システムソフトウェアと
オペレーティング・システム 71−15
マルチメディア通信と分散処理 73−15
（1995. 12. 1）

UNIXネットワークにおけるコマンドレベルでの動的負荷分散

山下 貴幸† 高井 昌彰†† 佐藤 義治†
†北海道大学工学部, ††北海道大学大型計算機センター

本論文では、UNIXワークステーションから構成される分散システムを前提としたコマンドレベルの動的負荷分散方式を提案する。本方式ではUNIX内部のRound-Robin待ち行列の前に設けたFIFOの待ち行列に対しタスクを互いに転送することにより負荷の一様化をはかり、かつ負荷の極端な増大を回避する。本方式を分散制御型による動的負荷分散システムとして実際のUNIXネットワーク上に実装する。システムは、各計算機で実行され負荷状態の観測やコマンド遠隔実行先の決定・タスク待ち行列管理などを行うデーモンyshdと、デーモンと通信を行ないRPC(REX)を用いてコマンドを遠隔実行するシェルyshより構成される。

A Dynamic Load Balancing System on a UNIX network

Takayuki Yamashita†, Yoshiaki Takai†† and Yoshiharu Sato†
†Faculty of Engineering, Hokkaido University
††Computing Center, Hokkaido University

In this paper, we propose a command level dynamic load balancing scheme on a distributed system consisting of UNIX workstations. In this scheme, we locate an FIFO task queue in front of a Round-Robin process queue in a UNIX kernel. To balace workloads among computers, each computer sends tasks to the other computer's FIFO queue. This mechanism avoids extremely high load status. We construct a dynamic load balancing system based on the proposed scheme. This system consists of two processes. One is "yshd", a daemon which observes load state, selects remote execution host and manages FIFO queue. The other is "ysh", a command-shell which communicates with "yshd" and performs remote execution by using the RPC(REX).

1 はじめに

近年、複数のコンピュータをLAN(Local Area Network)で接続して、プロセッサ、外部記憶装置や周辺機器などの資源の共有を図る分散システムが急速に普及してきている。このような分散システムにおいて接続されたコンピュータの能力をより有効に利用するために、負荷分散の機能が要求される。これにより、一部のコンピュータだけに負荷が集中する状態を避け、応答時間の短縮や資源利用率の改善など、システム性能の向上を図ることができる。この目的のため、種々の負荷分散方式が多くの研究者によって提案されてきた[1]-[6]。

また、より実際的なアプローチとしてOSレベルで負荷分散、共有仮想記憶などの高度な分散処理を行う分散OSの研究も多数報告されれている[7]。性能・信頼性の向上、スケーラビリティなどの点でOSレベルでの分散処理は極めて有用であると思われるが、多種多様な計算機上で様々なOSが稼働している現状の分散システムから、OSの仕様レベルで統一した高度な分散システムにスムーズに移行させるのは困難であろう。

そこで本研究ではOSを含めた既存のシステムの利用を考慮し、UNIXワークステーションで構成された分散システム上において、負荷情報の管理とタスク転送の決定を各計算機で独立して行う分散制御型の動的負荷分散システムを提案する。

既存の環境をそのまま利用するために、本方式はカーネルに手を加えずにアプリケーションレベルのシステムとして実現する。また、負荷分散の対象となるタスクはUNIXにおけるシェルコマンドを単位とする。

負荷分散アルゴリズムとしては、負荷の重い計算機からタスクの転送要求を発するsender-initiated algorithms [6] を基本としたアルゴリズムを採用する。

本論文では、まず負荷分散システムのモデルと実装について述べる。次に実行例を示し、最後に今後の課題について述べる。

2 対象とする分散システム

本方式はUNIXワークステーションで構成される分散システムを対象とする。

コマンドの遠隔実行については、多くのUNIXに実装されているRPC(Remote Procedure Call)機構によるREX(Remote Execution)サービスが使用可能であるものとする。REXはNFS (Network File System)を用いて遠隔マシンにユーザの現ワーキングディレクトリをマウントすることによりローカルなファイルの使用が可能となり、ネットワーク透過性を実現する。また、名前の透過性、つまり遠隔実行先となる各計算機上に同一名・同一機能の実行可能コマンドが存在することを前提とする。ただし、実行可能なコマンドを各計算機にそれぞれに登録することにより、適宜計算機を選択して分散させることも可能とする。

