システムソフトウェアと
オペレーティング・システム 85－5
（2000．8．4）

汎用クラスタ上の資源情報を用いた HTTP サーバにおける負荷分散性能の評価

大平 怜† 松本 尚† 平木 敬†

ワークステーションを連結させた汎用クラスタ上の HTTP サーバは毎秒のリクエスト数が一定の場合は固定数のノード（固定ノード）上で動くが、リクエスト数が突発的に増えた場合は固定ノード以外のマシンも動的に使用して対処すべきである。本論文では、汎用クラスタ上で OS が提供する資源情報を用いて、各ノードの CPU 負荷の増減に応じて台数の増減を判断する HTTP サーバのモデルについて述べる。更に、その判断の基準となる負荷の閾値と判断の頻度によってサーバモデルの負荷分散性能がどう変化するかをシミュレーションを用いて実験する。実験の結果、最適なパラメータを選ぶことによって固定台数方式より優れた性能が得られることが示された。

Performance Evaluation of Load Balancing in HTTP Server Using Resource Information on General-Purpose Cluster

Rei Odaira ,† Takashi Matsumoto † and Kei Hiraki†

An HTTP server on general-purpose cluster consisting of combined workstations runs on fixed number of nodes (fixed nodes) when the number of requests per second is constant. However, when the number of requests suddenly increases, the server should cope with them by dynamically utilizing machines which are not the member of the fixed nodes. In this paper, we describe the model of HTTP server that uses resource information provided by OS on general-purpose cluster, and determines whether to adjust the number of nodes in response to change of CPU load on each node. Then, by using simulation, we experiment with how the load-balancing performance of the server model changes by the threshold of load on which decision of adjustment depends, and by the frequency of decision. Simulation results show that the dynamic nodes method has better performance than the fixed nodes method by choosing optimal parameters.

1. はじめに

近年、高負荷がかかる HTTP サーバではネットワークで結合されたマシンからなる専用クラスタが用いられ始めている。専用クラスタを用いたサーバ☆ではサーバプロセスが動いているノード数は固定である [5]。ノード数固定方式の欠点として急激なリクエスト数増加への対処方法が挙げられる。HTTP サーバへのワークロードは一定ではなく、突発的に増大することが知られている [5]。突発的なリクエスト数の増加にノード数固定のサーバで対応するためには、クラスタにマシンを追加してサーバの動いているノード数を予め増やす必要がある。

一方、近年の LAN 性能の向上によりネットワークで結合されたワークステーション群 (Network of Workstation; NOW) からなる大規模汎用クラスタが実現されており、そのための OS が多数研究されている [1] [6] [4] [7]。本論文では、NOW 方式の汎用クラスタ上で高負荷な HTTP サーバを実行させることを対象とする。汎用クラスタ環境では多数のユーザの様々な並列プロセスがそれぞれマシン群の一部を使って同時に実行される。従って、HTTP サーバプロセスもクラスタ内の一部のノードを使って実行され、コンスタントな負荷がかかっている状態では固定台数方式と同様に一定数のノード上で実行される。しかし突発的にリクエスト数が増加した場合には、サーバに割り付けられていない計算機資源を利用して動的にノード数を増減させると性能を上げることができる。

しかしながら、汎用クラスタ環境下では必要以上にノード数を増やすと並列度が増すことにより HTTP サーバの性能が低下する。また、他のユーザプロセスの負荷の影響を受けてサーバの性能が低下する可能性が増す。従って動的に利用するノードの使用時間をなるべく

† 東京大学大学院理学系研究科情報科学専攻
Department of Information Science, Faculty of Science, University of Tokyo

☆ 以下、特に述べない限り、HTTP サーバもしくは HTTP サーバプロセスを指す。

短くし、かつ同時に突発的なリクエスト増による性能の悪化を抑える必要がある。

以上のことを考慮し、本論文では動的台数方式の突発的リクエスト増に対処する性能を固定台数方式に対して評価し比較する基準を以下のように定める：本論文で扱う「突発的リクエスト増に対処する性能」とは、予想される最大のリクエスト増に対してどれだけ少ない台数で指定時間内にレスポンス時間を回復できるか、を意味する。指定時間内にレスポンス時間を回復するためには、サーバがその間に来たリクエストを時間内にすべて処理し終わる必要がある。従って固定台数方式の性能を表す指標として、時間内にすべてのリクエストを処理するための必要十分なノード数 n_{opt} を用いる。次に動的台数方式の性能を評価するために、動的に利用したノードの使用時間を台数 n_s に換算する。動的台数方式が時間内にすべてのリクエストを処理し終わっているならば、n_s を n_{opt} と比較する。$n_s < n_{opt}$ ならば、この動的台数方式はこのワークロードに関して固定台数方式より優れた処理性能を持つ。

種々のワークロードについてそれぞれ最適な動的台数方式を選択する。各々の最適な動的台数方式がそのワークロードに対する最適な固定台数方式より優れた性能を示すならば、動的台数方式は固定台数方式より優れた方法であると言える。

