社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2006－ＤＳＭ－41（５）
2006／５／11

# 負荷変動を考慮したクラスタシステム向けの負荷分散アルゴリズムとノード追加方法

佐々木盛朗　田中淳裕

NECシステムプラットフォーム研究所 〒211-8666 川崎市中原区下沼部1753
E-mail: {s-sasaki@di, a-tanaka@dc}.jp.nec.com

**あらまし** 電子サービスを提供するシステムは、大部分のリクエストを要求されたレスポンスタイム内で処理するのに十分なキャパシティを持つべきである。しかし、クラスタシステムのキャパシティは、短期的な負荷変動によって容易に低下し、長期的な負荷変動によって不足する。短期的な負荷変動によるキャパシティ低下を緩和することを目的として、我々は仮想的な距離に基づいてリクエスト移送を行う負荷分散アルゴリズム、"Nearest Underloaded algorithm (N algorithm)"を提案・評価する。また、長期的な負荷変動に対応することを目的として、事前に見積もったキャパシティに基づいてノードを追加する方法を提案・評価する。

**キーワード** キャパシティ、負荷変動、負荷分散アルゴリズム、ノード追加

# A Load Balancing Algorithm and a Node Deployment for a Cluster System to Handle Workload Changes

Shigero SASAKI　and　Atsuhiro TANAKA

NEC System Platforms Research Laboratories　1753 Shimonumabe, Nakahara-ku, Kawasaki, 211-8666 Japan
E-mail: {s-sasaki@di, a-tanaka@dc}.jp.nec.com

**Abstract** Systems employed for E-services should have capacity enough to process requests within a short time. Capacity of a cluster system is easily reduced by short-term workload changes and is running short because of long-term changes. We propose and evaluate a load balancing algorithm, "Nearest Underloaded algorithm" (N algorithm), which transfers requests to other nodes in order of virtual node distance. The algorithm aimed at mitigating the reduction caused by the short-term changes. We also propose and evaluate a method of node deployment, which adds nodes based on capacities estimated in advance.

**Keyword** Capacity, Workload changes, Load balancing algorithm, Node deployment

## 1. はじめに

近年、ウェブサービスのような電子サービスを提供する基盤として、複数の（計算）ノードからなるクラスタシステムが広く用いられている。電子サービスのレスポンスタイムは、十分短いことが望ましい。なぜなら、レスポンスタイムが長いと、顧客がサービス終了を待てずに中断してしまい、機会損失が発生するからである。また、電子サービスではより多くの顧客にサービスを提供したいため、クラスタシステムのキャパシティは想定されるリクエスト到着率以上である必要がある。キャパシティとは、本稿ではほとんどのリクエストを十分短いレスポンスタイムで処理できるリクエスト到着率の最大値を指す。さらに、電子サービスはなるべく安価に提供されるべきである。つまり、クラスタをサービス基盤として用いる場合には、可能な限り、少数かつ安価なノードでクラスタを構築して、目標とするキャパシティを達成したい。

クラスタのキャパシティは、リクエストの平均処理時間とノード数に依存する。リクエスト処理のためにリソースが占有される時間は、待ち時間を除くと、平均処理時間とリクエスト到着率の積に等しい。この積をワークロード量と呼ぶ。平均処理時間が短ければワークロード量が、ノード数が多ければノードあたりのワークロード量が減少するので、キャパシティが拡大するのは明らかである。ただし、与えられた平均処理時間に対するノード数とキャパシティの関係は自明ではないことに注意する。なぜなら、負荷不均衡などのために処理の並列化が十分に進まないからである。

クラスタのキャパシティは、ワークロード特性と負荷分散アルゴリズムにも依存する。ワークロード特性とは、リクエスト到着のバースト性や個々のリクエスト処理時間の分散といった、ワークロード量には影響を及ぼさない、ワークロードを特徴づけるパラメータを指す。例えば、1分間の平均リクエスト到着率が600

である、といっても、0.1 秒毎に 1 リクエストずつ到着している場合と、ある特定の 0.1 秒間に 10 リクエスト到着している場合では、レスポンスタイムが全く異なる。さらに、発生する負荷の不均衡の度合いも異なるため、負荷分散アルゴリズムによって負荷を均衡させるまでにかかる時間が異なる。

負荷分散アルゴリズムは、短期的な負荷変動（ワークロード特性の変化）によらず、可能な限り大きいキャパシティを達成できるように設計されるべきである。つまり、到着のバースト性が高いなど、負荷不均衡が拡大しやすい状態でも、短時間で負荷を均衡させるようなアルゴリズムが望ましい。また、クラスタが多数のノードによって構成されているときでも、負荷を均衡させるべきである。そのようなアルゴリズムを採用することで、目標キャパシティを達成するためのノード数の削減または安価なノードへの切り替えが可能になり、システムコストを低減できる。