一般に「タスク」と「プロセス」は同義語として扱われることが多いが、本論文ではこの2つの単語を区別して扱う。「タスク」とはユーザから実行を依頼された仕事で、まだCPUによる処理が開始されていないものを指し、CPUによる処理が開始されたものを「プロセス」と呼ぶこととする。

本方式ではタスクのみを負荷分散の対象とする。また、各タスクは依存関係がなく実行順序を問わないものとする。ここでいうタスクはユーザにより実行されたUNIXのコマンドである。

ここで、分散システムを構成する各ノードのモデルを図1に示す。UNIXでは内部にRound-Robin方式によるプロセス待ち行列を持ち、プロセスを時分割で実行する。本方式では、各計算機のプロセス待ち行列の前にFIFOのタスク待ち行列を設けることにより、多数のプロセスが同時に実行され計算機の負荷が極端に大きくなる状態を回避する。

ユーザにより実行されたタスクは、負荷分散アルゴリズムにより各計算機のタスク待ち行列に割り当てられた後、先頭から順にOS内のプロセス待ち行列に入り実行される。

ただし、対話性が必要なタスクについてはタスク待ち行列に割り当てず、即実行することを可能とする。

3 動的負荷分散アルゴリズム

3.1 計算機の負荷

負荷分散アルゴリズムを考える場合、計算機の負荷を表す指標が必要である。

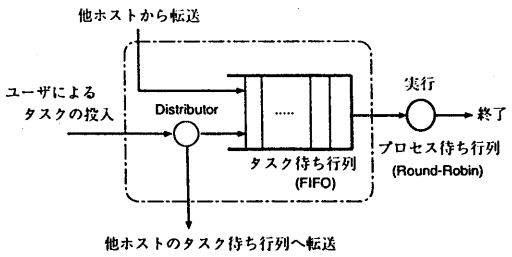

図 1: ノードモデル

本方式では、計算機 H_i の負荷 L_i を、タスク待ち行列長 q_i、ロードアベレージ l_i（過去 1 分間のプロセス待ち行列の長さの標本平均)、相対処理速度 s_i を用いて次のように表す。

$$L_i = \frac{aq_i + bl_i}{s_i} \qquad (a, b \text{は正の定数})$$

ここで s_i は、各計算機の処理速度の差を考慮するために、あらかじめ設定する正規化した処理速度であり、何らかのベンチマークにより求めるものとする。基準となる計算機を 1 台決め、その s_i を 1 とした比で表し、処理速度の早い計算機ほど値が小さくなるように設定する。

処理内容の違いにより s_i は厳密な意味での処理速度差とはなりえないが、ある程度の目安として使用する。

L_i がある閾値 Th 以上になった場合、負荷が重いと判断することとする。

3.2 負荷分散のアルゴリズム

本方式では分散制御型の動的負荷分散アルゴリズムの中でも基本的なものの一つである sender-initiated algorithms [6] を基本としたアルゴリズムを採用する。

このアルゴリズムはタスク転送の送り手、つまり負荷の重い計算機からタスク転送要求を送出する手法である。

転送要求送出方法については次の 3 つの方式を採用し比較を行う。

- ユニキャスト方式

  まずランダムに選ばれた計算機 1 台に対してタスク転送要求を送出し、要求先の計算機の負荷が軽いことがわかれば、そこに対してタスクを転送する。そうでなければ、負荷の軽い計算機が見つかるか、あるいは一定回数の限度まで転送要求を繰り返す。

- ブロードキャスト方式

  1 度に全ての計算機に対して転送要求を送出する。要求が受け入れられた計算機の中からランダムに転送先を決定する。

- マルチキャスト方式

  全計算機の中から選ばれた一部の集合に対して転送要求を送出する。要求が受け入れられた計算機の中からランダムに転送先を決定する。ここでは集合はランダムに選択することとする。