本論文では汎用クラスタ OS が提供する機能を動的台数方式の HTTP サーバがどう利用するかを第 2 節で説明する。第 3 節では HTTP サーバモデルを具体的に述べる。このモデルでは資源情報を用いてフロントエンドがリクエストをバックエンドに分散させる。また同時に、資源情報を利用して CPU 負荷の変化に合わせてバックエンドの台数を動的に変化させる。第 4 節において、動的台数方式の性能評価基準を詳述する。第 5 節では前述のサーバモデルの性能を評価するためのシミュレーション実験モデルを述べる。第 6 節でその実験結果を考察し、第 7 節でまとめる。

2. 汎用クラスタ

クラスタシステムには使用方法により大別して 2 種類が存在する。一つは Beowulf プロジェクト☆に代表される方式であり、常時クラスタ内の全マシンが単一目的のために使用される。並列プログラムは起動された時点で全てのノードへ割り付けられる。複数のユーザが任意のプロセスを立ち上げてクラスタ内のマシンを共有しながら使用することは想定されていない。

もう一つの方式は複数のユーザが複数の並列プロセスを同時に実行させるマルチプロセス汎用クラスタ方式である。この方式では各並列プロセスはクラスタ内の一部のノードを使用して実行される。マルチプロセス汎用クラスタ方式の中でも特に、各ユーザが所有するワークステーションをネットワークで結合する方式を Network of Workstation (NOW) 方式と呼び、実現例には SSS-CORE [7]、Sprite [6]、Condor [4] などがある。NOW 方式では各マシンは個人所有であるため、多くの時間アイドル状態にあると考えられる。NOW 方式ではこの余剰計算機資源を有効活用することを考える。ユーザが起動した並列プロセスは、自身が所有するマシン上とクラスタ内にあるアイドル状態のワークステーション上で実行される。本論文では NOW 方式のマルチプロセス汎用クラスタ上で HTTP サーバを動かすことを対象とする。

2.1 汎用クラスタ上の HTTP サーバ

我々の興味は突発的なリクエスト増に対処することであるため、HTTP サーバはコンスタントなリクエストを処理している状態では必要十分な一定数の固定ノードを利用し続けると仮定する。急激にリクエスト数が増大した場合、サーバはなんらかの方法でそれを検知し、汎用クラスタであることの利点を生かして台数を動的に増やす。ここで我々の方式では NOW 方式の汎用クラスタ OS が提供する資源情報と遠隔実行機能を利用する。

突発的なリクエスト増が元に戻った場合、サーバは増えた台数を元に戻す。なぜならば、サーバのスループットはある台数以上増やしても向上せず、並列度が上がることによりむしろ低下するからである。また、台数を増やすことによって他のユーザのプロセスと HTTP サーバプロセスが同じノード上で動く可能性が増す。もし他のユーザのプロセスによりノードの負荷が上がった場合、同じノード上で動いている HTTP サーバプロセスの性能が下がる。以上のことから汎用クラスタにおいて HTTP サーバは必要以上の台数を使うべきではない。

我々の提案する方式では台数を減らす場合にも資源情報を利用して負荷の低下を検知する。

2.2 資源情報

NOW 方式のクラスタではユーザが自身のプロセスがどのマシンで動いているかを意識して制御可能である。従って、システムは何らかの判断基準をユーザに提供する。最も単純なものでは、どのマシンがアイドル状態にあるかをユーザに知らせる [2]。本論文では、より具体的に各マシンの資源情報を OS がユーザに提供することを仮定する。資源情報が提供されるクラスタシステムには SSS-CORE [7], GLUnix [3] などが存在する。本論文で扱う資源情報は、CPU 負荷、ネットワーク負荷、及びそれらの過去一定時間の平均値のことを指す。

突発的なリクエスト数の増加により HTTP サーバの各ノードの負荷は上昇するため、サーバは資源情報を利用することでそれを検知することができる。また、資源情報を利用することにより同じノードで HTTP サーバ以外のプロセスが実行を開始したことによる負荷上昇も検知することができるが、他プロセスとの相互作用に関して本論文ではこれ以上詳しくは述べない。

☆ http://www.beowulf.org/

2.3 遠隔実行

汎用クラスタでは新たにプロセスを起動する場合、親プロセスとは別のマシン（プロセッサ）に割り当てることができる。また、プロセス実行中にコンテキストを他のマシンに移すプロセスマイグレーションをサポートするOSも存在する [2] [4] [1]。NOW方式のクラスタでは遠隔実行やマイグレーションをユーザレベルの判断で行うことができる。

HTTPサーバは遠隔実行機能を利用し、リクエスト数の増加を検知すると空いているマシンを探して新たにサーバプロセスを立ち上げる。一方、サーバが台数を減らす場合、リクエストを処理中でないサーバプロセスは単に終了させればよい。リクエストを処理中の場合は残りのノードにマイグレーションさせることが考えられるが、マイグレーションはコンテキストの転送によるコストが大きい [2]。またHTTPのコネクションは長時間持続されることはないので、我々の提案する方式ではサーバプロセスがリクエストを処理し終わるまでそのノードを使用し続けることを仮定する。