我々は負荷分散アルゴリズム、Nearest Underloaded algorithm (N algorithm)を提案・評価する。N algorithm はノード間に設けた仮想的な距離に基づいてリクエストを移送し、高負荷ノードは、自ノードよりも近い高負荷ノードからの負荷移送を受けたあとも低負荷でいるノードにのみ負荷を移送する。評価では、N algorithm と既存のアルゴリズムを用いて、8 ノードから 16 ノードまでを含むクラスタシステムのキャパシティを測定した。測定では 3 種類のワークロードを用いた。評価結果は、N algorithm は既存のアルゴリズムよりも大きいキャパシティを達成できる、または、より少ないノードで同等のキャパシティを達成できることを示した。

電子サービスのワークロード量は、時間とともに変化する。例えば、顧客数が 1 万から 10 万に増加すれば、ワークロード量も 10 倍になり、達成すべきキャパシティも 10 倍になる。このようなワークロード量の変化を長期的な負荷変動と呼ぶ。ワークロード特性はごく短い時間に変化するのに対して、ワークロード量は比較的長い時間をかけて変化するためである。

我々は、長期的な負荷変動に対応するためのノード追加方法も提案・評価する。提案方法は、リクエスト到着率の増加速度に上限を設定し、その範囲内でキャパシティを満たすようにノードを追加する。ただし、負荷均衡が適切に計られているか、ノードあたりのキャパシティに十分な余裕があると仮定して、クラスタのキャパシティはノードあたりのキャパシティとノード数の積であるとした。評価では、電子商取引を模したワークロードをかけ、ワークロード量が増加してもレスポンスタイムが悪化しないことを示した。

## 2. 課題と性能指標

本稿の課題は、第一に、短期的な負荷変動（ワークロード特性の変化）が発生しても高いキャパシティを達成することであり、第二に、長期的な負荷変動（ワークロード量の変化）に対応して、クラスタのキャパシティを拡大することである。

第一の課題に対しては、様々な負荷不均衡を高速に均衡させる、というアプローチをとる。負荷の不均衡はリクエストの到着と出発によって生じるが、我々は特に出発による不均衡に着目している。到着による不均衡は、最も負荷の低いノードにリクエストを振り分けることで十分に解消できる。しかし、リクエスト振分では出発による不均衡を迅速に解消することは一般に困難であり、また、高負荷ノードに到着したリクエストのレスポンスタイムが改善されることはない。

リクエストの出発による不均衡を解消する方法の一つは、高負荷ノードのリクエストを低負荷ノードに移送することである。既にノードに到着したリクエストを移送するため、移送の判断は個々のノードが独立に行うことが多い。そのため、多数の高負荷ノードからの少数の低負荷ノードへのリクエストが起きることで、低負荷ノードが高負荷になってしまって負荷が均衡しない場合がある。これは群集効果[2]と呼ばれ、不適切なリクエスト移送の一例である。

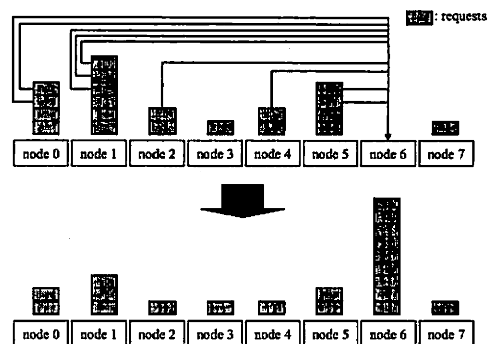


**図 1 群集効果**

図 1は、自ノードとリクエスト数が最小であるノードのリクエストの数の差の半分を移送するリクエスト移送アルゴリズムを用いた場合の群集効果の例である。一度に移送するリクエストの数を減らすことで群集効果を緩和することができるが、その場合、負荷均衡状態に遷移するまでに何回かのアルゴリズムの適用を要求するため、負荷の状態によっては均衡状態に遷移するまでに長い時間がかかってしまう。キャパシティという観点から見ると、遷移の遅い負荷移送は、群集効果と同様に不適切な負荷移送である。望ましいのは高速に負荷均衡状態に遷移させるアルゴリズムである。

第二の課題に対しては、必要十分な数のノードを追

加する、というアプローチをとる。長期的な負荷変動の速度（リクエスト到着率の増加率）に上限を設定した上で、ノード数に対するキャパシティと到着率の検出からノード追加の終了までにかかる時間が既知であれば、追加終了時点で達成しているべきキャパシティが事前にわかる。よって、追加すべきノード数も事前にわかる。

