近で平均到着率が大きくなると平均応答時間の改善率が減少する現象が見られるが，これは各ホストの負荷がかなり大きくなっているため，どのホスト上で新しいプロセスを実行しても大差がない場合が多く，その結果負荷の尺度の違いが応答時間の平均や分散に与える影響が小さくなったと考えられる．

3.6 まとめ

本章では分散システムで用いられているRRスケジューリング方式を対象とし，新しいプロセスの割り当てによる平均応答時間の増加量を基にした負荷の尺度を提案した．

本尺度の有効性については，シミュレーションによって平均応答時間，応答時間の分散のいずれについてもJSQやLUWよりも優れており，JSQと比較すると平均応答時間については最大で5%程度，応答時間の分散については最大で20%程度改善されていることが確認された．特に，ホスト数が少なく，平均到着率がある程度大きい時にJSQに対する改善率が高いことがわかった．

本尺度の他の特徴として，多くの分散システムにおいて実装が容易である点が挙げられる．すなわち，本尺度では各プロセスの要求仕事量の期待値を用いる必要があるが，多くのOSではこの値を統計情報収集・集計機能を用いて比較的容易に求めることができるため，非UNIX系の分散システムにも適用することが可能である．

今後の課題としては，本尺度の解析的手法による性能評価がある．また，現実のシステムではCPU制約のプロセスだけではなく，I/O制約 (I/O bound) のプロセス，対話的 (interactive) なプロセスやデーモンプロセス (daemon process) など，3.2.1節

の仮定 (特に 2) が満足されているとは限らないため，これらのプロセスを含んでいる場合にも適するように負荷の尺度を改良することも今後の課題の１つである．

更に，本尺度は新しいプロセスが到着する度に全てのホスト上の全てのプロセスに関する情報を用いるので情報収集を頻繁に行う必要があり，ネットワークに大きな負荷をかけたり，負荷情報を収集するためのオーバヘッドが大きくなったりすることが予想される．

そこで，この問題に対処するため，次章では負荷情報の効率的な収集方法について論じる．

第 4 章

負荷情報の収集

本章では，ネットワークの負荷に大きな影響を与える負荷情報の収集方法について述べる．本研究では，まず殆どのLANが持つ同報通信機能に着目し，ネットワークの負荷を抑えながら各ホストの負荷情報を遅滞なく収集する方法を提案する．また，シミュレーションにより本方法の性能評価を行い，本方法が従来の定期的同報通信 (periodical broadcast) 方式や入札 (bidding) 方式より優れていることを示す．

4.1 負荷情報の収集における問題点

動的負荷分散では，各ホストの現在の状況に応じて負荷を分散するため，各ホストの負荷の大きさを収集する必要がある．その方法として，各ホストが定期的に自分自身の現在の負荷の大きさを同報通信 (broadcast) を用いて他のホストに知らせる定期的同報通信 (periodical broadcast) 方式がよく用いられる．しかし，この方法は次のような問題点があることが知られている [25] .

1. 全てのホストが同報通信機能を持っている必要がある．
2. 負荷の大きさを知らせるために要する通信量がかなり大きい．
3. 同時に同一の (以前は) 負荷の軽かったホストに負荷を与えようと試みる危険性が大きい．

このうち，1に関しては，本研究で対象としている典型的な分散システムでは同報通信機能を備えており，問題点にはならない．

一方，2と3はトレードオフ (trade-off) の関係にあり，同一のホストへの負荷の集中を避けるためには負荷情報の送信を頻繁に行わなければならず，逆に通信量を抑えるために同報通信の間隔を大きくすると多くのプロセスが同一のホストに集中することになる．

これに対して，Hsu と Liu は確率的なプロセス割り当てアルゴリズム [13] を提案した．このアルゴリズムでは負荷の大きいホストほどプロセスを受け取る確率を低く設定しており，またプロセスを送る度にこの確率を増減させることにより同一ホストへのプロセスの集中を避けるものである．しかし，このアルゴリズムでは負荷の大きいホストへプロセスを送る可能性もあるため，適切な負荷分散であるとはいい難い．

また，別の動的負荷分散手法として，入札 (bidding) 方式がよく知られている．これは，新たにプロセスを生成するときに各ホストに対して負荷を問い合わせ，これに対する応答を比較して最も適していると思われるホストに新しいプロセスを割り当てる方法である [15][16] ．この方法では3の問題点を避けることができるが，問い合わせに対して各ホストから応答があるため，ホストの台数を n とするとシステム全体での通信量は $O(n^2)$ となり，定期的同報通信方式の $O(n)$ に比べるとネットワークに対する負荷が大きいといえる．また，この方法ではどの時点で負荷情報の問い合わせに対する応答の受付を打ち切るかが問題となる．すなわち，応答受付時間が長過ぎるとコマンドが入力されてから実行されるまでの時間が長くなり，逆に応答受付時間が短過ぎると負荷の最も軽いホストが時間内に応答できずに無視される可能性が高くなるという問題点がある．

そこで，本章では，典型的な分散システムにおいて通信量を抑えながら各ホストの負荷情報を遅滞なく収集する方法を提案する．本方法では負荷分散に不可欠なメッセージを同報することにより負荷情報の送信とプロセス割り当てを同時に行うため，上記の2と3の問題点を同時に避けることが可能である．

4.2 対象システムのモデル

本章では対象とする分散システムを 2.3節で述べたようにモデル化し，更にこのモデルにおいて次のように仮定する．

1. 全てのホストは等しい処理能力を持つものとする．この仮定により，以降の議論では仕事量の単位として秒を用いることにする．

2. PD の起動中やメッセージの送受信 (傍受を含む) 中はプロセスは実行されないものとする．

3. PD を除く各プロセスは互いに独立であり，PD 実行時を除いて，入出力を待つ，他のプロセスとの同期をとる，などのために実行を中断することはないものとする．

このモデルにおいて，ネットワークは同報通信機能を持ち，本来は特定のホストに対するメッセージを全て同報することにより，各ホストが任意のメッセージの内容を参照することができる．この場合，各メッセージには送信元や本来の送信先が含まれており，各ホストはこれらも参照できるものとする．

以下では，各ホストがメッセージの内容を参照する場合，そのメッセージの本来の送信先が自ホストあるいは全ホストであれば「受信する」と書き，他ホストであれば「傍受する」と書き表す．また，各ホストがメッセージを同報する場合，そのメッセージの本来の送信先が h_i であれば「h_i に送信する」と書き，全ホストであれば「同報する」と書き表す．

ユーザが与えたコマンドは PD により現在のホスト (ローカルホスト (local host)) あるいは他のホスト (リモートホスト (remote host)) の何れかに割り当てられるが，特に新しいプロセスをリモートホストに割り当てる場合，PD は割り当てメッセージ (assignment message) をそのホストに向けて送信する．

以下では，プロセスが割り当てられたホストをそのプロセスの実行ホスト (processing host) と呼ぶ．また，ローカルホストと実行ホストが同じプロセスをローカルプロセス (local process)，これらが異なるプロセスをリモートプロセス (remote process) と呼ぶ．

また，リモートプロセスが終了した場合，ホストはその終了通知あるいはその実行結果を終了メッセージ (termination message) としてそのプロセスのローカルホストに向けて送信し，ローカルホストはそれを受信すると直ちにユーザに知らせる．なお，本論文ではプロセスの応答時間をユーザがコマンドを与えてから結果を受け取るまでの時間であると定義する．

4.3 効率的な負荷情報の収集方法

4.3.1 負荷の推定

本方法は多くの負荷の尺度に対して適用できるが，本章では議論を簡単化するため，JSQ(Join the Shortest Queue) 法 [12] と同様に，負荷の尺度として実行中のプロセス数を用いる．従って，以下の議論では通信量をあまり増やすことなく如何に正確に各ホストにおける実行中のプロセス数を推定するかが問題となる．なお，プロセス数の代わりに，第3章で述べた負荷の尺度を用いる場合については後述する．

以後，ホスト h_i において推定したホスト h_j の負荷の大きさ (すなわち，ホスト h_j で実行中のプロセス数) を $L_i(j)$ と書き表すことにする．

従来の方法では，各ホスト h_i は自ホストの負荷の大きさ $L_i(i)$ をそのまま負荷情報として他のホストに知らせるものが多い．しかし，本方法では $L_i(i)$ 自身ではなくその増減を負荷情報として他のホストに知らせている．各ホストのプロセス数は，ホストの起動時あるいは一旦稼働を停止したホストの稼働再開時に同報通信で他のホストのプロセス数を問い合わせる等の方法を用いることにより初期化できるので，本アルゴリズムの方法は $L_i(i)$ をそのまま知らせる方法と等価である．

ところで，プロセス割り当てによる負荷分散では，4.2節で述べた 2 種類のメッセージ，すなわち割り当てメッセージと終了メッセージが不可欠である．これらのメッセージが送られるのはそれぞれリモートプロセスの割り当て時，終了時でありかつそのときに限るので，これらのメッセージを同報し他ホストでそれを傍受すれば，リモートプロセスの割り当て，終了による負荷の大きさの増減をどのホストにおいても直ちに知ることができる．すなわち，ホスト h_j に向けて割り当てメッセージが送信された場合，別のホスト h_i はこのメッセージを傍受して，新しいプロセスがホスト h_j に割り当てられたことを知り，ホスト h_j の負荷の大きさ $L_i(j)$ を 1 つ増やすことができる．また，ホスト h_j から終了メッセージが送信された場合には，ホスト h_i はこのメッセージを傍受して，ホスト h_j に割り当てられていたリモートプロセスが終了したことを知り，$L_i(j)$ を 1 つ減らすことができる．なお，この方法では既存のメッセージを同報しそれを他ホストで傍受するだけであるため，メッセージ数は増えていない点に注意する．

一方，ローカルプロセスの割り当て，終了に起因する負荷の大きさの増減に関し

ては，割り当てメッセージや終了メッセージが送られないため，このままでは他ホストは負荷の増減を知ることができない．そこで新しいプロセスのローカルホストへの割り当てを知らせるローカル割り当てメッセージ (local assignment message) 並びにローカルプロセスの終了を知らせるローカル終了メッセージ (local termination message) の 2 種類のメッセージを新たに導入し，これらを同報することにより他のホストに直ちに知らせる．この場合，1 つのプロセスにつき 2 つのメッセージが新たに必要となるが，これはプロセスをリモートホストに割り当てる場合と同じであり，メッセージ数は十分に少ないと言える．

この方法では，もしネットワークが十分に高速であれば，各ホストは他のホストで実行されているプロセス数の増減をプロセスの種類にかかわらず直ちに知ることができ，定期的同報通信方式で見られたような同一ホストへの負荷の集中を回避できる．また，負荷分散に不可欠なメッセージを負荷の大きさの推定にも用いているため，入札アルゴリズムのように正確に負荷の大きさを推定するために通信量を増やす必要がなく，通信ネットワークに対する負荷も軽減できる．

4.3.2 通信傍受方式による動的負荷分散アルゴリズム

4.3.1節で述べた通信傍受による負荷情報収集手法を用いた動的負荷分散アルゴリズムを以下に示す．また，このアルゴリズムの概略を表すフローチャートを図 4.1に示す．

〔アルゴリズム A: 通信傍受方式による動的負荷分散〕

1°: 初期化

ホスト h_iは，起動時あるいは一旦稼働を停止した後に稼働を再開したしたときに，同報通信を用いて問い合わせる等の方法により全てのホストの負荷の大きさ $L_i(j)$ を初期化する．

2°: プロセスの実行

各ホスト h_iは，割り当てられたプロセスが存在すれば，それらをラウンドロビン (RR) スケジューリングに基づいて実行する．もし，実行中に次の何れかのイベントが起きれば，プロセスの実行を一時中断し，各イベントの処理を行う．

(a) ユーザから新しいプロセスが到着した場合には3°へ進む.