いずれの方式も、転送に失敗した場合はタスクが投入された計算機で実行される。

4 システムの設計と実装

4.1 システムの構成

本方式は、次の 2 つのプロセスにより構成される（図 2)。

- シェル (ysh)

  ユーザがコマンドを実行するためのインターフェースプログラムである。デーモンとメッセージ通信を行い、必要に応じて REX を用いてコマンドを遠隔実行する。

- デーモン (yshd)

  各計算機上で 1 つだけ実行され、相互にメッセージ通信を行うことにより負荷状態の観測やコマンド遠隔実行先の決定・タスク待ち行列管理などを行う。

4.2 システム動作の詳細

ユーザの実行したコマンドは以下の過程を経て実行される。ysh にはあらかじめ遠隔実行可能なコマンド列を初期化ファイル、またはコマンドラインより登録しておく。

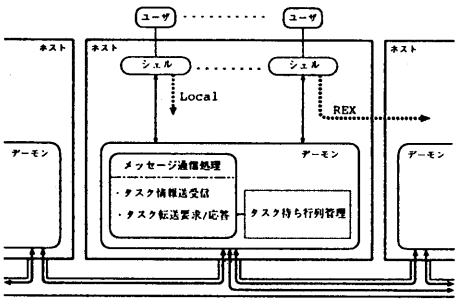

図 2: システムの構成

1. ysh 上でコマンドが実行されると、それが遠隔実行可能なコマンドの場合、コマンドに関する情報が yshd へ送信される。

2. コマンドに関する情報を受けとったデーモンは負荷分散アルゴリズムにしたがい、ローカル実行するか遠隔実行するかを決定する。

3-a. ローカル実行の場合、そのタスクはローカルホストのタスク待ち行列に投入される。

3-b. 遠隔実行の場合、他の計算機で実行されている yshd とメッセージ通信を行い遠隔実行先を決定する。 決定した実行先の yshd に対してタスクに関する情報を転送する。

4. 各 yshd はタスク待ち行列の先頭のタスクから順に実行処理を行う。ローカルホストより投入されたタスクの場合、投入した ysh へ実行開始メッセージを 送信する。他の計算機から転送されたタスクの場合、転送元の yshd に対してメッセージを送信する。転送元 yshd は ysh に対して遠隔実行先の情報を含んだ遠隔実行開始メッセージを送信する。

5-a. 実行開始メッセージを受信した ysh は fork して実行を開始する。実行終了後、ysh から yshd に対し実行終了メッセージが送信され、タスク待ち行列からそのタスクが削除される。

5-b. 遠隔実行開始メッセージを受信した ysh は REX によりコマンドを実行する。実行終了後、ysh から yshd に対し実行終了メッセージが送信される。受信した yshd は転送先の yshd に遠隔実行終了メッセージを送信する。