我々の目標は、ほとんどのリクエストを十分短いレスポンスタイムで処理することである。レスポンスタイムにも色々あるが、第一の課題に関しては、特に *x*th Percentile Response Time (*x*th PRT)を用いる。一部のリクエスト、5%かもしれないし、1%か 0.5%かもしれない、は状況次第ではレスポンスタイムが長くても許容されるが、平均レスポンスタイムを用いた場合、レスポンスタイムがある値以上になるリクエストの割合はわからないからである。そこで *x*th PRT に基づいてキャパシティを算出する。このキャパシティを *x*th Percentile Capacity (*x*th PC)と呼ぶ。

## 3. 負荷分散アルゴリズム

本節では、既存のアルゴリズムと我々がシステムにおいた仮定について述べる。そして、提案アルゴリズムである Nearest Underloaded algorithm (N algorithm)について述べる。

### 3.1. 負荷分散アルゴリズムの分類

ここでは、負荷分散アルゴリズムの差異を明らかにするために、アルゴリズムを下記の 5 つのポリシーに分解した上で、既存のポリシーについて説明する[3][4]。

1. Initiation policy: 負荷分散を行う主体は何か
2. Activation policy: いつ負荷分散アルゴリズムが起動されるか
3. Information policy: 負荷指標は何で、いつ更新されるのか
4. Transfer policy: どのリクエストを移送するか
5. Placement policy: どのノードの負荷情報を入手し、どのノードでリクエストを処理するのか

Initiation policy は大きく集中ポリシーと分散ポリシーに分けられる[5]。集中ポリシーの下では、一つのノードやロードバランサーによってリクエスト振分が行われる。分散ポリシーの下では、各ノードがリクエスト移送を行う。分散ポリシーはさらに移送元主導ポリシーと移送先主導ポリシーに分類される[6]。しかし本稿では、以降は移送元主導ポリシーのみをスコープに含める。両者の差は本稿では本質的ではないためである。Activation policy は、タイムドリブンポリシーまたはイベントドリブンポリシーのいずれかになる。タイムドリブンポリシーは周期的なアルゴリズムの起動を意味することがほとんどであり、イベントドリブンポリシーは、負荷の変化によってアルゴリズムを起動することが多い。Information policy において、負荷指標としてよく用いられるのはリクエストのキュー長である[7]。CPU または他のリソースの使用率もよく用いられる。負荷指標の更新は activation policy で示した方法が用いられる。

Transfer policy は、全てのリクエストを等価だとみなした場合、いくつのリクエストを移送するかを決定するポリシーである。移送するリクエストの数は、移送元ノードと移送先ノードの負荷の差に基づいて経験的に決定されるのが一般的である[8][9]。リクエストは等価ではないとみなし、リクエストをプリエンプティブに移送する場合に関しては[10]に示されるポリシーがあるが、本稿では非プリエンプティブ移送のみを扱い、リクエストは等価であるとみなす。多くの placement policy はノードスコープ内で最も負荷の低いノードを移送先として選択する。ノードスコープは、移送元ノードが負荷情報を共有しているノードの集合を示す。ノードスコープの例としては、[11]等のようにランダムに選択されたノードの集合がある。[12]では、各ノードがランダムに選択したノードの負荷を取得し、自ノードが高負荷でランダム選択されたノードが低負荷であれば、そのノードをスコープに含める。逆に、低負荷ノードは高負荷ノードをスコープに含める。

### 3.2. システムにおいた仮定

我々がターゲットとするクラスタシステムは、同種のステートを持たないノードから構成される。例えば、ウェブサーバや科学技術計算用途のサーバからなるクラスタシステムがそれに該当する。より詳細には、以下の仮定が成り立つシステムがターゲットである。

1. クラスタシステムは等価な（計算）ノードからなり、ノードがボトルネックになる
2. 単一のサーバーアプリケーションが動作する
3. リクエスト移送のオーバーヘッドは小さい
4. サーバーアプリケーションは状態を持たない
5. リクエスト間に依存性はない
6. 同時に処理されるリクエストは m(≧1)個以下

ウェブサーバはステートを持たず、クッキーが全てのノードで同じ方法で処理される限りにおいては、ウェブリクエスト間には依存性はないことに注意する。仮定 5 は、ノードに $k$ 個のリクエストがある場合、$(k-m)$リクエストが移送可能であることを意味する。また、ウェブリクエストは単なる短いテキストであり、その移送には限られた CPU 時間とネットワーク帯域しか消費されない。

### 3.3. Nearest Underloaded Algorithm

ここでは、N algorithm の transfer policy と placement policy について詳細に述べる。N algorithm の transfer policy では、移送されるリクエストの数は、各ノード