(b) 実行中のプロセスが終了した場合には4°へ進む.

(c) 割り当てメッセージを受信あるいは傍受した場合には5°へ進む.

(d) 終了メッセージを受信あるいは傍受した場合には6°へ進む.

(e) 同報通信を受信した場合には7°へ進む.

3°: 新しいプロセスの到着

新しいプロセスがホスト h_iに到着すると，h_iのPDが起動され，負荷の大きさ $L_i(j)$ が最も小さいホスト h_jを選択する．この場合，もし負荷の大きさが同じホストが2つ以上あれば，そのうちの1つを無作為に選ぶ．次に，プロセスをリモートホストで実行するときのコストを考慮して，予め定められた閾値θ(定め方については後述) に対して不等式

$$L_i(i) - L_i(j) > \theta \tag{4.1}$$

が成り立てば，PDはホスト h_jに割り当てメッセージを送り $L_i(j)$ を1つ増やす．そうでなければ，PDは新しいプロセスをローカルホスト h_iに割り当てて $L_i(i)$ を1つ増やし，更にローカル割り当てメッセージを同報する．これらの処理が終わると2°に戻ってプロセスの実行を再開する．

4°: プロセスの実行終了

ホスト h_iに割り当てられていたプロセスが終了すると，ホスト h_iはまず $L_i(i)$ を1つ減らす．次に，もしそれがリモートプロセスならば，ホスト h_iはそのプロセスのローカルホストに終了メッセージを送る．そうでなければ，ホスト h_iはユーザに対して終了通知等を送り，更にローカル終了メッセージを同報する．これらの処理が終わると2°に戻ってプロセスの実行を再開する．

5°: 割り当てメッセージの傍受 (受信)

ホスト h_iは，ホスト h_jに向けられた割り当てメッセージを傍受 (受信) すると，$L_i(j)$ を1つ増やす．また，ホスト h_iが割り当てメッセージを受信した場合 (すなわち $h_i = h_j$のとき) には，更に割り当てられた新しいプロセスを待ち行列に入れる．これらの処理が終わると2°に戻ってプロセスの実行を再開する．

6°: 終了メッセージの傍受 (受信)

ホスト h_iは，ホスト h_jからホスト h_kに向けられた終了メッセージを傍受 (受

信) すると，$L_i(j)$ を 1 つ減らす．また，ホスト h_i が割り当てメッセージを受信した場合 (すなわち $h_i = h_k$ のとき) には，更にユーザに対して終了通知等を送る．これらの処理が終わると 2° に戻ってプロセスの実行を再開する．

7°: ローカル割り当て・終了メッセージの受信

ホスト h_i は，ホスト h_j から送られたローカル割り当てメッセージを受信すると $L_i(j)$ を 1 つ増やす．また，ホスト h_j から送られたローカル終了メッセージを受信すると $L_i(j)$ を 1 つ減らす．これらの処理が終わると 2° に戻ってプロセスの実行を再開する． ■

なお，式 (4.1) において θ はローカルホストをどの程度リモートホストより優先するかを表すパラメータであり，メッセージの伝送遅延時間 (メッセージを送ろうとしてから実際に受け取られるまでの時間で待ち時間を含む) などを考慮して決定されるべきものである．ローカルプロセスとリモートプロセスの間で単位時間当りに受ける仕事量に差がないと仮定すると，ローカルホストに割り当てた場合の新しいプロセスの応答時間の期待値は $\overline{W}L_i(i)$ であるのに対し，リモートホストに割り当てた場合には $\overline{W}L_i(j) + 2\delta_N$ となる．ここで $\overline{W}$ はプロセスの要求仕事量の平均を表し，δ_N はメッセージの伝送遅延時間 (待ち時間を含む) を表す．従って，新しいプロセスをリモートホストに割り当てた方が有利になるのは

$$\overline{W}L_i(i) > \overline{W}L_i(j) + 2\delta_N \tag{4.2}$$

が成り立つときである．これより，θ の候補として，

$$\theta = 2\delta_N/\overline{W} \tag{4.3}$$

が考えられる．なお，この場合ネットワークが十分高速で $\theta < 1$ であるならば，式 (4.1) の左辺が整数であるため $\theta = 0$ としても実行ホストの選択には影響を与えない．

なお，この方式において 4 種類のメッセージ中にコマンド名などプロセスの種類を示す情報を含めると，各ホストは通信を傍受することにより他のホストで実行中の全てのプロセスの種類を知ることができる．従って，第 3 章で述べた式 (3.18) を用いて負荷の大きさを求めることも可能である．

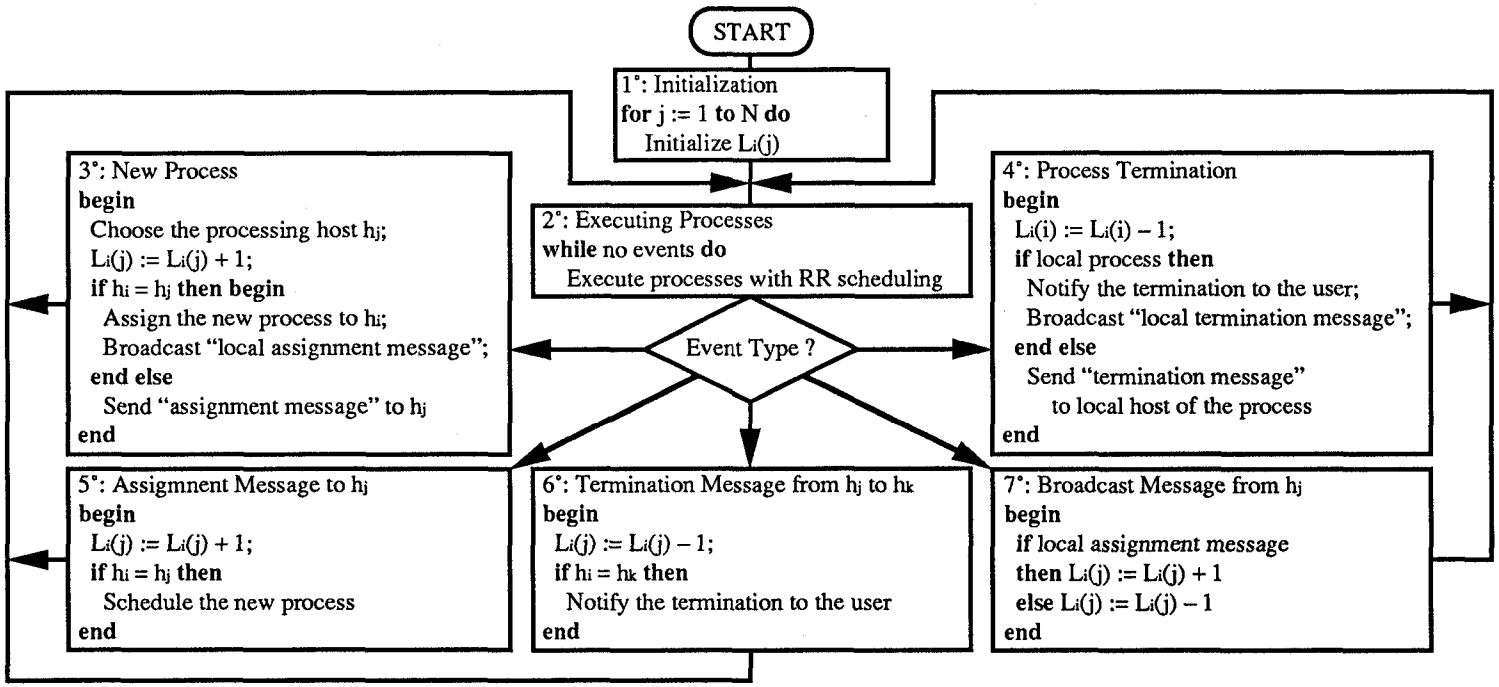


図 4.1: 通信傍受方式による動的負荷分散アルゴリズムの概略

4.3.3 他の負荷情報収集手法による動的負荷分散アルゴリズム

4.4節で述べるように，本方式 (アルゴリズム A) の性能評価において，負荷情報の収集方法として定期的同報通信方式 (同 B)，入札方式 (同 C) 及び負荷分散を行わない場合 (同 D) の 3 つの方式を比較の対象とした．以下，これらの方式を用いた動的負荷分散アルゴリズムについて簡単に示す．

(1) 定期的同報通信方式

この方式は，各ホストが予め定められた一定時間 t_o毎に負荷の大きさを知らせるメッセージ (負荷メッセージ) を同報する点がアルゴリズム A と異なる．なお，この方式では，4.2節のモデルにおいてホスト h_iは同報通信の間隔を測るための減算タイマ T_iをそれぞれ持っており，時間切れまでの残り時間 r_iを独立に設定できるものと仮定する．

〔アルゴリズム B: 定期的同報通信方式による動的負荷分散〕

1°: 初期化

システムが起動されたとき，各ホスト h_iは全てのホストの負荷の大きさ $L_i(j)$ を 0 に初期化する．また，タイマ T_iの残り時間 r_iを $t_o \times i/N$に設定する．ここで，Nはホストの台数を表す．なお，この設定はネットワークから見た負荷メッセージの到着間隔を一定にするためである．

2°: プロセスの実行

各ホスト h_iは，割り当てられたプロセスをアルゴリズム A と同様に実行する．もし，実行中に次の何れかのイベントが起きれば，プロセスの実行を一時中断し，各イベントの処理を行う．

(a) ユーザから新しいプロセスが到着した場合には 3°へ進む．

(b) 実行中のプロセスが終了した場合には 4°へ進む．

(c) タイマ T_iの残り時間 r_iが 0 になった場合には 5°へ進む．

(d) 負荷メッセージを受信した場合には 6°へ進む．

3°: 新しいプロセスの到着

アルゴリズム A と同じ方法により実行ホストを決定する．但し，もしリモートホスト h_jに割り当てる場合には，h_jに割り当てメッセージを送るが，$L_i(j)$ を更新しない．また，ローカルホスト h_iに割り当てる場合には，$L_i(i)$ を 1 つ増やすが，同報通信は行わない．

4°: プロセスの実行終了

ローカルプロセスが終了した場合には同報通信を行わない点を除いて，アルゴリズム A と同じである．

5°: 負荷メッセージの送信

タイマ T_iの残り時間 r_iが 0 になると，ホスト h_iは直ちに r_iを t_o に再設定し，自分自身の負荷の大きさ $L_i(i)$ を含む負荷メッセージを同報する．これらの処理が終わると 2°に戻ってプロセスの実行を再開する．

6°: 負荷メッセージの受信

ホスト h_iは，ホスト h_jから送られた負荷メッセージを受信すると，その中に含まれるホスト h_jの負荷の大きさ $L_j(j)$ を $L_i(j)$ に代入する．その後2°に戻ってプロセスの実行を再開する． ■

(2) 入札方式

このアルゴリズムは，新しいプロセスが到着する度に同報通信を用いて各ホストに現在の負荷の大きさを問い合わせる点がアルゴリズム A と異なる．すなわち，新しいプロセスが到着すると，ホストはその旨を知らせるメッセージ (到着メッセージ) を同報し，これに対する各ホストからの負荷の大きさを知らせるメッセージ (入札メッセージ) をすべて受信して実行ホストを決定する．なお，このアルゴリズムでは特に初期化は必要としない．

〔アルゴリズム C: 入札方式による動的負荷分散〕

1°: プロセスの実行

各ホスト h_iは，割り当てられたプロセスをアルゴリズム A と同様に実行する．もし，実行中に次の何れかのイベントが起きれば，プロセスの実行を一時中断し，各イベントの処理を行う．

(a) ユーザから新しいプロセスが到着した場合には2°へ進む.

(b) 到着メッセージを受信した場合には3°へ進む.

(c) 入札メッセージを受信した場合には4°へ進む.