ユニキャスト方式を例にとったメッセージ通信の流れを図 3に示す。各プロセスの添字 i は実行されているが計算機 H_i であることを表す。

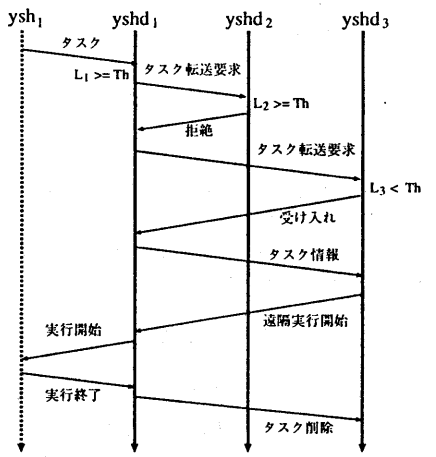

図 3: ユニキャスト方式におけるメッセージ通信

ysh_1 で実行されたコマンドはまず $yshd_1$ に送信される。H_1 の負荷が重いと判断され、$yshd_1$ から $yshd_2$ に対してタスク転送要求が送信される。H_2 の負荷も重いと判断され要求は拒絶される。次に $yshd_3$ に対してタスク転送要求を行い、ここで H_3 の負荷が軽いと判断され要求が受け入れられる。$yshd_1$ から $yshd_3$ にタスク情報が転送され、$yshd_3$ のタスク待ち行列に投入される。$yshd_3$ でそのタスクの実行処理が開始されると $yshd_1$ に対して遠隔実行開始メッセージが送信され、$yshd_1$ から ysh_1 へ実行開始メッセージが送信される。実行が終了すると、実行終了メッセージが $yshd_1$ へ送信され、さらに $yshd_3$ へ遠隔実行終了メッセージが送信され遠隔実行は終了する。

4.3 ysh のビルトインコマンド

状況に応じて、ローカル実行・遠隔実行を選択可能にするために、ysh にいくつかのビルトインコマンドを用意する。以下、遠隔実行可能として登録さ

れているコマンドを dc (distributable commands) と表記する。

- prdc (print dc)
  dc リストの表示
- addc command (add dc)
  dc の追加
- rmdc command (remove dc)
  dc の削除
- ie command (instantly execute)
  待ち行列をバイパスしてローカル実行

4.4 システムの実装

これまでに述べた設計に基づき、ysh と yshd を UNIX ネットワーク上に実装した。使用した OS は SunOS 4.1.4 と IRIX 5.3 である。

遠隔実行に用いる REX はサーバ・クライアント方式のサービスである。クライアントはネットワーク上の他の計算機で実行されている REX サーバに対して、コマンド名・環境変数・ワーキングディレクトリなどの情報を送信する。REX サーバはその情報をもとにクライアントのワーキングディレクトリを NFS マウントしローカルファイルの利用を可能とする。次にクライアントと TCP コネクションを張り標準入出力・標準エラー出力を接続した後コマンドを実行する。

本方式では ysh がクライアントに相当する。

REX サーバはユーザの確認に RPC Unix 認証を用いているために、サーバとクライアントが同じ NIS (Network Infomation Service) ドメインを共有、あるいは両方のマシンにユーザのエントリが存在しなければならない。ここで用いた UNIX ネットワークでは、NIS により全計算機のアカウントを共有している。

5 評価

5.1 実行例

実行に関する情報を逐次表示する verbose モードによる実行例を以下に示す。以下で用いる "loops" というコマンドは簡単な浮動小数点演算を繰り返し、実行に要した時間と実行ホスト名を表示するコマンドである。

```
 1: YSH.ss2:~ > prdc
 2: loops
 3: YSH.ss2:~ > loops -c 30
 4: YSH: "loops" listed in dclist.
 5: YSH: "ss2" Heavy. Try Remote Execute.
 6: YSH: "loops -c 30" execute at "aruba"
 7:  Number of calculations: 30 (x 1000000)
 8:  Elasped_Time : 5.552
 9:  Executed on: aruba
10: YSH: Remote Execute Successed.
11: YSH.ss2:~ >
```

行頭に "YSH:" とある行は ysh の出力したメッセージである。prdc コマンドで遠隔実行可能なコマンドリストを表示する。"loops" が登録されている (1–2 行目)。loops を実行する (3 行目) と loops は遠隔実行可能として登録されていることを表示する (4 行目)。ローカルホスト ss2 の負荷が重いため負荷分散手続きを開始する (5 行目)。遠隔実行先が aruba に決定し実行が開始される (6 行目)。loops コマンドの実行結果が出力され (7–9 行目)、遠隔実行が終了し (10 行目)、プロンプトに戻る (11 行目)。

5.2 評価実験

各アルゴリズムにおけるタスクの平均応答時間の比較を行う。

使用した分散システムは SUN SPARCstation2, classic, 20 と SGI Indy ×5 の計 8 台 (順に H_1〜H_8 とする) を 10Mbps のイーサネットで接続した UNIX ネットワークである。H_2〜H_5 の 4 台にタスクを発生させその平均応答時間を測定した。