が持っているリクエストの平均値と移送元ノードが持っているリクエストの数の差である。そして、平均よりも多い数のリクエストを持つノードは高負荷ノードとみなし、平均よりも少ない数のリクエストを持つノードを低負荷ノードとみなす。N algorithm の placement policy は、ノード間にもうける仮想的な距離に基づいて、高負荷ノードが近いノードから順にリクエストの移送数を決定していく。現在のターゲットノードが低負荷であれば、ターゲットノードを高負荷にしない範囲で負荷を移送する。高負荷ノードが高負荷でなくなった時点でそのノードからの負荷移送は終了する。

N algorithm の下では、ノードは絶対ノード ID と相対ノード ID とを持つ。クラスタを構成するノードの数を $n$ として、ノードに 0 から$(n-1)$までの数字を割り当て、これを絶対ノード ID とする。相対ノード ID は、二つのノードの絶対ノード ID の差で表される。ノード $i$ から見たノード $j$ の相対ノード ID は$(i-j+n)\%n$ で与える。例えばノード数が 8 であると仮定すると、ノード 3 から見たノード 5 の相対ノード ID は 2 であり、ノード 7 から見たノード 5 の相対ノード ID は 6 である。

N algorithm のアルゴリズムを以下に示す。負荷指標は整数または実数を想定しているが、いずれにせよ本質的な違いはないため、負荷指標は整数であることを仮定する。以下のステップでは、ノード ID は相対ノード ID であることに注意する。

1. 負荷の平均値 $m$ を算出する
2. 自ノードの負荷と *ceil*($m$)の差を *over* とする
3. *over* が正でなければアルゴリズムを終了する
4. ターゲットノードの相対ノード ID を表す $t$ を 1 で、ターゲットノードへの負荷の移送量を表す under を 0 で初期化する。
5. $m$ とノード $t$ の負荷の差を *under* に加える
6. *floor*(*under*)が正であれば、*floor*(*under*)と *over* の小さい方の分の負荷をノード $t$ に移送する
7. *over* が *floor*(*under*)よりも小さければ、アルゴリズムを終了する
8. *over* と *under* の両方から、*floor*(*under*)を引き、$t$ をインクリメントする。$t$ が $n$ になったのであれば、アルゴリズムを終了する
9. ステップ 5 に戻る

負荷を均衡させるのに不必要な負荷移送をさけるため、高負荷ノードは平均以上である負荷の分のみを移送する。N algorithm においては、ノードの過負荷部分は *over* で表現される。各高負荷ノードは、過負荷部分を移送するため、これらのノードへの負荷移送が起きなければ、負荷移送後に高負荷ではなくなる。一方、低負荷ノードへの過度な負荷移送が起こると、低負荷ノードを高負荷にしてしまうため、負荷が均衡しない。これを防ぐために、N algorithm ではより近いノードからの負荷移送を優先する。*under* は自ノードよりも近いノードからの負荷移送が発生した後においてターゲットノードにどれ位の負荷を移送しても高負荷にならないかを表す。

N algorithm が各ノードで同時に起動された場合、負荷指標、つまりリクエスト数は平均化される。現在の負荷指標は時間的な遅れのために不正確であり、起動の同時性は保証されないが、これらの問題は本稿のスコープ外とし、情報の正確さと起動の同時性については、キャパシティの観点からのみ扱う。

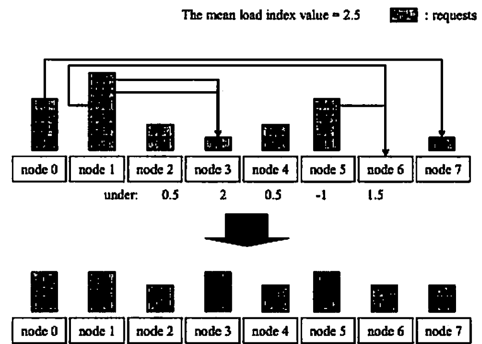


図 2 N algorithm の下での負荷状態の遷移

図 2に、N algorithm の下での負荷均衡状態への遷移の動作例を示す。図 2は、8 ノードクラスタ上に 20 個のリクエストがある場合の例であり、ノードには絶対ノード ID が振られている。リクエスト数を負荷指標としているので、負荷指標の平均値は 2.5 である。ノード 2 から 6 の下に示された値は、ノード 1 がこれらのノードをターゲットノードとした時の under の値である。

## 4. キャパシティの評価

本節では、N algorithm を用いることで xth PC を増大させられること及びより少数のノードで目標とする xth PC を達成できることを示す。

### 4.1. 評価環境

評価は、ギガビットイーサネットで接続された 16 台のノードからなるクラスタ上で行った。ノードの CPU は Xeon 2.4GHz であり、4GB のメモリ容量を持つ。ノードの OS は Linux2.4.7 を使用した。各ノードでは、我々が実装した負荷分散デーモンである Server Wrapper Daemon (SWD)とダミーアプリケーションを動作させた。さらに、簡単な負荷発生プログラムを実装し、これによって負荷を発生させた。以下、実装したソフトウェアと合成したワークロード、評価対象と