(d) 実行中のプロセスが終了した場合には5°へ進む.

2°: 新しいプロセスの到着

新しいプロセスがホスト h_iに到着すると，h_iのPDは到着ホストを同報し，その後2°に戻ってプロセスの実行を再開する．この段階では新しいプロセスはどのホストにも割り当てられていない状態である.

3°: 到着メッセージの受信

ホスト h_iがホスト h_jからの到着メッセージを受信すると，現在の自分自身の負荷の大きさ $L_i(i)$ をホスト h_jに送信する．その後2°に戻ってプロセスの実行を再開する.

4°: 入札メッセージの受信

ホスト h_iは，ホスト h_jから送られた入札メッセージを受信すると，その中に含まれるホスト h_jの負荷の大きさ $L_j(j)$ を $L_i(j)$ に代入する．ここで，もしまだ受信していない入札メッセージがあれば，2°に戻ってプロセスの実行を再開する．そうでなければ，アルゴリズムBと同じ方法によりリモートホストへの割り当てメッセージの送信あるいはローカルホストへのプロセスの割り当てを行い，その後2°に戻ってプロセスの実行を再開する.

4°: プロセスの実行終了

アルゴリズムBと同じである. ■

(3) 負荷分散無し

これは新しいプロセスをすべてローカルホストに割り当てるものである．従って，全く通信を必要としない．但し，新しいプロセスをスケジューリングするための処理時間は他のアルゴリズムと同様に考慮する.

なお，このアルゴリズム(アルゴリズムD)の詳細は自明なので省略する.

4.4 評価と考察

4.4.1 性能評価の概要

4.3節で示した通信傍受方式の有効性を評価するため，10 台のホストを接続した分散システムについてシミュレーションを行った．また，4.3.3で示したアルゴリズム B, C, D についても同様のシミュレーションを行ない，性能を比較した．

各ホストの処理モデルは 4.2節で述べたものを用いたが，以下の実験では更に次のように仮定した．

1. 通信ネットワークは FCFS(first come first served) 単一サーバでモデル化され，そのメッセージ伝送時間 (待ち時間を含まない) は平均t_N秒の指数分布に従うものとする．
2. 各ホストへのプロセスの到着間隔はパラメータλの指数分布に従うものとし，その平均$1/\lambda$は全てのホストにおいて等しいものとする．
3. 各ホストに到着するプロセスの要求仕事量は指数分布に従うものとし，その平均$\overline{W}$は全てのホストにおいて等しいものとする．
4. PD が実行ホストを決定するために要する処理時間δ_D及びメッセージを受信あるいは傍受した場合の処理時間δ_Mは，アルゴリズムの種類，メッセージの種類，各ホストの状態などによらずそれぞれ一定であるとする．

実験では，プロセスの平均到着率λと通信ネットワークの平均伝送時間t_Nをそれぞれ変えた場合について各アルゴリズムにおけるプロセスの平均応答時間を測定した．但し，アルゴリズム B については，同報通信の間隔t_oを 0.5 秒，1 秒，10 秒の 3 種類に変え，それぞれについて測定を行った．それぞれのシミュレーションでは 50 回の独立試行を行ない，各試行ではプロセスの応答時間を開始 100 秒後から 300 秒後まで集計した．全ての実験に共通のパラメータを表 4.1に示す．

表 4.1: 各シミュレーション実験に共通のパラメータ

パラメータ	値
PD における処理時間 (δ_D)	0.01秒
メッセージの処理時間 (δ_M)	0.01秒
RR の単位時間量 (t_q)	0.01秒
プロセスの平均要求仕事量 ($\overline{W}$)	1秒
アルゴリズム A~C における閾値 (θ)	0

4.4.2 平均到着率を変えた場合

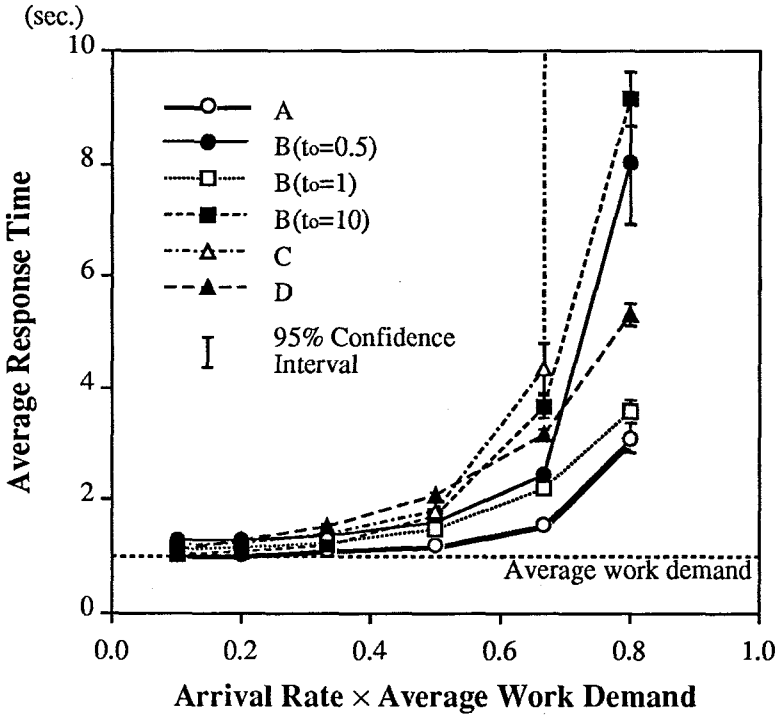
まず，通信ネットワークの平均伝送時間 t_N を 0.01 秒とし，プロセスの平均到着率λを変化させた場合について各アルゴリズムの性能を測定した．その結果を図 4.2 に示す．なお，$\lambda\overline{W} = 4/5$ におけるアルゴリズム C の平均応答時間は，ホストの処理能力が飽和してしまったため測定できなかった．

図 4.2を見ると，全ての場合においてアルゴリズム A が他のアルゴリズムより優れていることがわかる．また，平均到着率λの増加に対して，アルゴリズム A は他のアルゴリズムより平均応答時間の増加が穏やかであることがわかる．

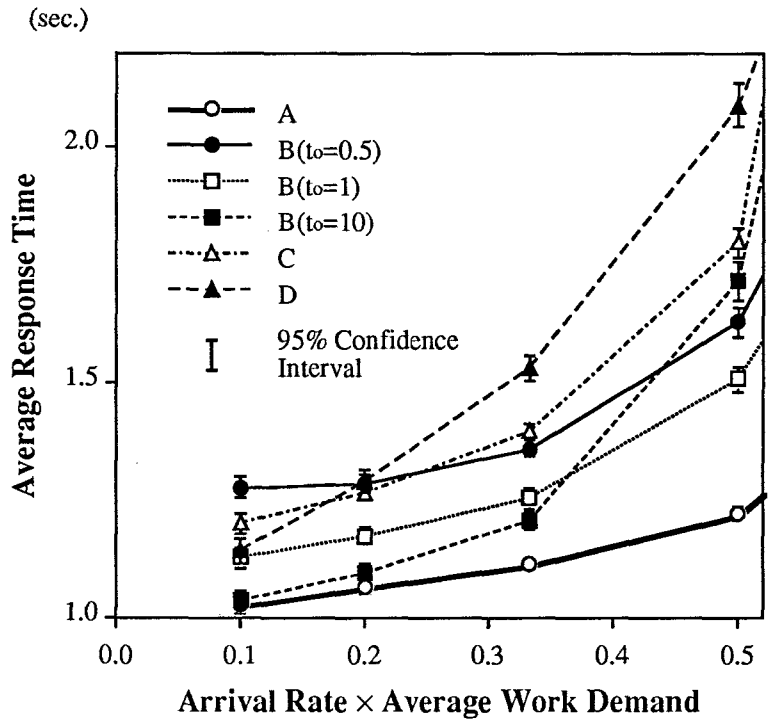
図 4.2の現象の理由は次のように説明できる．

アルゴリズム A とアルゴリズム B を比較すると，平均到着率λが小さい ($\lambda\overline{W} \leq 1/3$) とき，プロセスの数が少ないため各ホストの状態はあまり変化せず，負荷情報を頻繁に収集する必要がない．従って，同報通信の間隔 t_oが小さいと通信のオーバヘッドの面で不利であり，アルゴリズム A や $t_o = 10$ のアルゴリズム B が良好な性能を示す．

平均到着率λが大きくなる ($\lambda\overline{W} \geq 1/2$) に従って，$t_o = 10$ のアルゴリズム B の性能が劣化している．これは負荷情報の収集が十分行われず，特定のホストに負荷が集中するためと思われる．一方，$t_o = 1$ のアルゴリズム B は十分情報を収集しているため，適切に負荷分散が行われていると思われる．アルゴリズム A はアルゴリズム B より常に良い性能を示しているが，これはより少ないメッセージでより正確な負荷情報を得ているためと思われる．なお，($\lambda\overline{W} = 4/5$) のときにはアルゴリズム A と $t_o = 1$ のアルゴリズム B の性能が比較的接近しているが，これはアルゴリズム



(a) 全体図



(b) 拡大図

図 4.2: 平均到着率の変化に対する平均応答時間

A のメッセージ数が増え，通信のオーバヘッドに関する両者の差があまりなくなったためと思われる．

平均到着率λの大小にかかわらず，アルゴリズム B において $t_o = 0.5$ のものは $t_o = 1$ のものに比べて性能が悪い．これは，$t_o = 0.5$ では負荷情報の正確さよりも通信のオーバヘッドが大きな影響を与えるためである．アルゴリズム C の性能が悪いのも同様の理由で説明できる．

以上の結果並びに考察から，アルゴリズム A はアルゴリズム B, C と比べて性能が優れており，また様々な負荷に対してもいずれも比較的良い性能を示す．現実のシステムでは負荷の変動が頻繁に起こり得るため，アルゴリズム A のこれらの性質は好ましいと思われる．

4.4.3 ネットワークの平均伝送時間を変えた場合

次に，プロセスの平均到着率λを一定 ($\lambda \overline{W} = 1/2$) にし，通信ネットワークの平均伝送時間 t_N を変化させた場合について各アルゴリズムの性能を測定した．その結果を図4.3に示す．なお，$t_N = 0.1$ の場合のシミュレーションにおいて，$t_o = 0.5$ 並びに $t_o = 1$ のアルゴリズム B とアルゴリズム C では通信ネットワークが飽和したため，これらのアルゴリズムの平均応答時間を測定できなかった．

図4.3を見ると，アルゴリズム C を除いて $t_N = 0.001$ のときと $t_N = 0.01$ のときの違いは殆ど見られないことがわかる．これはアルゴリズム C 以外のアルゴリズムは $t_N = 0.01$ でも通信に要するオーバヘッドが十分小さいことを表している．逆にアルゴリズム C は通信に要するオーバーヘッドが大きいため通信ネットワークの伝送速度の影響を受けやすいが，十分ネットワークが高速であれば正確な負荷情報を素早く得られるためアルゴリズム B よりも性能がよくなると考えられる．

一方 $t_N = 0.1$ のときには，平均伝送時間が大きすぎるため，アルゴリズム A~C の殆どはアルゴリズム D より平均応答時間が悪くなり，負荷分散を行う意味を失っている．しかし，アルゴリズム A はこの状況においても他の多くのアルゴリズムとは異なってネットワークが飽和しておらず，ネットワークの伝送速度の影響を比較的受けにくいことがわかる．アルゴリズム A のこの性質は，1つのネットワークに多くのホストが接続されネットワークが混雑している場合でも比較的有効に負荷分散を行えることを意味し，現実のシステムには好ましいと思われる．