3. HTTP サーバモデル

前節で述べた動的台数方式のHTTPサーバのモデルをシミュレーションで評価するための具体的なサーバモデルを述べる。我々のモデルではApache*と同等の1コネクション1プロセス方式を複数ノードに拡張する。ただし、動的台数方式自体はイベント駆動方式やマルチスレッド方式にも適用可能である。

3.1 サーバ構成

サーバ全体の構成を図1に示す。全体は大きくフロントエンドノードとバックエンドノードに分かれる。各バックエンドノードには複数のServerProcessと1つのParentProcessが動作している。すべてのHTTPクライアントはフロントエンドノードに向けてサービスを要求する。フロントエンドはクライアントからのサービス要求を各バックエンドノードに分散する役割を持つ。TCPコネクション上でクライアントからHTTPリクエストを受け取り、実際のHTTPサーバとしてのサービスを提供するのはServerProcessの役割である。ParentProcessは自分が動いているバックエンドノード上のServerProcessの数を管理する。

フロントエンドはHTTPクライアントからサービスを要求するHTTP/TCPのSYNパケットを受け取ると、OSが提供する資源情報を利用してその時点で最も負荷の軽いバックエンドノードを選んでSYNパケットを転送する。ここで利用する資源情報にはCPU負荷とネットワーク負荷が考えられる。本論文のモデルでは近年の動的コンテンツの増加やシミュレーションで用いるワークロード量を考慮してCPU負荷（＝実行待ち行列中のプロセス数）を利用する。フロントエンドがいったんあるSYNパケットを転送すると、以降そのコネクションに関してHTTPクライアントから来るパケットはすべてOSレベルでバックエンドへ転送される。

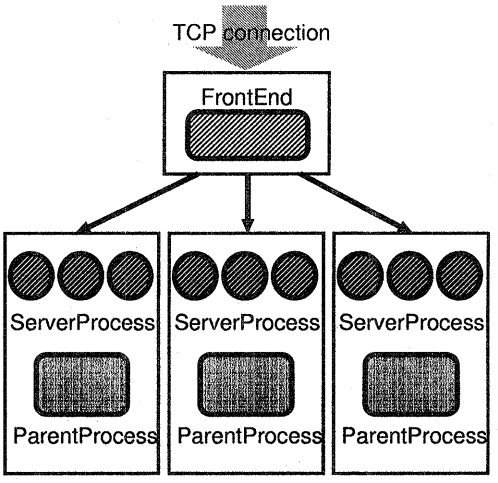

図1 HTTPサーバ構成

各バックエンドノード上の複数のServerProcessはHTTPポートでコネクションを待ち受けている。フロントエンドが転送したSYNパケットをバックエンドが受け取るとHTTPクライアントとの間でTCPコネクションを確立し、ServerProcessの1つが起こされてそのコネクションを担当する。ServerProcessからHTTPクライアントへのデータ転送はフロントエンドノードを経由する必要はない。ServerProcessはリクエストを処理し終えると再びHTTPポートで待つ。

ParentProcessはタイマで1秒に1回起動され、リクエストを処理中でない自ノード中のServerProcessの数が11以上の場合は1つ終了させる。また、5つ未満の場合は新たにServerProcessを生成する。1秒に1回という間隔とServerProcessを終了させる際の閾値11はApacheで実際に採用されている数字である。シミュレーションによる予備実験において、リクエストを処理中でない自ノード中のServerProcess数は高負荷時には0～3個であったので、ServerProcessを生成する閾値として5を選択した。生成されるServerProcess数は初めは1つであり、毎秒連続して生成される場合は2の巾乗で増え、一度に最大16個のServerProcessを生成する。一度にこれ以上生成すると無駄なServerProcessが増える。

3.2 台数増減方法

フロントエンドは一定時間毎（0.5 sec、1.0 sec、1.5 secの3通り）にバックエンドのCPU負荷情報を用いて台数増減の判断を行う。フロントエンドは過去10 tick(= 100 msec)のCPU負荷平均の全バックエンドノードにわたる平均がMaxCPULoad値を越えると

* http://www.apache.org/

1台ノードを増やす。新たなノードは最もCPU負荷が軽いものが選ばれる。ここでtickとはOSへのタイマ割り込みの単位であり、1 tick毎にプロセスをスケジューリングする機会が訪れる。

過去一定時間のCPU負荷平均ではなく現時点でのCPU負荷を用いると1 tick間程度の短時間のCPU負荷増加に影響を受けて台数が増える可能性がある。従って我々の方式では過去10 tick間のCPU負荷平均の情報を使用する。これよりも長い時間のCPU負荷平均を使うと急激なリクエスト数の増加に対応できなくなる。