した負荷分散アルゴリズムについて述べる。

### 4.1.1. 評価用ソフトウェア

SWD はリクエスト移送を可能にするデーモンであり、クラスタの各ノード上で動作し、クライアントからアプリケーションへのリクエストをインターセプトする。インターセプトされたリクエストはアプリケーションまたは他ノードで動作する SWD に移送される。SWD はアプリケーションまたは他ノードの SWD からリプライを受け取ると、クライアントまたは移送元ノードで動作する SWD にリプライを転送する。SWD は与えられた負荷分散アルゴリズムにしたがってリクエストの移送先を決定する。SWD はリクエストのキューを持ち、キューの先頭のリクエストをアプリケーションに移送し、キューの最後尾のリクエストを他ノードで動作する SWD に移送する。同時にアプリケーションに転送するリクエストの数は、本評価では 1 とした。ただし、処理中のリクエストは移送できないことに注意する。負荷移送の機能に加えて、SWD は負荷情報を他の SWD と共有する機能を持っている。本評価での負荷情報とはリクエスト数を指す。

負荷発生プログラムは HTTP リクエストを発行し、リプライを受け取る。負荷発生プログラムはリクエスト振分機能を併せ持つ。ダミーアプリケーションは、HTTP リクエストを受け取り、リクエストに指定された回数だけビジーループをまわり、リプライを返す。

### 4.1.2. ワークロード

本評価では、平均リクエストサイズが 40ms である三つの合成ワークロードを用いた。平均リクエストサイズは、fine-grain trace と medium-grain trace の間の大きさである。第一のワークロードを標準ワークロードと呼ぶ。標準ワークロードにおいては、リクエストはポワソン到着し、リクエストサイズは対数正規分布に従う。この分布の標準偏差と平均リクエストサイズの比は、[14]のファイルサイズの分布にならって 2.16:1 にした。第二のワークロードを分散ワークロードと呼ぶ。分散ワークロードと標準ワークロードでは、リクエストサイズの標準偏差が異なり、2.16 のルート 2 倍の 3.06 になっている。第三のワークロードはバーストワークロードと呼ぶ。バーストワークロードと分散ワークロードの差異は、バーストワークロードでは一度に 4 つのリクエストが発生する点である。

### 4.1.3. 負荷分散アルゴリズム

本評価で対象とする負荷分散アルゴリズムは、ラウンドロビン(RR)、集中型(CT)、最低負荷(LL)、N algorithm (N)の 4 つである。これらのアルゴリズムのポリシーを表 1に示す。Information policy の上段には負荷指標を、下段には情報の更新方法が示した。CT の下では負荷情報はリクエストの到着または出発が起こったときに負荷情報が更新される。また、Transfer policy の上段にはどのリクエストが移送され、下段にはいくつのリクエストが移送されるかが示されている。本稿においては、LL は自ノードと最低負荷のノードのキュー長の差の半分未満のリクエストを移送する。LL と N においては、リクエストはラウンドロビンで各ノードに到着する。

表 1 評価対象の負荷分散アルゴリズム

<table>
<tr><th></th><th>RR</th><th>CT</th><th>LL</th><th>N</th></tr>
<tr><td>initiation policy</td><td colspan="2">centralized</td><td colspan="2">distributed</td></tr>
<tr><td>activation policy</td><td colspan="2">event-driven (on arrival)</td><td colspan="2">time-driven (each 40ms)</td></tr>
<tr><td rowspan="2">information policy<br>(index & update)</td><td>N/A</td><td colspan="3">the length of the request queue</td></tr>
<tr><td>N/A</td><td>event-driven</td><td colspan="2">time-driven (each 20ms)</td></tr>
<tr><td rowspan="2">transfer policy<br>(which & how many)</td><td colspan="2" rowspan="2">N/A</td><td colspan="2">the request at the tail of the queue</td></tr>
<tr><td>difference based</td><td rowspan="2">described in Section 3.3</td></tr>
<tr><td>placement policy</td><td>round robin</td><td>least-loaded</td><td>least-loaded</td></tr>
</table>