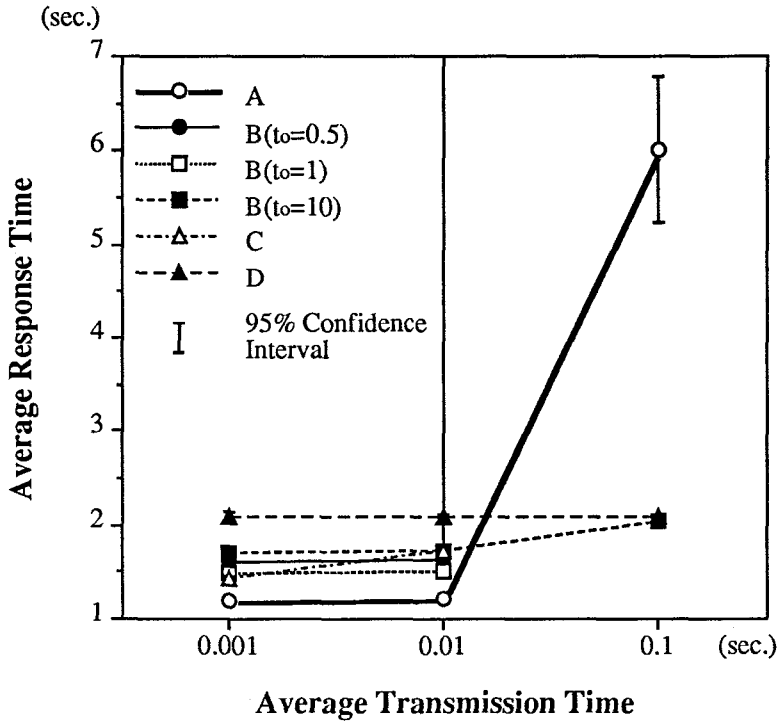


図 4.3: ネットワークの平均伝送時間の変化に対する平均応答時間

4.5 まとめ

本章では，典型的な分散システムにおいて通信量を抑えながら同時に同一のコンピュータに負荷を集中させるないで各計算機の負荷情報を遅滞なく収集する手法を提案した．

シミュレーション実験の結果，本手法は定期的同報通信方式や入札方式より優れており，またプロセスの到着率やネットワークの伝送時間の変動に対して比較的強いことがわかった．現実のシステムではこれらの変動が頻繁に起こり得るため，この性質は実際に本手法を適用する際に重要であると思われる．

しかし，本手法ではネットワークのメッセージ平均伝送時間がある程度以上大きくなると，負荷分散を行わない場合より性能が悪くなり，負荷分散を行う意味を失ってしまう点が問題となる．

今後の課題としては，このような状況でも有効に負荷分散を行う負荷情報の収集方法の開発が挙げられる．また，本手法を現実のシステム上に実現し，性能評価を行うことも今後の課題の１つである．

本手法はLANが持つ同報通信機能を利用しているため，同報通信機能を持たない広域分散システムに対しては現在のところ適用できない．しかし，今後は光ファイバの普及や広帯域ISDN(Broadband Integrated Services Digital Networks, B-ISDN)[41]の開発・実用化により，広域ネットワークにおいても同報通信機能がサポートされ，本手法を広域分散システムに対しても適用可能となることが期待される．

第 5 章

動的負荷分散機能の実装

本章においては，耐故障性やネットワーク透過性を考慮した動的負荷分散機能のUNIX への実装方法について論じる．本研究では，故障したホストに対する耐故障性を実現する方法として，コマンド入力時に同報通信を用いてホストが稼動しているかどうかの問い合わせを行い，ローカルホストより早く応答したホスト (ローカルホスト自身を含む) をコマンド割り当ての対象とする方法を提案する．また，ネットワーク透過性を実現する方法として，REX を用いたコマンド遠隔実行機能のコマンドインタープリタ (シェル) への実装を提案する．

5.1 実装における問題点

2.1.2節でも述べたように，標準的な UNIX では rsh, rcmd, rexec などの遠隔実行機能が用意されている．しかし，これらの機能は全て明示的にコマンドを実行するホストを指定する必要があるため不十分であり，動的負荷分散機能を実装するためには負荷の軽いホストを自動的に選択し，そのホスト上で自動的にコマンドを実行する機能を組み込む必要がある．

また，分散システムに動的負荷分散機能を実装する場合，コマンドがどのホストで実行されるかをユーザが事前に知ることができないため，ネットワーク透過性が重要となる．すなわち，コマンドがどのホストで実行される場合でも同じ結果を与えることを保証し，ユーザがネットワークの存在を意識する必要のないようにしなければならない．また，分散システム中の一部のホストで障害が発生した場合でも，例えば障害が発生したホストに負荷を分散しないなど，これに対する耐故障性 (fault

tolerance) を有する必要がある.

負荷分散を行う場合にネットワーク透過性や耐故障性を持たせることは分散 OS の主要な目的の 1 つであり, 例えば, LOCUS[1] などではこれらの性質が実現されている. しかし, 分散 OS は現在の時点では研究・開発段階であまり普及しておらず, あるいは利用するにはカーネルやアプリケーションプログラムの修正を必要とするため, 既存のシステムのユーザはこれらを利用することは困難である.

一方, 既存の UNIX システムのカーネルやアプリケーションプログラムを変更しないで動的負荷分散機能を実装した例として, Butler[32] や LB[33] などが挙げられる.

このうち, Butler では各計算機が使用中であるか否かをいくつかのホスト登録サーバで管理し, 新しいコマンドを使用中でないホストのいずれかで遠隔実行する方法を用いている. この方法では, ユーザがコマンドを入力するとまずホスト登録サーバにどの計算機が使用中でないかを問い合わせ, 次にホスト登録サーバが指定したホストが使用中でないことを確認し, もし実際にそのホストが使用中でなければ実行環境を設定してコマンドの遠隔実行を行う. その際, 耐故障性を持たせるため, ホスト登録サーバを分散システム上に複数配置するなどの工夫が施され, また実行環境を保存し AFS(Andrew File System)[10] を用いてファイルの位置透過性を持たせるなど, ネットワーク透過性についても配慮されている. しかし, この方法では全てのホストが使用中の時にはそれ以上の負荷分散を行うことができず, また故障しているホストが存在する場合の耐故障性が必ずしも十分ではない. 例えば, ホスト登録サーバの指定した計算機が故障している場合, そのホストが使用中かどうかの確認を行う場合にその応答を時間切れになるまで待ち続ける危険性がある.

また, LB は入札方式により負荷を問い合わせ, 新しいコマンドを負荷の最も軽いホストで遠隔実行する方法を用いている. この方法では, ユーザがコマンドを入力すると登録されている各ホストの統計情報サーバ rstatd に過去 1 分間の平均実行待ちプロセス数 (負荷平均, load average) を個別に問い合わせ, その値をホストの処理速度で割った商を負荷の大きさと定義し, 全ての負荷の大きさを比較して最も負荷の小さなホストで rsh を用いてそのコマンドを遠隔実行する方法を用いている. しかし, この方法では全てのホストからの応答を待つ必要があるため, コマンドの割り当てに要する時間が長くなり, また故障したホストが存在する場合にそのホストからの応答を時間切れになるまでの間 (実測値では 1 分間) 待ち続ける等, 耐故障

性に問題がある．更に，LB ではコマンドの遠隔実行に rsh を用いているため，ファイルの位置透過性やコマンド実行時の環境変数の保存などのネットワーク透過性が確保されていない．

更に，両者とも負荷分散を行うには，Butler では “rem *command* [*args* ...]”, LB では “lb *command* [*args* ...]” のように負荷分散を行うことを陽に指定するため，ユーザがコマンド入力時に毎回負荷分散をするかどうかを判断をする必要があり，ネットワーク透過性の面から見て好ましくない．

そこで，以下では UNIX 上でカーネルや既存のアプリケーションプログラムを変更しないでネットワーク透過性や耐故障性を持たせることを目的とした負荷分散機能の実装方法について述べる．

5.2 問題点に対する対策

5.2.1 耐故障性に対する対策

耐故障性を持たせるために故障したホストへの割り当てを避けるには，入札方式のようにまずホストに問い合わせを行い，これに対する応答を確認してから遠隔実行を行う方法が有効である．しかしこの場合，どの時点で問い合わせに対する応答の受付を打ち切るかが問題となる．すなわち，応答受付時間が長過ぎると，故障したホストが存在した場合に無駄となる時間が長くなり，逆に応答受付時間が短過ぎるとホストが時間内に応答できずに故障と判断される可能性が高くなる．また，問い合わせに対する応答時間は一般にシステムの状態によって変わるため，この方法では応答受付時間もシステムの状態に応じて調整する必要がある．

この問題点に対して，我々は同報通信を用いた問い合わせに対して最も早く応答したホストを選択する方法を提案した [44]．この方法では少なくともローカルホストは応答可能であるため，応答を待ち続ける危険性を回避することができる．シミュレーションの結果では，この方法の一応の有効性が確認されたが，一方この方法では処理速度が大きいホストは負荷が大きい場合でも問い合わせに対する応答時間が短いため負荷が集中し過ぎ，またローカルホストは問い合わせとそれに対する応答を並行して行う必要があるため問い合わせに対する応答時間が比較的長く，殆ど選択されない傾向が見られた．

そこで，本研究ではローカルホストの同報通信に対する応答時間が比較的長いことから，ローカルホストが応答するまでの間を応答受付時間とし，次に示す手順によりコマンドを割り当てるホストを選択する．

〔ホスト選択手順〕

1°: 最も負荷の軽いホストの調査

コマンドを割り当てる直前に最も負荷の軽いホストがローカルホストであるかどうかを調べる．もしそうであればローカルホストを選択し，この手順を終了する．

2°: 同報通信による問い合わせ

負荷の最も軽いホストがローカルホストでない場合，各ホストに対して同報通信を行い，各ホストが故障しているかどうかを問い合わせる．

3°: 応答の受信

同報通信に対する各ホストからの応答を待ち，応答を返したホスト名を記録する．この操作をローカルホストあるいは最も負荷の軽いホストが応答するまで繰り返す．

4°: ホストの選択

3°において最も負荷の軽いホストが応答を返した場合には，無条件にそのホストを選択する．そうでなければ応答を返した各ホストの負荷を比較し，このうち最も負荷の軽いホストを選択する． ■

この手順ではローカルホストが応答するまでの間を応答受付時間としているため，システムの状態に応じて応答受付時間が自動的に調整され，故障した計算機が存在する場合でもシステムの状態に関わらず無駄な待ち時間を小さく抑えることが可能となる．また，この場合ローカルホストより応答が遅いホストを故障しているものと見なして無視するが，このようなホストはローカルホストの応答が比較的遅いことからたとえ故障していないとしても負荷が大きいと推測されるため，これを無視しても負荷分散の性能に殆ど影響を与えないと思われる．

なお，この手順の2°において，同報通信で単に各ホストが故障であるかどうかを問い合わせるだけでなく，例えばLBと同様に統計情報サーバrstatdなどに問い合

わせることにより同時に種々の負荷情報を入手することも可能である．実際，5.3節で述べるように，本研究で試作したシェルでは簡単化のために rstatd に問い合わせを行って負荷の大きさを推定している．

また，2°において，コマンド名，計算機の機種名，あるいは応答を返してほしいホスト群などの種々の条件をメッセージに含めて同報し，これを受け取った各ホストが，指定されたコマンドを実行可能であるとき，ローカルホストと同一機種であるとき，あるいは指定されたホスト群に属するときなど，指定された条件に合致するときだけ応答を返すようにすれば，無駄な応答を避けネットワークの通信量を減らすことが可能である．特に分散システムにマルチキャスト機能が提供されている場合には，特定の条件を満たすホストだけに問い合わせを行うことにより同様の効果を得ることができる．