また、フロントエンドがリクエストの負荷分散を行うにも関わらずバックエンドノード間で負荷の不均衡が生じる可能性がある。従って、「各バックエンドノードのCPU負荷がMaxCPULoadを越えた時」ではなく「全バックエンドノードにわたるCPU負荷の平均値がMaxCPULoadを越えた時」に台数を増加する。

一方、過去10 tickのCPU負荷平均の全バックエンドノードにわたる平均が

$$\mathrm{MinCPULoad} = \frac{(N-1)\times(\mathrm{MaxCPULoad}-1)}{N}$$

(ここで、Nは現在のバックエンドのノード数を表す)を下回ると1台ノードを減らす。たとえどのノードの使用を中止したとしても使用を中止した直後に全体のCPU負荷平均がMaxCPULoadを越えないように、MinCPULoadの式を設定した。このことから、ノード数を減らした直後にまた増やす必要性が生じることはなくなる。使用を中止するノードは一番最近に使用を開始したノードが選ばれる。従って突発的なリクエスト増が元に戻ると、HTTPサーバのノード構成は元のコンスタントな負荷がかかっていた時の構成に戻る。

4. 動的台数方式の性能評価基準

動的台数方式を採用する場合、急激なリクエスト増に対するその方式の処理性能を評価する基準が必要である。突発的にリクエスト数が増加すると負荷の増加によりHTTPサーバのレスポンス時間が悪化する。従って、指定時間内にレスポンス時間を回復することができるかどうかでHTTPサーバの処理性能を評価することができる。指定時間内にレスポンス時間を回復するためには、サーバがその間に来たリクエストを時間内にすべて処理し終わる必要がある。よって、リクエストが増加し始めた時点から指定時間後までにその間のリクエストをすべて処理し終わっているかどうかを性能評価の基準とする。さらに、第2節で述べたとおり、動的台数方式では動的に使うノードの使用時間がなるべく短い方がよい。このことから、使用時間を以下に述べる方法でノードの台数に換算することで様々な動的台数方式の間や固定台数方式との比較を行う。

以下では突発的なリクエスト数増加の最も単純な例として図2に示すHTTPワークロード(workload1)を考える。図2中の"Pageview"とは1つのWebコンテンツとそこへ埋め込まれている画像ファイルの集合への一連のリクエストを指す。従って、HTTP/1.1においてはpersistentコネクションの数と等価である。また、毎秒P_{low}ページビューと毎秒P_{high}ページビューのコンスタントなワークロードをそれぞれworkload2, workload3とする。

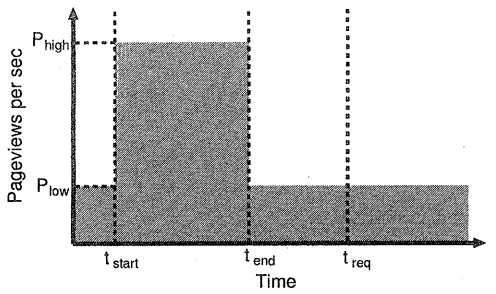

図2 突発的なHTTPワークロード例(workload1)

4.1 固定台数方式

固定台数方式でworkload2とworkload3を処理した場合の、台数とスループットの関係は図3のようになる。また、workload1を処理した場合のt_{start}からt_{req}までのスループットを計測すると、図3のworkload1のようになる。thr_{max}はt_{start}～t_{req}間にその間のすべてのリクエストを処理した場合のスループットを表す。$t_{start} < t_{req} \leq t_{end}$の場合、workload1のグラフはworkload3と一致する。また、t_{req}を無限遠とした場合workload2と一致する。従って、workload1のグラフは必ずworkload2とworkload3の中間に位置する。

workload1のグラフは、n_{opt}台未満ではその間のリクエストを完全には処理し切れておらず、時刻t_{req}までにレスポンス時間を元の値にまで回復できないことを意味する。一方n_{opt}台よりノード数が多いとサーバの処理能力が余ることを意味する。従って、固定台数方式で突発的な毎秒P_{high}ページビューの負荷に対して時刻t_{req}までにレスポンス時間を回復するためにはn_{opt}台が最適である。

4.2 動的台数方式

動的台数方式の場合もworkload1を処理させて前節と同様にt_{start}からt_{req}までのスループットを計測する。もし図3のthr_{max}と同等の性能が得られた場合、その動的台数方式は突発的な毎秒P_{high}ページビューの負荷に対して十分な処理能力を有している。

また、動的台数方式における時間経過と台数の関係の一例を図4に示す。図の斜線部分の面積Sが動的ノードの使用時間に相当する。ここでSをt_{start}～t_{req}間のノード使用に換算し、$n_s = n_i + \frac{S}{t_{req}-t_{start}}$を換算台数と呼ぶ。もし複数の動的台数方式がworkload1に対して同等のスループット性能を有しているならば、換算台数n_sが小さい方が優れた方式である。また、ある動的