## 4.2. xth percentile capacity の測定

ここでは、CT, LL, N の下でのクラスタシステムの xth PC について述べる。我々が実際に測定したのは、8～16 ノードのクラスタ上でリクエスト到着率を変化させたときのレスポンスタイムである。このレスポンスタイムから xth PRT を算出し、隣接する測定点の間においては、リクエスト到着率と xth PRT の間に比例関係が成り立つとして xth PC を算出した。以下の評価結果は、CT, LL, N のアルゴリズムの下で 3 つの合成ワークロードに対する xth PC を求めたものである。xth PRT の上限は、16 ノードクラスタに N algorithm を適用して、320 req/sec の負荷をかけたときの xth PRT に設定した。

図 3は 99.5th PRT の上限を 643ms としたときの標準ワークロードに対する 99.5th PC である。図 4は、99.5th PRT の上限を 900ms としたときの分散ワークロードに対する 99.5th PC、図 5は 99.5th PRT の上限を 927ms としたときのバーストワークロードに対する 99.5th PC である。LL を用いたときの 16 ノードでの 99.5th PC がプロットされていないのは、LL では対応する負荷に耐えられなかったためである。

標準ワークロードに対する 99.5th PC は、どの負荷分散アルゴリズムを適用しても大きくは変わらないが、分散ワークロードに対する 99.5th PC はアルゴリズムによって大きく変わる。リクエストサイズのばらつきが大きければ、出発による不均衡が発生しやすくなるため、これを解決しない CT の 99.5th PC は大きく縮小する。また、ノードの負荷が高い（キュー長が長い）ときには、負荷不均衡の度合いが大きく、LL は安定して高速に負荷均衡状態に遷移しない。これが、負荷の増大とともに N と LL の差が顕在化する理由である。

図 5では、N の下での 16 ノードクラスタの 99.5th PC は CT のそれに比べて 44.6%大きいことが見て取れる。また、16 ノードクラスタで CT を用いたときの 99.5th

PCは、12ノードクラスタでNを用いたときの99.5th PCにほぼ等しい。つまり、Nを用いることで、25%のノード数削減が可能であることがわかる。

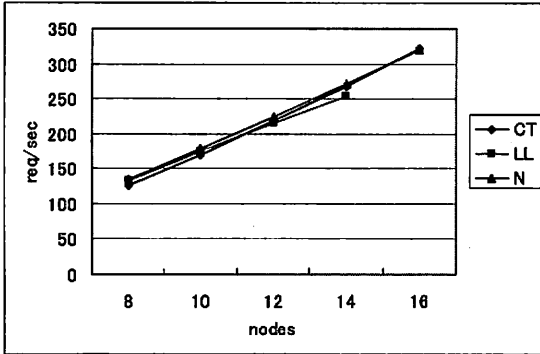


図 3 99.5th PC（標準ワークロード）

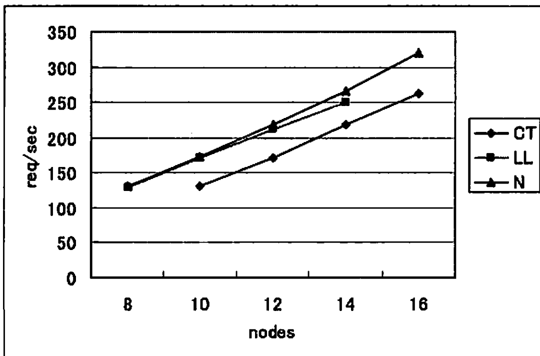


図 4 99.5th PC（分散ワークロード）

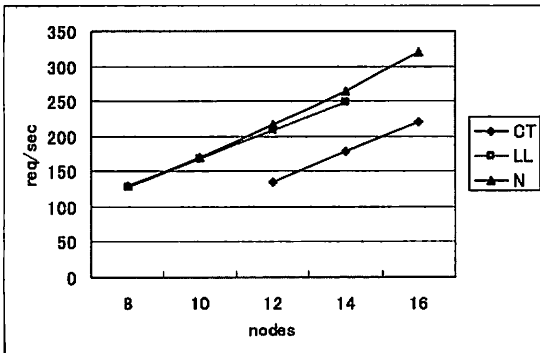


図 5 99.5th PC（バーストワークロード）

### 4.3. 考察

リクエスト移送を行う場合と行わない場合で、また、どのようにリクエストを移送するかでキャパシティが異なるのは、出発による負荷不均衡が生じていることを意味する。リクエストを振り分けているのに負荷不均衡が発生する原因は二つある。一つは、リクエスト処理時間のばらつきである。ばらつきが大きいほど、あるノードに到着したリクエストと他のノードに到着したリクエスト処理の終了時間の差が大きくなり、負荷の不均衡が発生しやすくなる。もう一つの原因は、リクエストのバースト到着である。徐々にリクエストが到着する場合には、振り分けたリクエストの終了を確認しながらリクエストの振分先を決定できる。しかし、到着のバースト性が高いほど多くのリクエストを短い時間で振り分けなければならない。キューにリクエストが多く滞留している状態では、出発による負荷不均衡が大きくなりやすい。