5.2.2 ネットワーク透過性に対する対策

(1) ファイルの位置透過性と実行環境の保存

UNIX においてネットワーク透過性を保ちながらコマンドを遠隔実行するためには，2.1.2節で述べたように

- ファイルの位置透過性
- カレントディレクトリの保存
- 環境変数の保存

などについて考慮する必要がある．

本研究では，ネットワーク透過性と可搬性を考慮して，ファイルシステムとして広く普及している NFS(Network File System)[35] を用い，コマンドの遠隔実行に多くの UNIX で RPC サービスの 1 つとして提供されている REX を用いている．

このうち，2.1.2節で述べたように，REX では必要であればカレントディレクトリが含まれているファイルシステムを自動的に NFS によりマウントし，カレントディレクトリと環境変数を設定した後に指定されたコマンドを実行する．この機能により，REX はコマンドの実行環境を保存し，少なくとも REX 自身がマウントし

たファイルシステム内のファイルを参照する相対パス名に対する位置透過性を満足する．しかし，ファイルの位置透過性に関しては，実際のシステムではファイルを絶対パス名で指定することが比較的多いにも関わらず，REX では絶対パス名に対する位置透過性が保証されていない．

そこで本研究では絶対パス名に対してもある程度の位置透過性を得るためにファイルシステム自動マウントプログラム automount[36] を導入している．automount は NFS version 4.0 以降で利用することができるファイルシステム自動マウントプログラムで，仮想ディレクトリ名とそれに対する参照先ディレクトリ名の対を指定することにより，仮想ディレクトリがアクセスされると参照先ディレクトリを自動的に NFS によりマウントし，実際にはこれがアクセスされるように機能する．

図 5.1 に automount の設定例を示す．この例では，ファイル auto.master は「/user 直下の仮想ディレクトリ名と実際に参照されるディレクトリ名との対応はファイル auto.user で指定されており，自動マウントするときのオプションは “-rw,intr,hard” である」ことを示している．また，ファイル auto.user は「仮想ディレクトリ user1, user2 が参照されると，実際には計算機 host1 のディレクトリ/home/user1, 計算機 host2 のディレクトリ/home/user2 がそれぞれ参照される」ことを示している．従って，仮想ディレクトリ/user/user1, /user/user2 が参照されると，それぞれ計算機 host1 のディレクトリ/home/user1, 計算機 host2 のディレクトリ/home/user2 がオプション “-rw,intr,hard” で自動的にマウントされ，これらのディレクトリが実際には参照される．この機能を利用すれば，例えば各ユーザのホームディレクトリ等を仮想ディレクトリ名を用いて参照することにより，これらについては絶対パス名に対する位置透過性を持たせることが可能となる．

従って，NFS，REX 及び automount を併用することにより，実用上十分と思われるネットワーク透過性を持たせることが可能となる．なお，これらの併用によっても，他のファイルシステムを参照する相対パス名や，/tmp のように内容が計算機によって異なるディレクトリあるいは/dev のように特殊ファイルを含むファイルを参照するパス名に対しては一般には位置透過性は満足されない．しかし，これらのパス名を用いる頻度はかなり小さいと思われるため，本研究における実装では 5.3 節に述べるように，この場合だけユーザが一時的に負荷分散を行わないように指定する方法を用いている．

```
# dir   map file    mount options
/user   auto.user   -rw,intr,hard
```

(a) auto.master

```
# dir           host:dir
user1           host1:/home/user1
user2           host2:/home/user2
```

(b) auto.user

図 5.1: automount の設定例

(2) 負荷分散機能の透過性

UNIX では，現在のプロセスの状態を表示するコマンド ps のように本来実行結果が計算機に依存するコマンドや，日時を表示するコマンド date のように実行時間が短いコマンドなど，負荷分散すべきでないコマンドも存在する．従って，コマンドを実行する場合，まずそのコマンドを負荷分散するかどうかを判定する必要がある．これに対して，Butler や LB では前記のように rem コマンドや lb コマンドのように負荷分散を行うコマンド用いてユーザが陽に指定する方法が用いられている．

分散システムにおいてネットワーク透過性が包含する概念に，移送透過性 (migration transparency) が挙げられる [40]．これはプロセスやファイルなどのオブジェクトの移送がユーザに意識されず移送の過程においてユーザあるいはアプリケーションの処理に影響を与えないことを意味する．従って，ネットワーク透過性を考慮して負荷分散を行うには，Butler や LB のようにユーザが陽に指定する方法は好ましくない．

そこで，本研究では負荷分散機能をコマンドインタープリタ (シェル) に組み込み，負荷分散の対象となるコマンドだけをシェルに登録し，コマンド入力時に登録の有無を調べて登録されているコマンドのみ負荷分散を行う方法を提案する．これによりユーザや管理者がシェルの初期設定ファイルなどで負荷分散対象コマンドを登録すれば良く，コマンド入力時にユーザがその都度判断する必要なく自動的に負荷分散を行うことが可能となる．

5.3 負荷分散機能を持つシェルの試作

本研究では，5.2節の議論に基づき，コマンドレベルでの動的負荷分散機能をシェルの1つであるbashへ実装した．これはbashはソースプログラムが公開されており，UNIXのライセンス契約と無関係に利用することができるためである．以下では，負荷分散機能のbashへの実装方法について述べる．

5.3.1 負荷の大きさの推定

試作したシェルでは，故障した計算機に対する耐故障性に重点を置いたため，負荷の大きさの推定方法を簡単化し，LBと同様に統計情報サーバrstatdにより求められる過去1分間の負荷平均をその計算機の処理速度で割ったものを負荷の尺度として用いている．従って，本シェルのホスト選択手順は以下に示すとおりになる．

〔アルゴリズムE: 試作したシェルのホスト選択手順〕

1°: 同報通信による問い合わせ

シェルは各ホストのrstatdに対して同報通信により各ホストの過去1分間の負荷平均を問い合わせ，同時に各ホストが故障していないかどうかを確認する．

2°: 応答の受信

シェルは同報通信に対する各ホストからの応答を待ち，応答が返されると応答したホストの負荷を次式より求める．

$$\text{負荷} = (\text{負荷平均} + \text{オーバヘッド値})/\text{処理速度} \tag{5.1}$$

ここでオーバヘッド値はコマンドの遠隔実行時に余分にかかる負荷の大きさを表す値であり，処理速度と共にホスト毎に指定されているものとする．

3°: ホストの選択

次にシェルは応答したホストを調べ，それがローカルホストであれば，シェルは応答の受付を中止し，これまでに応答したホストの内，最も負荷の軽いものを選択する．応答したホストがリモートホストであれば，再び他のホストからの応答を待つ． ■

5.3.2 コマンドの登録機能

試作したシェルでは，5.2.2節で述べたように予め負荷分散の対象となるコマンドだけをシェルに登録し，コマンド入力時に登録の有無を調べて登録されているコマンドのみ負荷分散を行う機能を用いている．この機能は，シェル内部で使われているコマンドパス名検索用のハッシュ表に負荷分散の対象であることを示すフラグを付け加えることにより実現しており，与えられたコマンドを負荷分散すべきかどうかを高速に判定することができる．なお，このフラグを操作するために新しい内部コマンド balance をシェルに追加した．このコマンドの使い方を表5.1に示す．

表 5.1: balance コマンド

コマンド	意　味
`balance`	登録コマンドの一覧を表示
`balance` *command ...*	*command, ...* を登録
`balance -n` *command ...*	*command, ...* の登録を削除
`balance -r`	全コマンドの登録を削除

また，5.2.2で述べたように，本実装ではREXとNFS, automountとの併用により，実用上十分と思われるネットワーク透過性を持たせている．しかし，/tmpや/dev等を参照するパス名など一部のパス名に対してはファイルの位置透過性を満足させることができないため，このようなパス名を参照するコマンドを実行する場合には，コマンドが負荷分散の対象として登録されていても負荷分散の対象にするべきでない．そこで本実装では，このようなパス名をアクセスする場合を考慮して，新しい内部コマンド off を追加し，このコマンドを前置すると負荷分散の対象となっているコマンドでも一時的に負荷分散を行わないように指定することができるようにした．また，コマンドをパス名で指定した場合にも，ハッシュ表を検索しないため負荷分散は行われないため同様の効果が得られる．

5.3.3 シェルへの組み込み

負荷分散機能の bash への組み込みに際しては，コマンドの解析部分には原則的には手を触れず，新しいプログラムの実行を行う execve システムコールの部分を修

正することにより実装している．その他の修正部分は，前記の内部コマンド balance 及び off の追加と起動時における初期化ファイル (~/.balance) の処理の２点である．このうち，後者に関しては初期化ファイル中に図5.2のように各計算機に対する処理速度とオーバヘッド値が書かれており，式(5.1) を用いて負荷を計算するときに用いられる．

```
#  MACHINE      SFACTOR OVERHEAD
nnctgw          4       0.06
artemis         3       0.08
poseidon        2       0.12
```

図 5.2: 初期化ファイル ~/.balance の設定例

execve システムコールの代わりに実行される具体的な負荷分散手順を以下に示す．

〔負荷分散手順〕

1°: 負荷分散の有無の判定

シェルは，入力されたコマンドが負荷分散の対象として登録されているかどうかを調べ，もし登録されていれば2°へ進む．そうでなければ3° へ進む．

2°: 最も負荷の軽いホストの選択

シェルは5.3.1節のホスト選択手順により最も負荷の軽い計算機を選択する．もし，最も負荷の軽い計算機がローカルホストであれば3° へ進み，そうでなければ4°へ進む．

3°: ローカルホストでのコマンド実行

入力されたコマンドが負荷分散の対象として登録されていない場合，あるいは最も負荷の軽い計算機がローカルホストである場合には，修正前の通り execve システムコールを用いてローカルホストで実行する．コマンドの実行終了後，シェルは次のコマンド入力を待つ．

4°: リモートホストでのコマンド実行

シェルは最も負荷の軽い計算機上の rexd に対して入力されたコマンドの遠隔実行を依頼し，5°へ進む．

なお，このとき利用者認証などによるエラーが発生した場合，計算機選択手順で求めた負荷が次に軽い計算機を選択し，その計算機がローカルホストであるかリモートホストであるかにより 3°あるいは 4°を行う．

5°: 入出力の中継

シェルはコマンドの遠隔実行が終了するまで標準入出力や標準エラー出力をローカルホストとリモートホストの間で中継する．コマンドの実行終了後，シェルは次のコマンド入力を待つ． ■

なお，この実装法はネットワーク透過性が満足されている状況では，内部で複数のプロセスを起動するコマンドにおいてプロセス単位で負荷分散を行う場合でも適用可能である．

5.4 評価と考察

本節では本研究で試作したシェルにおいて，5.2節で述べた実装の問題点に対する対策が有効であることを確認する．特に 5.2.1 節で提案した耐故障性に対する対策が負荷分散の性能に殆ど影響を与えていないことを示す．

5.4.1 動作例

まず，動的負荷分散機能を組み込んだシェルの動作例を図 5.3 に示す．この例では負荷分散の様子がわかるように，敢えてホストに依存するコマンドである hostname(ホスト名を表示するコマンド) を実行している．この図において，“#” 以降は説明のためのコメントであり，実際の入出力とは無関係である．また，“bash$” は bash が出力するプロンプトであり，これに続く文字列が端末から入力されている．行頭がプロンプトでない行は全てコマンドの実行結果である．

1 行目では内部コマンド balance を実行しているが出力がなく，負荷分散の対象として登録されているコマンドが何もないことを示している．従って，2 行目でコマンド hostname を実行すると，3 行目にローカルホスト名である artemis が出力される．4 行目では負荷分散の対象コマンドとして hostname を登録し，5, 6 行目では登録内容を確認している．次の 7 ～ 12 行目ではコマンド hostname を 3 回連続して

実行し，1, 2 回目はリモートホストの 1 つである nnctgw で実行され，3 回目は別のリモートホストである athena で実行されている．これによりコマンド hostname は自動的に遠隔実行されていることがわかる．13, 14 行目は内部コマンド off を用いることにより，強制的にローカルホストで hostname を実行している様子を表す．また，15, 16 行目はコマンド off の効力が一時的なもので，コマンド hostname が再び自動的に遠隔実行されていることを示している．