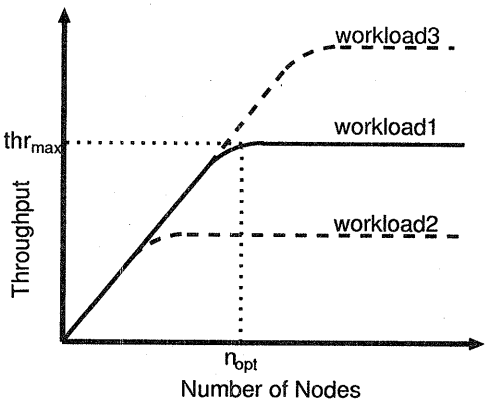

図 3　固定台数方式における台数とスループットの関係

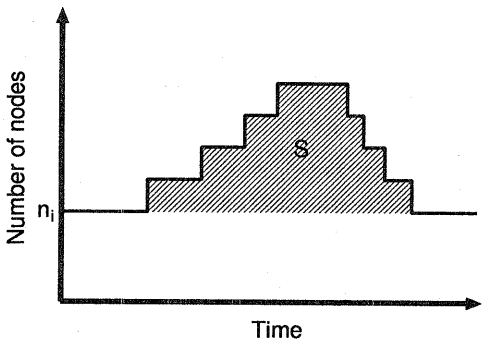

図 4　動的台数方式における時間とノード数の関係

台数方式が workload1 に対して thr_{max} と同等のスループット性能を持つ場合、換算台数 n_s を n_{opt} と比較することで固定台数方式との性能比較を行うことができる。

5. シミュレーションモデル

動的台数方式の突発的リクエスト増に対する性能を計測し、固定台数方式と比較するためにシミュレーション実験を行った。本節ではその際のシミュレーションモデルを述べる。

5.1 クラスタモデル

クラスタモデルの構成を図 5に示す。

通常ノードは 25 台存在し、この内の 1 台の上でサーバのフロントエンドプロセスが動く。さらに残りのノード群の一部の上でバックエンドプロセスが動く。各ノード上ではプロセスのメモリ使用量に関するシミュレートは行わない。クラスタ内には NFS サーバが 1 台存在し、HTTP クライアントがアクセスするすべてのコンテンツが納められている。HTTP サーバは通常ノードのディスクキャッシュがミスした場合は UDP 上の NFS/RPC プロトコルで NFS サーバにアクセスする。

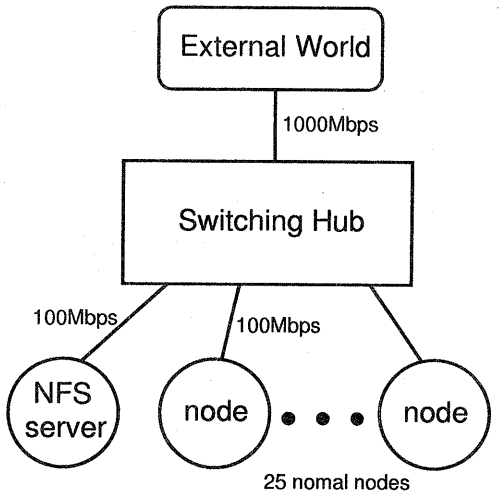

図 5　クラスタモデル

クラスタ内のノードはすべて 100Mbps でスイッチングハブに接続される。HTTP クライアント群を模擬する ExternalWorld とは 1000Mbps で接続され、この部分が性能のボトルネックにならないようにする。

通常ノードの OS はクラスタ内のすべてのマシンに対して 1 tick = 10 msec 毎に自分のノードの資源情報を UDP を使って送信する。"10 msec" は予備実験によりサーバ性能が最も向上する数値として選択した。

5.2 ワークロードモデル

HTTP ワークロードに関しては SPECweb99* のモデルを元に変更を加えている。ExternalWorld 上では HTTP クライアントを模擬した複数のプロセスがそれぞれ指定した時刻に立ち上がる。各クライアントは 5 ～ 15 個のコンテンツを順にサーバにリクエストする。全クライアントの 70% が HTTP/1.1 の persistent モードで動作する。

実験で用いる突発的なリクエスト増を模擬する 3 種類のワークロード workloadA, workloadB, workloadC を図 6,7に示す。workloadA を図 2と対応させると、$t_{start} = 1$, $t_{end} = 21$, $P_{high} = 45$, $P_{low} = 15$ となる。

サーバ上のコンテンツは静的コンテンツと動的コンテンツに大きく分かれる。アクセスパターンには偏りがあり、一番アクセスされるコンテンツに対して二番目のコンテンツへのアクセス頻度は 1/2、三番目は 1/3……という分布になる。静的コンテンツの大きさは 100 Byte から 900 KByte まであり、1 KByte ～ 9 KByte のコンテンツが最も頻繁にアクセスされる。動的コンテンツはサーバ上で 5msec ～ 500msec 実行された後、生成されたコンテンツをクライアントへ転送する。全リクエストの 10% は動的コンテンツに対するものである。