タスクとしては第 5.1節で使用した loops コマンドを用い、ループ回数を平均 60×10^6 回の指数分布、到着間隔を平均λ 秒のポアソン到着とし、4 台の各ホスト毎に 100 個のタスクを生成させた。ループ回数 60×10^6回のタスク 1 個の応答時間は H_2〜H_5 の平均で約 11.1 秒である。λ を変化させた結果を図 4に示す。横軸を平均到着率ではなく、平均到着間隔 λ 秒 にとってあるのは、同じタスクでも計算機により実行時間が異なるため、平均到着率が一意とならないためである。

実験の結果、負荷分散を行わない場合は、到着間隔が短くなるにつれ多数のプロセスが同時に実

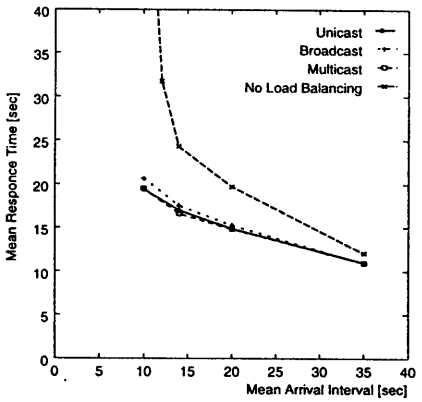

図 4: 平均応答時間の比較

行される確率が高くなるため実行速度が極端に落ち、応答時間の増大が顕著に現れている。負荷分散を行った場合、タスク待ち行列を設けたこともあり、極端な実行速度の低下は避けられている。また、ユニキャストとマルチキャストの間にほとんど差は見られないが、ブロードキャストはわずかに応答時間が増加している。これはメッセージ通信量の増加によるタスク転送に要するオーバーヘッドの増加が影響していると考えられるが、有意と言える差ではないと思われる。これは、オーバヘッドの増大量と比較してタスク 1 つあたりの平均実行時間が十分に長いため、全体としては影響が現れにくいためであると考えられる。

6 おわりに

本研究では、既存のシステムの利用を考慮し、UNIX ワークステーションで構成された分散システム上における分散制御型の動的負荷分散システムを提案し、REX サービスを用いるシステムとして実装した。実験結果から、本方式を使用した場合、負荷分散を行わないときよりもタスクの平均応答時間が短縮することが認められたが、転送要求送出方法による差は大きくは現れなかった。

今後の課題としては、現段階ではサブセット的なシェルである ysh を、csh などの既存のシェルの拡張として作成することによる高機能化、タスク待ち行列内のタスク転送といった負荷分散アルゴリズムの改良、タスク依存関係情報の付加よる同期実行機能の追加などがあげられる。

参考文献

[1] 山下貴幸, 棟朝雅晴, 高井昌彰, 佐藤義治: "GA と確率学習オートマトンを用いた動的負荷分散システム", 情報処理学会第 49 回全国大会講演論文集, Vol.2, pp.251–252, Sep 1994.

[2] 山下貴幸, 棟朝雅晴, 高井昌彰, 佐藤義治: "UNIX ネットワークにおける動的負荷分散への遺伝的操作の導入", 情報処理学会第 50 回全国大会講演論文集, Vol.1, pp.249–250, Mar 1995.

[3] 山井成良, 若林進, 下条真司, 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, Vol.J77-D-I, No.7, pp.483–492, Jul 1994.

[4] T. T. Y. Suen and J. S. K. Wong: "Efficient Task Migration Algorithm for Distributed Systems", *IEEE Trans. on Parallel and Distributed Syst.*, Vol.3, No.4, pp.488–499, July 1992.

[5] Thomas Kunz: "The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme", *IEEE Trans. on Software Eng.*, Vol. 17, No. 7, pp.725–730, July 1991.

[6] N.G.Shivaratri, P.Krueger and M.Singhal: "Load Distributing for Locally Distributed Systems", *IEEE COMPUTER*, Vol.25, No.12, pp.33–44, Dec 1992.

[7] 清水謙多郎: "分散 OS の研究動向", 情報処理学会誌, Vol.36, No.8, pp.699–701, Aug 1995.