また，図では示されていないが，一部のリモートホストが停止している状態で負荷分散の対象として登録されているコマンドを入力した結果，LB の場合とは異なり直ちにコマンドが実行され，試作したシェルが耐故障性を有することを確認した．

また，パス名で指定したディレクトリ中のファイル一覧を出力するコマンド ls，カレントディレクトリを表示するコマンド pwd，環境変数とその値の一覧を表示するコマンド printenv などのいくつかのコマンドについて，ローカルホストで実行した場合と遠隔実行した場合を比較した結果，絶対パス名や同一ファイルシステム内を参照する相対パス名に対するファイルの位置透過性，カレントディレクトリの保存，環境変数の保存，コマンドの実行環境が保存されていることを確認した．

5.4.2 負荷分散機能の性能評価

次に，シェルに実装した負荷分散機能の性能評価を行うため，最も負荷の軽いホストを選択する部分のみを 2, 3 のアルゴリズムに変更したものについて，実際のシステムにおいてシミュレーション実験を行った．実験に使用したシステムは Sun-3/260(4 MIPS)1 台，Sun-3/160(2 MIPS)1 台，Sun-3/60(3 MIPS)3 台の計 5 台であり，OS はすべて SunOS-4.0.3 である．実験で用いたシステムの構成図を図 5.4に示す．なお，実験に必要なファイルは全て各ホスト中に格納されているか，あるいは予め NFS を用いてマウントされており，REX のファイルシステム自動マウント機能は用いないようにした．

実験では，5.3.3節の負荷分散手順中の 2°のホスト選択手順の部分を，本章で提案した方法 (アルゴリズム E) の他に，LB のように rstatd の全ての応答を比較する方法 (アルゴリズム F)，文献 [44] で提案した問い合わせに対する応答時間を用いる方法 (アルゴリズム G)，並びに負荷分散を行わない方法 (アルゴリズム H) を用いたものについてもシミュレーションを行った．アルゴリズム F, G の概要を次に示す．

```
 1: bash$ balance           # no registered command
 2: bash$ hostname          # show the local hostname
 3: artemis
 4: bash$ balance hostname # register "hostname"
 5: bash$ balance           # show registered commands
 6: /usr/bin/hostname
 7: bash$ hostname          # load balancing (1st)
 8: nnctgw                  # "nnctgw" is selected
 9: bash$ hostname          # load balancing (2nd)
10: nnctgw                  # "nnctgw" is selected
11: bash$ hostname          # load balancing (3rd)
12: athena                  # "athena" is selected
13: bash$ off hostname      # force local execution
14: artemis
15: bash$ hostname          # load balancing again
16: athena                  # "athena" is selected
```

図 5.3: 動的負荷分散機能の動作例

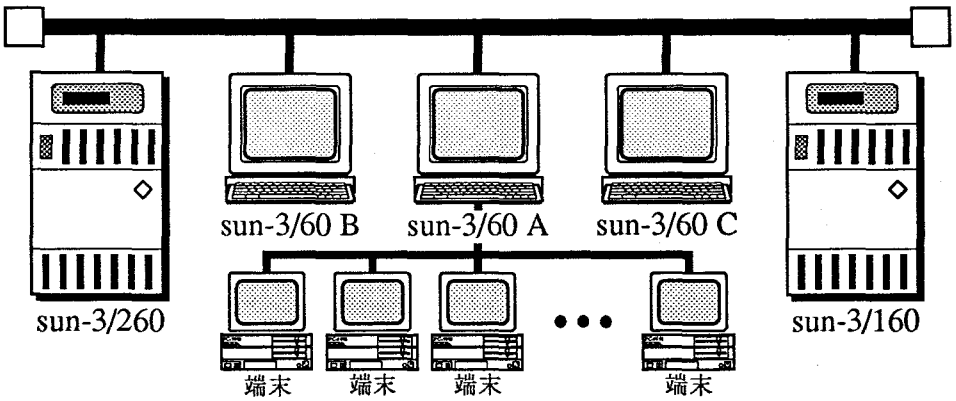


図 5.4: 実験システムの構成

なお，アルゴリズム H については自明であるので省略する．

〔アルゴリズム F: LB のホスト選択手順〕

1°: 1対1通信による問い合わせ

シェルは初期化ファイル ~/.balance に登録されているホストの1つに対して，その rstatd に過去1分間の負荷平均を RPC(Remote Procedure Call) を用いて1対1通信で問い合わせる．

2°: 応答の受信

シェルは問い合わせに対する応答を最大で1分間待ち，応答が返されるとアルゴリズム E と同様にそのホストの負荷を式 (5.1) より求める．

3°: ホストの選択

シェルは初期化ファイル ~/.balance に登録されている各ホストについて1°と2°を繰り返し行い，全てのホストからの応答が返された時点で最も負荷の軽いものを選択する． ■

〔アルゴリズム G: 文献 [44] のホスト選択手順〕

1°: 同報通信による問い合わせ

シェルは全ての RPC 用サーバを管理する特別のサーバ portmap に対して，同報通信により REX サーバの有無を問い合わせ，同時に各ホストが故障していないかどうかを確認する．

2°: 応答の受信

シェルは同報通信に対する各ホストからの応答を待ち，最初の応答を受け取るとその応答の送信元のホストを選択する． ■

実験では，1 台の Sun-3/60(以下，Sun-3/60 A) 上でベンチマークプログラムを 100 回連続して実行するようなシェルスクリプトを同時に複数の端末から実行してプログラムの平均応答時間を測定した．このベンチマークプログラムは引数として与えた回数だけ浮動小数点数の加算を行うもので，シェルにより 1 回ずつ最も負荷の軽いホストに割り当てられる．また，演算回数は平均 100,000 回の指数分布に近似させている．なお，アルゴリズム E, F において，各ホストの処理速度は上記の

MIPS 値をそのまま用い，またアルゴリズム E において式 (5.1) 中のオーバヘッド値はいずれも 0 とした．また，実験の開始直後や終了直前の不安定な状態を避けるため，平均応答時間は各端末で収集した 100 個のデータのうち，中央の 50 個のみを取り出して集計した．

各アルゴリズムにおけるベンチマークプログラムの平均応答時間を図 5.5に示す．この値はコマンドを入力した時点からシェルがコマンド終了を認識した時点までの時間であり，計算機の選択やコマンドの起動に要する時間を含んでいる．このうち，各アルゴリズムにおいて計算機の選択に要する時間を図 5.6に，コマンドの起動に要する時間を図 5.7に示す．これらの図において横軸は端末数，すなわち同時に実行されるコマンドの数を表す．更に，この実験ではアルゴリズム E, F, G において各計算機で実行されたコマンド数についても調査した．その結果を図 5.8に示す．

図 5.5を見ると，負荷分散を行わないアルゴリズム H では平均応答時間は端末数にほぼ比例して増加しているのに対して，アルゴリズム E, F, G では端末数が 10 台になっても平均応答時間は端末が 1 台の時の 2.5 ～ 5.4 倍程度にしかならず，いずれのアルゴリズムも負荷分散機能が有効に動作していると言える．

図 5.5でアルゴリズム E, F, G を比較すると，端末 1 台のときを除いてアルゴリズム E, F の平均応答時間が同程度で他のアルゴリズムより短くなっている．これはアルゴリズム E でローカルホストより応答が遅いホストを無視しても負荷分散の性能に悪影響を与えないことを表している．

アルゴリズム G は端末が 1 台の時は他のアルゴリズムより優れているが，それ以外の時はアルゴリズム E, F より劣っている．これは図 5.8から明らかなように，アルゴリズム G ではローカルホストである Sun-3/60 A と処理速度が最も小さい Sun-3/160 には殆どコマンドを割り当てず，負荷が他の 3 台に集中するためである．なお，端末が 1 台のときにはアルゴリズム C は最も優れているが，これは全てのコマンドを最も処理速度の大きい Sun-3/260 に割り当てているためである．一方，アルゴリズム E, F では割り当てられたコマンドが終了した場合でも直ちにそのホストの負荷が軽くなったことが認識されないため，Sun-3/260 に連続してコマンドが割り当てられず，平均応答時間がアルゴリズム C よりも悪くなっている．

次に，図 5.6, 図 5.7を見ると，アルゴリズム E, F ではホストの選択やコマンド起動に要する時間は全体的には端末数の増加に伴って緩やかに増加する傾向にあるが，あまり大きな差はなくいずれも 0.3 ～ 0.4 秒程度であり，プログラムのコンパイル