* http://www.spec.org/osg/web/

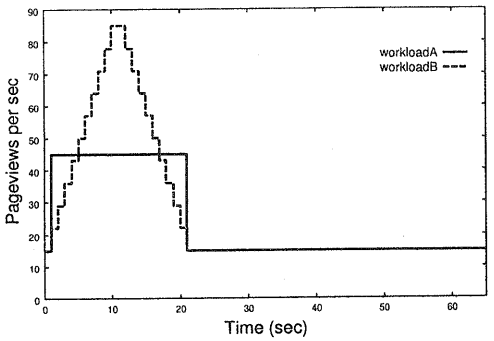

図 6　シミュレーションで用いるワークロードモデル (workloadA, workloadB)

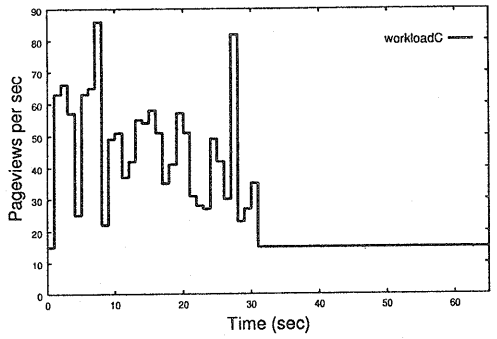

図 7　シミュレーションで用いるワークロードモデル (workloadC)

5.3 HTTP サーバモデル

サーバの実行時間は 400MHz Celeron 上の Linux2.2.14 と Apache1.3.9 の実行時間を計測した値を用いる。また、バックエンドノード上にサーバプロセスを遠隔実行させる場合、NFS サーバから 450KB のサーバプログラムをノード上にロードさせる必要がある。従ってフロントエンドが遠隔実行を決定してからサーバプロセスがバックエンドノード上で実際に動作を開始するまで 250msec ～ 400msec の遅延が生ずる。遠隔実行のためのプロトコルやその他のフロントエンド—バックエンド間の通信には UDP が用いられる。動的台数方式の場合、バックエンドサーバは初期状態で 3 台である。3 台とする理由は 6.1節で述べる。

6. 実験および考察

workloadA において $t_{req} = 35$ とした場合を例にとってシミュレーション結果を具体的に検証する。その後、3 種のワークロードに関して t_{req} を変化させて固定台数方式と動的台数方式の比較を行う。

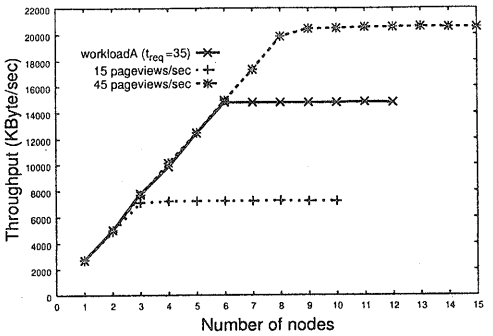

図 8　シミュレーション結果：固定台数方式における台数とスループットの関係

6.1 固定台数方式

固定台数方式の性能を評価する際の HTTP サーバモデルには、台数の動的な増減を行わない以外は第 3節で述べた HTTP サーバモデルと全く同じモデルを用いる。台数を変化させ、workloadA に対して $t_{req} = 35$ とした時のスループットを図 8に示す。また同時に、毎秒 $P_{low} = 15$ ページビューと毎秒 $P_{high} = 45$ ページビューのコンスタントなワークロードを処理した場合の台数とスループットの関係を示す。

このグラフから、固定台数方式を使って workloadA によるレスポンス時間の悪化を $t_{req} - t_{start} = 34$ 秒以内に抑えるためには、$n_{opt} = 6$ 台が最適値であることが分かる。また、その際のスループットは $thr_{max} = 14758$KB/sec である。

また、毎秒 15 ページビューのコンスタントなワークロードを処理するためにはバックエンドが 3 台あれば必要十分であることが分かる。このことから、動的台数方式のシミュレーションではバックエンドの初期台数を 3 台とする。

6.2 動的台数方式

動的台数方式の実験では、3.2節で述べたフロントエンドの台数増減判断の間隔 (interval) を 0.5 sec, 1.0 sec, 1.5 sec の 3 通り、また MaxCPULoad を 15 から 65 まで変化させる。

図 9は interval = 0.5 sec として MaxCPULoad = 25, 40, 55 の時の時間経過と台数の関係を表す。この図から、MaxCPULoad が小さいほどリクエストの増加に対して素早く対応して台数を増加させているが、動的ノードの使用時間も増加していることが分かる。従って必要十分な最適値を性能評価により求める必要がある。

また、MaxCPULoad が小さ過ぎると CPU 負荷の僅かな増加に対し敏感に反応して台数を増やすため、MaxCPULoad = 25 の場合に台数のピークが 2 つ生じている。