標準ワークロードでは、リクエスト処理時間のばらつきが小さいため、出発による不均衡が発生しにくい。そのため、CT, LL, Nの間でのキャパシティの差は小さい。一方、リクエスト処理時間のばらつきが大きい分散ワークロードでは、出発による負荷不均衡が発生するため、リクエスト移送を行わないCTのキャパシティは、リクエスト移送を行うLLとNに比べて小さくなる。高負荷になるほどNがLLより大きいキャパシティを達成できるのは、高負荷なほどキュー長は長くなり、負荷を均衡させるのが困難だからである。Nは不均衡が大きい状態からでも、高速に均衡状態に遷移するため、リソースを有効に活用できる。さらに、リクエスト到着のバースト性が高いバーストワークロードに対しては、CTのキャパシティの小ささが際立つ。これは、バースト到着による不均衡度合いが拡大したためである。

## 5. ノード追加

本節では、提案するノード追加方法について説明し、評価で用いるワークロードであるTPC-W[15]と評価環境について述べる。そして、提案方法を用いることで、負荷が増大する場合でも必要十分なキャパシティを達成し、レスポンスタイムを低く抑えられることを示す。

### 5.1. ノード追加方法

提案ノード追加方法は、現在のリクエスト到着率(*arrival*)、リクエスト到着率の最大増加率 (increase)、リクエスト到着率のサンプリング間隔 (*interval*)、ノード追加時間 (*delay*)、ノードあたりのキャパシティ(*nodecap*)を入力としてとる。そして、現在のノード数 *cn* が

$$fn = (arrival + (interval + delay) * increase) / nodecap$$

以上であれば、想定する範囲で負荷が増加してもリクエスト到着率がキャパシティ以下であるので、レスポンスタイムは上限を超えることはない。したがって、*cn* が *fn* 以下であればノードは追加せず、*cn* が *fn* よりも小さいときは、ceil(*fn* − *cn*)台のノードを追加する。

上式では、ノードあたりのキャパシティがノード数によらず一定であることを想定して *fn* を算出している。そのため、同じデータを複数ノードで操作する場合に生じるオーバーヘッドや同期待ち時間は考慮していない。また、ワークロード量はリクエスト到着率のみで測られており、平均リクエスト処理時間が一定であることを暗黙に仮定している。本稿では、ノード数の変化に伴うノードあたりのキャパシティの変化と平均リクエスト処理時間の変化はスコープには含まない。

ノードあたりのキャパシティはワークロード特性にも依存する。ワークロード特性を考慮しないのは、想定する最小キャパシティを用いるためである。ワークロード特性によるキャパシティの縮小は、負荷分散によって最小限に抑えられることを仮定している。

### 5.2. TPC-W

ノード追加アルゴリズムの評価に用いる TPC-W は、電子商取引システムを対象としたベンチマークの仕様である。TPC-W は、電子商取引サイトである SUT (System Under Test)、電子商取引のユーザである RBE (Remote Browser Emulator)、認証サーバである PGE (Payment Gateway Emulator)によって構成される。RBE はユーザからのリクエストをエミュレートして負荷を発生させ、PGE はリクエストの処理に伴って発生する外部への認証要求を処理し、SUT は RBE からのリクエストを処理する。

TPC-W において RBE が発行するリクエストは 14 種類に分類されており、各種リクエストの構成比によって異なる負荷をかけられる。TPC-W では主要な負荷パターンである「Shopping mix」と補助的な負荷パターンである「Browsing mix」、「Ordering mix」が定義されており、対象とするサービス内容に合わせて負荷パターンを選択できる。また、リクエストに対して SUT が扱うコンテンツは、商品のサーチなど、データの書き換えを必要としない「静的コンテンツ」と、商品売買に伴うトランザクション処理を行うための「動的コンテンツ」に分類される。

### 5.3. 評価環境

ノード追加アルゴリズムの評価環境は、図 6の通り。評価環境は TPC-W に沿って構築した。RBE を動作させるノードは 10 台である。リクエストはリクエスト振分を行うプロセスである glbd が動作するノードに対して発行される。glbd はリクエストのヘッダーを見て、それが静的コンテンツか動的コンテンツへのリクエストなのかを判別し、いずれかの swd にリクエストを転送する。静的コンテンツへのリクエストは、apache が動作する、静的グループに属するノードの swd に転送され、動的コンテンツへのリクエストは、PHP モジュールを組み込んだ apache が動作する、動的グループに属するノードの swd に転送する。各グループのノード数は可変であり、グループ内のリクエスト振分先ノードはラウンドロビンで選択される。動的コンテンツの処理はデータベースアクセスを伴うため、適宜 postgreSQL が動作する 4 台ノードと通信する。どのノードにリクエストを発行するかは、ユーザ ID で振り分ける。

glbd は過去 3 秒間の平均リクエスト到着率と平均レスポンスタイムをグループ毎に測定する。グループごとの到着率が閾値を越えると、適切な数のノードを追加する。ノード追加にかかる時間は約 0.8 秒である。また、静的グループに関しては、ノードあたりのキャパシティは 200 リクエスト／秒に、動的グループに関しては 12 リクエスト／秒にした。これらのキャパシティは事前評価により求めた。