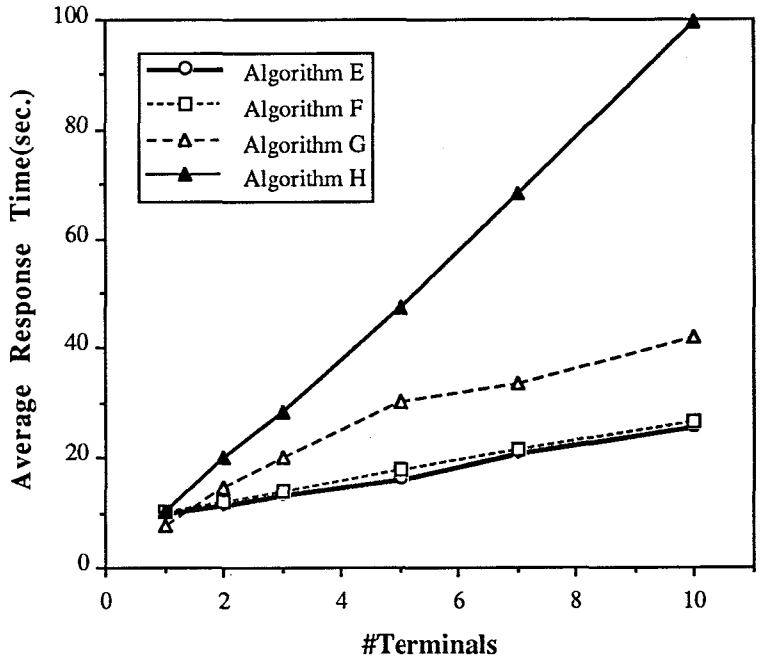


図 5.5: 各アルゴリズムにおけるコマンドの平均応答時間

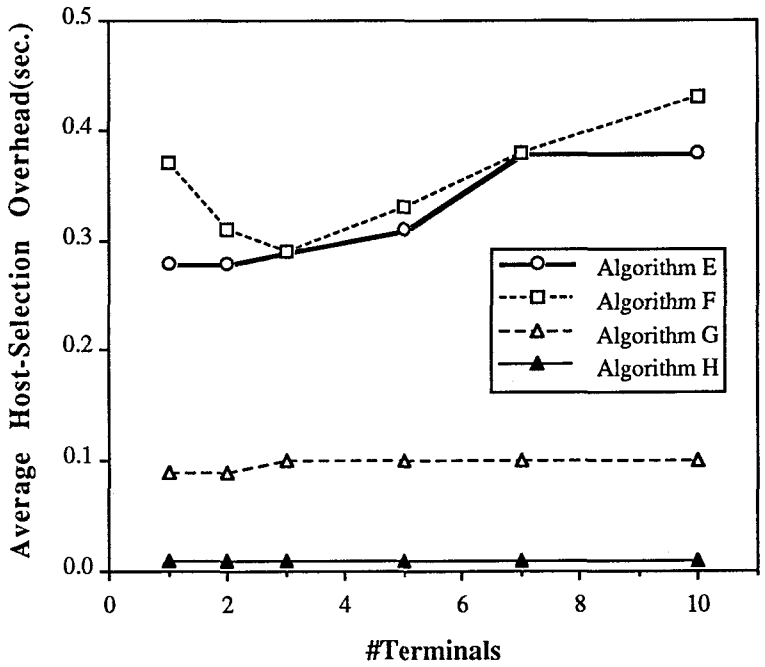


図 5.6: 各アルゴリズムにおけるホスト選択時間

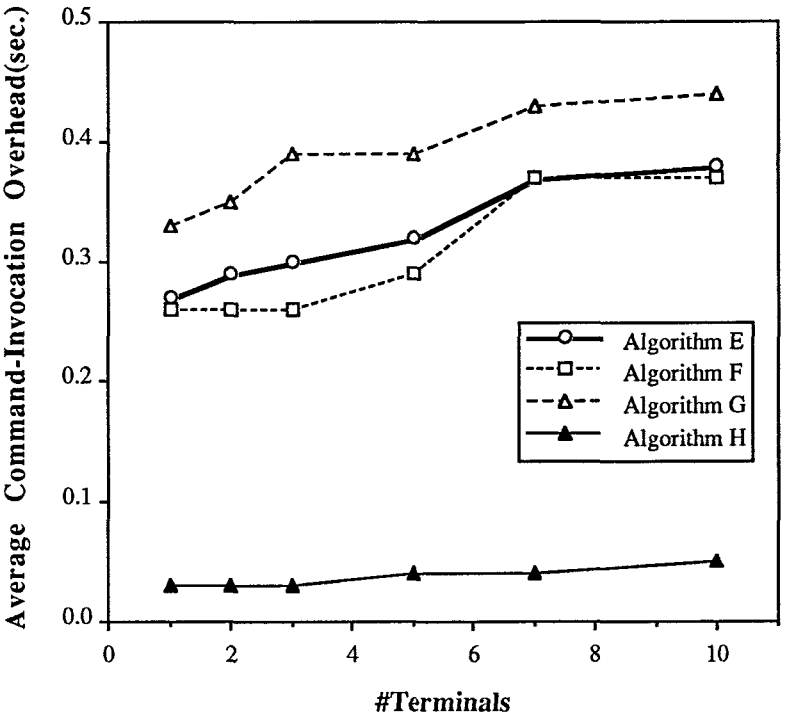


図 5.7: 各アルゴリズムにおけるコマンド起動時間

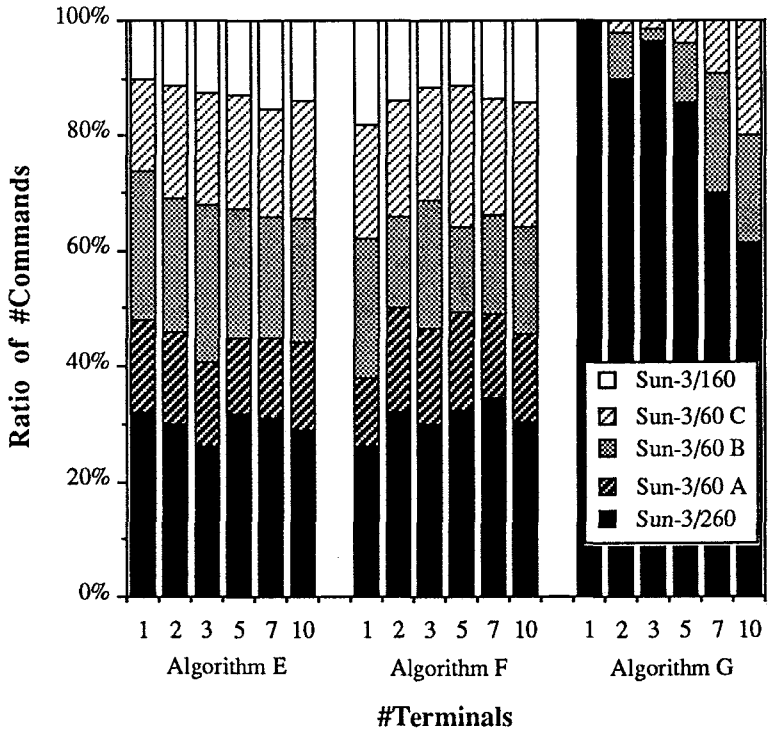


図 5.8: 各ホストにおけるコマンド実行回数

や文書処理など実行時間が長いコマンドに対しては十分小さい値といえる．

アルゴリズムGはアルゴリズムE, Fよりホストの選択に要する時間が短いが，これは問い合わせに対する最初の応答を受け取った時点でホストの選択が終了し，他の計算機からの応答を待つ必要がないためである．また，アルゴリズムGのコマンド起動に要する時間がアルゴリズムE, Fより長くなっているのは，コマンド起動に要する時間が最も短いSun-3/60 Aに殆どコマンドが割り当てられていないためである．

また，図5.8を見ると，アルゴリズムE, Fは比較的均等にコマンドを割り当てていることがわかる．全体の傾向として，両者ともSun-3/260に最も多くコマンドを割り当て，Sun-3/160にはあまりコマンドを割り当てていないが，これは処理速度の違いが負荷を求めるときに考慮されたもので妥当な結果といえる．また，Sun-3/60 Aもコマンド割り当ての割合が小さいが，これはこのホスト上では端末台数と等しい数のシェルが動作しており，その点で他のホストより負荷が重いためと思われる．

以上の結果ならびに考察から，アルゴリズムEは特定のホストに集中することなく適切にホストの選択を行い，性能やホストの選択やコマンドの起動に要する時間の面でアルゴリズムFと同程度の性能を持ち，故障したホストに対する耐故障性を有している点でアルゴリズムFより優れていると言える．従って5.2.1節で提案した耐故障性に対する対策が負荷分散の性能に殆ど影響を与えていないことがわかる．

5.4.3 試作シェルの問題点

5.4.2節で示したシミュレーション実験の他に，本シェルを試験的に運用してみた結果，プログラムのコンパイルや文書処理など比較的時間のかかるコマンドを効果的に負荷分散し，負荷分散機能の有効性を確認することができた．しかし，同時に次に示すようないくつかの問題点があることがわかった．

第1の問題点は，シェルの現在の拡張機能だけでは一部のホストだけをコマンドの種類に応じて割り当ての対象としたり，逆に対象から外したりできない点である．このような機能は，特に異機種が混在するシステムでコンパイルを行う際に必要となる．この機能は，マルチキャスト機能などを用いて同一機種の計算機だけに負荷情報の問い合わせを行う機能をシェルに追加したり，コマンドの種類とローカルホストの機種により応答するかどうかを判断する負荷情報サーバを開発・実装したり

することにより今後実現する予定である.

第2の問題点は，現在の実装では一部の対話的なコマンドには対応していない点である．REX では対話モードが提供されており，これを用いれば対話的なコマンドに対応することが可能であるが，その場合も標準入力の EOF(End of File) を正しく送れない点が新たに問題となるため，現在の実装ではこの機能を使用していない．これについては，今後 rexd の改良などで対応する予定である．

5.5 まとめ

本章では，耐故障性やネットワーク透過性を考慮した動的負荷分散機能の実装方法について述べた．本方法では，同報通信による問い合わせに対してローカルホストより早く応答したホスト (ローカルホスト自身を含む) をコマンド割り当ての対象とする方法を提案し，これにより負荷分散の性能に影響を与えることなく耐故障性を実現することが可能となった．また，REX によるコマンドの遠隔実行機能をシェルに組み込む実装方法を提案し，これによりネットワーク透過性を高めながら自動的に負荷分散を行うことが可能となった.

本章で提案した実装方法は主に UNIX の機能を用いているが，他の OS においても同様の機能が提供されていることが多く，非 UNIX 系の分散システムにおいても本方法を適用することは十分可能であると思われる．例えば，NFS は 2.1.2節で述べたように UNIX 以外にも多くの OS でサポートされており，また REX についても原理的には UNIX 特有の機能を用いておらず，コマンドの遠隔実行機能がサポートされていれば容易に同様の機能を実現することが可能である.

今後の課題としては 5.4.3で示したような制限を設けないように試作したシェルを改善することが挙げられる．また，試作したシェルでは最も負荷の軽いホストの選択に rstatd が返す負荷平均を用いたが，その代わりに第3章や第4章で提案した方法を用いることにより，現在以上に効果的な負荷分散を目指したい.

第 6 章

結論

本論文では，いくつかの UNIX ワークステーションを Ethernet などのローカルエリアネットワーク (LAN) で相互接続することによって構成される典型的な分散システムを主たる対象とし，全体としての効率や性能の向上を図るための動的負荷分散手法や，その実装方法について述べた．

本論文では，典型的な分散システムが持つ種々の機能を示し，これより現時点ではプロセス発送方式による動的負荷分散が現実のシステムに適していることを示した．また，この方式による動的負荷分散において負荷の尺度や負荷情報の収集方法が負荷分散の性能に大きな影響を与えることを指摘し，これらに対して現実の分散システムに適した方法を提案した．

このうち，負荷の尺度については，分散システムで用いられているラウンドロビンスケジューリング方式を対象とし，新しいプロセスの割り当てによる平均応答時間の増加量を基にした負荷の尺度を提案し，シミュレーションにより，従来の尺度である残余仕事量の合計やプロセス数よりも優れていることを示した．

また，負荷情報の収集方法については，LAN が持つ同報通信機能に着目してネットワークの負荷を抑えながら各計算機の負荷情報を遅滞なく収集する方法を提案し，シミュレーションによりこの方法が従来の定期的同報通信方式や入札方式よりも優れていることを示した．

更に，本論文では動的負荷分散を実装する場合には耐故障性やネットワーク透過性が重要であることを示し，これらを考慮した動的負荷分散機能の UNIX への実装方法を提案した．この方法では，同報通信による問い合わせに対してローカルホス

トより早く応答したホストをコマンド割り当ての対象とすることにより耐故障性を実現し，また，REX, NFS, automount を用いた負荷分散機能をシェルに組み込むことによりネットワーク透過性を高めており，実験の結果負荷分散の性能を低下させることなく耐故障性やネットワーク透過性を有していることを確認した．

現在の時点では，第5章で述べた動的負荷分散機能を持つシェルは試作段階であり，第3章で述べた負荷の尺度や第4章で述べた負荷情報の収集方法は実装されていないため，現実のシステムにおける有効性は現在の時点では不明である．また，5.4.3節で述べたように現在の実装ではまだ多くの制限がある．

そこで，今後の課題としてはまず本論文で提案した負荷の尺度や負荷情報の収集方法を組み込み，現実のシステムにおいて性能評価を行うことが挙げられる．また，負荷分散機能の実装については，試作したシェルに見られる制限を解消した実用的な負荷分散機能の実現を目指したい．

その他にも，本研究で提案した負荷の尺度では CPU 制約のプロセスのみを対象とし，I/O 制約のプロセス，対話的プロセスやデーモンプロセスが考慮されていないため，これらを考慮した負荷の尺度の検討も今後の課題として挙げることができる．また，最近徐々にサポートされつつあるマルチキャスト機能や将来主流になるであろう分散 OS の動向にも注意を払い，例えば広域分散システムにおけるマルチキャスト機能を用いた負荷情報の収集方法や分散 OS におけるプロセス移送方式による動的負荷分散などについても検討したいと考えている．

謝辞

本研究の全過程を通じて，終始懇切な御指導，御鞭撻を賜った大阪大学基礎工学部情報工学科の宮原秀夫教授に衷心より感謝の意を表する．

本研究をまとめるにあたり，有益な御助言を頂いた大阪大学基礎工学部情報工学科の菊野亨教授，都倉信樹教授，並びに同大学工学部情報システム工学科の白川功教授に深謝する．特に白川功教授には筆者が大阪大学工学部電子工学科在学中から今日に至るまで常に御指導，御鞭撻を頂いたことをここに記し，心より感謝の意を表する．