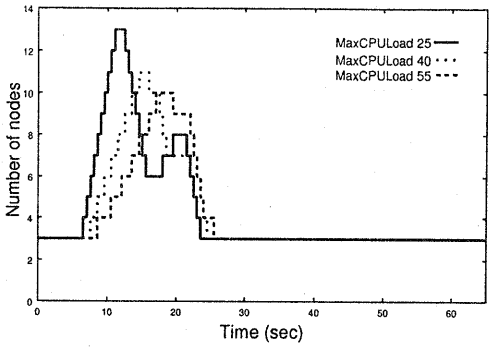

図 9　シミュレーション結果：動的台数方式における時間とノード数の関係

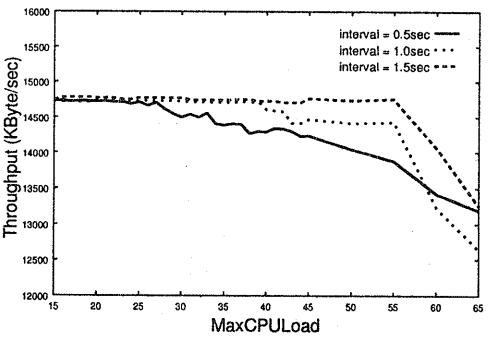

図 10　シミュレーション結果：MaxCPULoad とスループットの関係

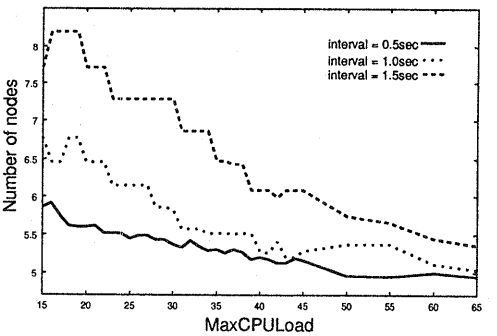

図 11　シミュレーション結果：MaxCPULoad と換算台数の関係

前節と同様に $t_{req} = 35$ とした時の workloadA に対するスループットを図 10に示す。また、$t_{req} = 35$ に対する換算台数を図 11に示す。

図 10より、MaxCPULoad をある値 Max_{opt} より小さくすると図 8の $thr_{max} = 14758$KB/sec に到達し、それ以上は性能が向上しなくなることが確認できる。

interval	Max_{opt}	換算台数
0.5	27	5.49
1.0	39	5.52
1.5	55	5.66

表 1　各 interval 値に対する MaxCPULoad の最適値と換算台数

interval = 0.5 sec, 1.0 sec, 1.5 sec に関してそれぞれ最適な MaxCPULoad の値 Max_{opt} を求め、図 11を使ってその場合の換算台数を求めると表 1のとおりとなる。この表から、workloadA によるレスポンス時間の悪化を $t_{req} - t_{start} = 34$ 秒以内に抑えるためには、interval = 0.5 sec, MaxCPULoad = 27 とした方式が最適であることが分かる。

ここで、最適な動的台数方式の換算台数は表 1より $n_s = 5.49$ 台である。この値は前節で求めた固定台数方式での最適台数 $n_{opt} = 6$ よりも小さい。このことから、workloadA, $t_{req} = 35$ に対して動的台数方式はパラメータを最適に設定すれば固定台数方式より優れた方式であると言える。

6.3　固定台数方式と動的台数方式の比較

workloadA に関して t_{req} を 25 から 45 まで変化させたときの固定台数方式における n_{opt}、及び動的台数方式における Max_{opt} に対する換算台数を図 12に示す。この図は、$t_{req} = 25$ において interval = 1.5 sec とした時の換算台数を除けば、動的台数方式は固定台数方式より優れた処理能力を持つことを意味する。

workloadB, workloadC に関して t_{req} を変化させたときの同様のグラフを図 13, 14に示す。図 13を見ると、t_{req} が小さい領域で interval=1.0sec, 1.5sec の場合の性能が悪い。これは台数増減判断の間隔が長すぎるとworkloadB の急激なリクエスト数増減に短時間で対処できないことを示す。

一方、図 14を見ると interval = 1.5 sec の場合の性能が他の 2 つに比べて優れている。これは workloadC の時刻 28 ～ 29 秒における 82 ページビューが原因である（図 7参照）。interval = 0.5 sec, 1.0 sec ではリクエストの増減に台数が敏感に反応するために、いったん減った台数が時刻 28 ～ 29 秒における負荷でまた増加に転じる。一方で interval = 1.5 sec では時刻 28 秒の時点で台数が他の 2 つに比べて減っていないために、時刻 28 ～ 29 秒における負荷に対して台数を増やすことなしに対処できる。

図 12, 13, 14を通して見ると、3 種のワークロードそれぞれに対して適切なパラメータを選択すれば、動的台数方式は最適な固定台数方式より優れた処理能力を持つと言える。以上のことは、突発的なリクエスト増の影響をある一定時間内に抑える能力のみに関する優劣を論じたものである。最初にも述べた通り、固定台数方式では予めノードの台数を増やしておかなければならない、という欠点を持つ。このことから、汎用クラスタ環境では