筆者が大阪大学工学部電子工学科，同大学院工学研究科前期課程，同大学院基礎工学研究科後期課程に在学中，御指導，御教授頂いた寺田浩詔教授 (現在，大阪大学工学部情報システム工学科教授)，大村皓一助教授 (現在，大阪学院大学教授)，河田亨助教授 (現在，シャープ株式会社取締役技術本部副本部長)，西尾章治郎助教授 (現在，大阪大学工学部情報システム工学科教授)，馬野元秀助教授 (現在，大阪大学工学部精密工学科助教授) に深謝する．

本研究の細部に渡り，終始熱心な御討論及び有益な御助言を頂いた大阪大学大型計算機センターの下條真司助教授に心から謝意を表する．また，筆者の抱くさまざまな疑問に対して熱心に御討論頂いた大阪大学基礎工学部情報工学科の村田正幸助教授に深謝する．

本研究に際し，特に第3章の研究について，有益な御助言ならびに種々の御援助を頂いたシャープ株式会社の千葉徹氏，久保登氏，島田明宏氏，舟渡信彦氏に感謝の意を表する．また，特に第5章の研究について，有益な御助言ならびに種々の御援助を頂いた松下電器産業株式会社の高野豊氏，大阪大学工学部の中野秀男助教授，同大学情報処理教育センターの松浦敏雄助教授，シャープ株式会社の中村眞氏，大阪大学基礎工学部情報工学科の齊藤明紀助手，奈良先端科学技術大学院大学の山口英

助教授をはじめとする日本UNIXユーザ会の方々，電子メールで熱心に御討論頂いた琉球大学の新城靖助手，ならびに卒業研究を通して御協力頂いた奈良工業高等専門学校情報工学科卒業生の若林進氏 (現在，大阪大学工学部情報システム工学科)，関努氏 (現在，豊橋技術科学大学工学部知識情報工学科) に感謝する．

本研究に際し，多大な御援助を頂いた大阪大学基礎工学部情報工学科の荒木俊郎助教授，同大学大型計算機センターの出口弘講師，藤川和利助手，奈良工業高等専門学校専攻科の田中洋氏，大阪大学基礎工学部情報工学科情報ネットワーク学講座の方々，ならびにシャープ株式会社の関係各位に謝意を表する．

最後に，本研究を進めるにあたり，多大な御支援ならびに御便宜を頂いた奈良工業高等専門学校の中西義郎校長，同校情報工学科の上田勝彦学科主任，鈴木忠二教授，五十嵐良教授，永田隆教授，多喜正城助教授，世古忠助教授，工藤英男助教授，浅井文男助教授，植村芳樹助手 (現在，広島大学助手)，小澤誠一助手 (現在，大阪教育大学助手)，下村満子助手，松尾賢一助手，山崎善弘技官 (現在，株式会社奈良情報システム)，西野貴之技官，田中俊之事務官 (現在，文部省)，田中祥仁事務官 (現在，労働事務官)，薮内松子事務官，同校電子計算機室の宮本止戈雄室長，川邊涼子技官をはじめとする奈良工業高等専門学校の諸先生方，事務職員の方々，ならびに筆者の研究室に所属する学生，卒業生諸氏に感謝する．

参考文献

[1] G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin and G. Thiel : "LOCUS : A Network Transparent, High Reliability Distributed System," *Proceedings of the 8th Symposium on Operating Systems Principles,* ACM, pp. 169–177(1981).

[2] D. R. Cheriton : "The V Distributed System," *Communications of the ACM,* Vol. 31, No. 3, pp. 314–333(1988).

[3] S. J. Mullender and A. S. Tanenbaum : "The Design of a Capability-Based Distributed Operating System," *The Computer Journal,* Vol. 29, No. 4, pp. 289–300(1986).

[4] A. Tevanian, Jr. : *Architecture-Independent Virtual Memory Management for Parallel and Distributed Environments: The Mach Approach,* Technical Report CMU-CS-88-106, Department of Computer Science, Carnegie Mellon University(1988).

[5] F. Douglis and J. Ousterhout : "Process Migration in the Sprite Operating System," *Proceedings of the 11th ACM Symposium on Operating Systems Principles,* pp. 18–25(1987).

[6] 新井潤，桜川貴司，立木秀樹，萩野達也，服部隆志，森島晃年："分散環境をサポートするOS ToMの構想—そのプログラミング・モデルとセキュリティ機構—"，日経エレクトロニクス，10月16日号，pp. 187–199(1989).

[7] 横手靖彦，所真理雄："Muse: 次世代計算環境構築のためのオペレーティング・システム"，コンピュータ・システム・シンポジウム論文集，情報処理学会，pp. 107–116(1991).

[8] B. Lyon, G. Sager, J. M. Chang, D. Goldberg, S. Kleiman, T. Lyon, R. Sandberg, D. Walsh and P. Weiss : "Overview of the SUN Network File System," *Proceedings of the Winter 1985 USENIX Conference,* pp. 1–8(1985).

[9] A. P. Rifkin, M. P. Forbes, R. L. Hamilton, M. Sabrio, S. Shah and K. Yueh : "RFS Architectural Overview," *Proceesings of the Summer 1986 USENIX Conference,* pp. 248–259(1986).

[10] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. H. Rosenthal and F. D. Smith : "Andrew: A Distributed Personal Computing Environment," *Communications of the ACM,* Vol. 29, No. 3, pp. 184–201(1986).

[11] J. Bloomer : *Power Programming with RPC,* O'Reilly & Associates(1992).

[12] Y. Wang and R. J. T. Morris : "Load Sharing in Distributed Systems", *IEEE Transactions on Computers,* Vol. C-34, No. 3, pp. 204–217(1985).

[13] C. H. Hsu and J. W.-S. Liu : "Dynamic Load Balancing Algorithms in Homogenious Distributed Systems", *Proceedings of the 6th International Conference on Distributed Computing Systems,* IEEE, pp. 216–223(1986).

[14] R. M. Bryant and R. A. Finkel : "A Stable Distributed Scheduling Algorithm," *Proceedings of the 2nd International Conference on Distributed Computing Systems,* IEEE, pp. 314–323(1981).

[15] D. J. Farber, J. Feldman, F. R. Heinrich, M. D. Hopwood, K. C. Larson, D. C. Loomis and L. A. Rowe : "The Distributed Computing System," *Proceedings of the 7th Annual IEEE Computer Society International Conference,* IEEE, pp. 31–34(1973).

[16] J. A. Stankovic and I. S. Sidhu : "An Adaptive Bidding Algorithm for Processes, Clusters and Distributed Groups," *Proceedings of the 4th International Conference on Distributed Computing Systems,* IEEE, pp. 49–59(1984).

[17] T. C. K. Chow and J. A. Abraham : "Load Balancing in Distributed Systems," *IEEE Transaction on Software Engineering,* Vol. SE-8, No. 7, pp. 401–412(1982).

[18] Y. C. Chow and W. H. Kohler : "Models for Dynamic Load Balancing in Heterogeneous Multiple Processor Systems," *IEEE Transaction on Computers,* Vol. C-28, No. 5, pp. 354–361(1979).

[19] H. S. Stone : "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," *IEEE Transaction on Software Engineering,* Vol. SE-3, No. 1, pp. 88–93(1977).

[20] H. S. Stone : "Critical Load Factors in Distributed Computer Systems," *IEEE Transaction on Software Engneering,* Vol. SE-4, No. 5, pp. 254–258(1978).

[21] V. M. Lo : "Heuristic Algorithms for Task Assignment in Distributed Systems," *Proceedings of the 4th International Conference on Distributed Computing Systems,* IEEE, pp. 30–39(1984).

[22] A. N. Tantawi and D. Towsley : "Optimal Static Load Balancing in Distributed Computer Systems", *Journal of the ACM,* Vol. 32, No. 2, pp. 445–465(1985).

[23] S. Shimojo, H. Miyahara and K. Takashima : "Process Assignment on Distributed System with Communication Contentions", *Computer Networking and Performance Evaluation,* Elsevier Science Publishers B. V., pp. 505–516(1986).

[24] 下條真司，宮原秀夫，高島堅助："通信競合を含めたマルチプロセッサにおけるプロセス割り当て問題"，**電子情報通信学会論文誌 D**，Vol. J68-D, No. 5, pp. 1049–1056(1985).

[25] A. S. Tanenbaum and R. Van Rennese : "Distributed Operating Systems", *Computing Surveys,* ACM, Vol. 17, No. 4, pp. 419–470(1985).

[26] S. J. Leffer, M. K. McKusick, M. J. Karels and J. S. Quarterman : *The Design and Implementation of 4.3BSD UNIX Operating System,* Addison-Wesley(1989).

[27] J. S. Quarterman, A. Silberschatz and J. L. Peterson : "4.2 BSD and 4.3 BSD as Examples of the UNIX System," *Computing Surveys,* ACM, Vol. 17, No. 4, pp. 379–418(1985).

[28] K. S. Trivedi : *Probability & Statistics with Reliability, Queueing, and Computer Science Applications,* Prentice-Hall(1982).

[29] L. Kleinrock : *Queueing Systems, Vol. 2:Computer Applications,* Wiley-Interscience(1976).

[30] N. Yamai, S. Shimojo and H. Miyahara : "A Process Dispatching Algorithm on Multiprocessor Time Sharing Systems", *Proceedings of the 11th Annual International Computer Software and Applications Conference,* IEEE, pp. 681–686(1987).

[31] N .Yamai, S. Shimojo and H. Miyahara : "A Process Dispatching Algorithm on Distributed Time Sharing Systems by Monitoring Network," *Proceedings of 1989 Joint Technichal Conference on Circuits/Systems, Computers and Communications,* IEICE, pp. 324–328(1989).

[32] D. A. Nichols : "Using Idle Workstations in a Shared Computing Environment," *Operatins Systems Review,* Vol. 21, No. 4, pp. 5–12(1987).

[33] D. Kassabian : "v24i088: Simple load-balancing program," Newsgroups: comp.sources.unix, Message-ID:<3615@litchi.bbn.com>, USENET(1991).

[34] J. Bloomer: *Power Programming with RPC,* O'Reilly & Associates(1992).

[35] Sun Microsystems, Inc. : *System & Network Administration,* Sun Microsystems, Inc.(1988).

[36] B. Callaghan and T. Lyon : "The Automounter," *Proceedings of the Winter 1989 USENIX Conference,* pp. 43–51(1989).

[37] J. R. Lyle and C. Lu : "Load Balancing From a Unix Shell," *Proceedings of the 13th Conference of Local Computer Networks,* pp. 181–183 (1988).

[38] E. Levy and A. Silberschatz : "Distributed File Systems: Concepts and Examples," *Computing Surveys,* ACM, Vol. 22, No. 4, pp. 321–374(1990).

[39] J. L. Peterson and A. Silberschatz : オペレーティングシステムの概念，上，培風館 (1987).

[40] 前川守，所真理雄，清水謙多郎編 : 分散オペレーティングシステム，共立出版 (1991).

[41] 宮原秀夫，尾家祐二 : コンピュータネットワーク，森北出版 (1992).

[42] 山井成良，下條真司，宮原秀夫："マルチプロセッサ時分割システムにおける負荷分散アルゴリズム"，**電子情報通信学会論文誌 (D-I)**，Vol. J72-D-I, No. 2, pp. 75–82(1989).

[43] 山井成良，下條真司，宮原秀夫："同報通信機能を持つ分散システムにおける負荷分散アルゴリズム"，**電子情報通信学会論文誌 (D-I)**，Vol. J75-D-I, No. 8, pp. 536–544(1992).

[44] 若林進，山井成良："UNIX における負荷分散の試み"，**第 18 回 UNIX シンポジウム論文集**，日本 UNIX ユーザ会，pp. 161–170(1991).

[45] 関努，若林進，山井成良："UNIX における負荷の推定方法に関する考察"，**第 20 回 UNIX シンポジウム論文集**，日本 UNIX ユーザ会，pp. 12–21(1992).