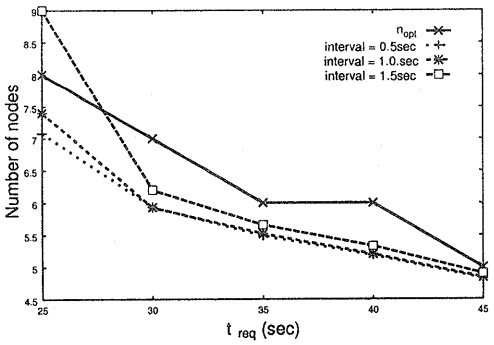

図 12 シミュレーション結果：t_{req} と最適な Max-CPULoad における換算台数の関係 (workloadA)

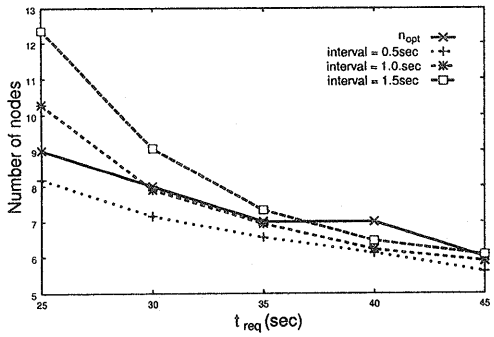

図 13 シミュレーション結果：t_{req} と最適な Max-CPULoad における換算台数の関係 (workloadB)

動的台数方式は突発的な負荷に対応するための有効な方法であると言える。

7. おわりに

汎用クラスタ上で CPU 負荷情報を利用して動的にノード数を増減させる HTTP サーバのモデルについて述べた。また、急激なリクエスト増に対するサーバモデルの処理性能を評価する基準を提案した。この評価基準ではレスポンス時間の悪化を指定時間内に抑えることができるかどうかに着目する。また、動的ノードの使用時間を台数に換算することで固定台数方式と性能の比較を行う。

さらに、台数を増減させる判断を起こす頻度と CPU 負荷の閾値をパラメータとしてシミュレーション実験で性能を評価した。実験結果から、台数増減の判断頻度が短いと突発的なリクエスト増に対応する能力が高いが、頻繁にリクエスト数が増減するワークロードに対しては性能が落ちる可能性があることが分かった。また、実験

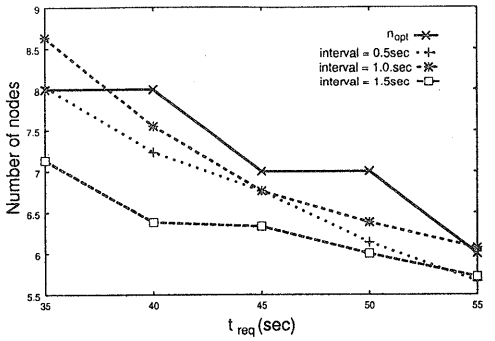

図 14 シミュレーション結果：t_{req} と最適な Max-CPULoad における換算台数の関係 (workloadC)

結果から動的台数方式の最適なパラメータを見つけだし、その時の処理性能は固定台数方式より優れたものであることが認められた。従って汎用クラスタ上の HTTP サーバに関して動的台数方式は突発的なリクエスト数の増加に対応できる有効な方法であることが確認された。

参考文献

1) Barak, A., La'adan, O. and Shiloh, A.: Scalable Cluster Computing with MOSIX for LINUX, *Proc. Linux Expo '99*, Raleigh, N.C., pp. 95–100 (1999).
2) Douglis, F. and Ousterhout, J.: Transparent Process Migration: Design Alternatives and the Sprite Implementation, *Software - Practice and Experience*, Vol. 21, No. 8 (1991).
3) Ghormley, D. P., Petrou, D., Rodrigues, S. H., Vahdat, A. M. and Anderson, T. E.: GLUnix: A Global Layer Unix for a Network of Workstations, *Software - Practice and Experience*, Vol. 28, pp. 929–961 (1998).
4) Litzkow, M., Livny, M. and Mutka, M. W.: Condor - A Hunter of Idle Workstations, *Proceedings of the 8th International Conference of Distributed Computing Systems*, pp. 104–111 (1988).
5) Mogul, J. C.: Operating Systems Support for Busy Internet Servers, Technical Note 49, DEC WRL (1996).
6) Ousterhout, J. K., Cherenson, A. R., Douglis, F., Nelson, M. N. and Welch, B. B.: The Sprite Network Operating System, *IEEE Computer*, Vol. 21, No. 2, pp. 23–36 (1988).
7) 松本尚ほか: 汎用スケーラブル OS SSS-CORE のカーネル構成について, 情報処理学会研究報告 98-OS-79 (SWoPP'98), Vol. 98, No. 71, pp. 53–60 (1998).