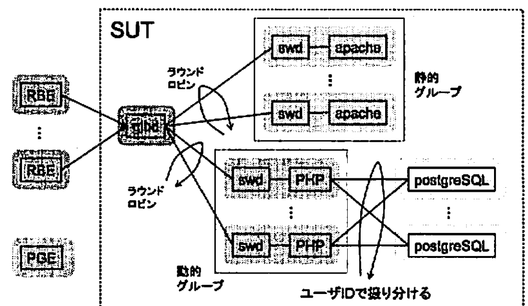


図 6 TPC-W 評価環境

### 5.4. 評価

ノードを追加する目的は、ワークロード量が増大してもレスポンスタイムを悪化させないことである。そこで、TPC-W のユーザ数を秒 10 ユーザ増やしつつ、3 秒おきにリクエスト到着率と平均レスポンスタイムを測定した。

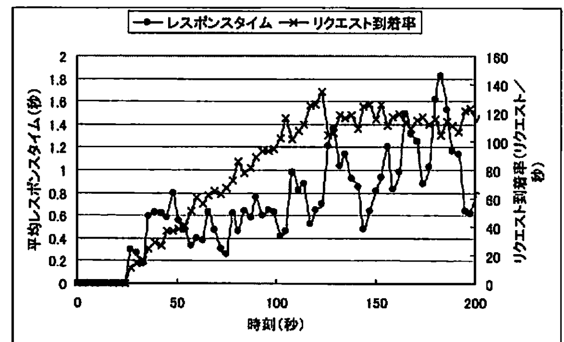


図 7 リクエスト到着率と平均レスポンスタイムの関係

図 7にユーザ数を 1000 まで増加させたときのレス

ポンスタイムを示す。許容されるレスポンスタイムの上限を 3 秒とし[16]、レスポンスタイムが 3 秒を越さないようにノードを追加した。適切にノードが追加されているために、リクエスト到着率が増加しているにも関わらず、レスポンスタイムが短いままであることが図 7からわかる。一方、図 8はリクエスト到着率とノード数の関係を示す。リクエスト到着率の増加に伴ってノード追加が行われていることがわかる。

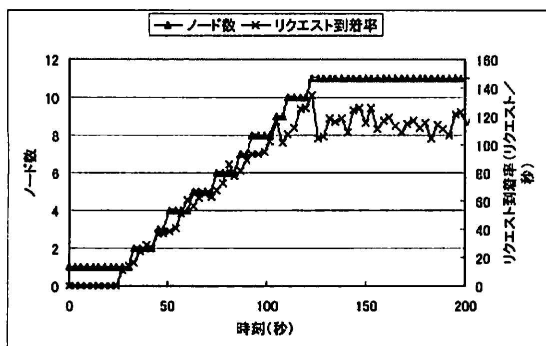


**図 8 リクエスト到着率とノード数の関係**

## 6. おわりに

クラスタシステムのキャパシティは、短期的な負荷変動によって容易に低下し、長期的な負荷変動によって不足する。短期的な負荷変動によるキャパシティ低下を緩和するため、我々は仮想的な距離に基づいてリクエスト移送を行う負荷分散アルゴリズム、" Nearest Underloaded algorithm (N algorithm)"を提案・評価した。評価結果では、N algorithm を 16 ノードのクラスタに適用すると、他のアルゴリズムを適用した場合に比べて 25%少ないノード数で同じキャパシティを達成できる例を示した。長期的な負荷変動に対しては、負荷増大検出とノード追加の時間遅れを考慮して、事前に見積もったキャパシティに基づいてノードを追加することで、レスポンスタイムを 3 秒以下に保つ例を示した。

## 謝辞

本研究は、新エネルギー・産業技術総合開発機構、基盤技術研究促進事業、「大規模・高信頼サーバの研究」の支援の下に行われた。

## 文献

[1] D.A. Menasce and V.A.F Almeida, Capacity Planning for Web Services, Prentice Hall PTR, 2002.

[2] M. Mitzenmacher, How Useful Is Old Information? *IEEE Trans. on Parallel and Distributed Systems*, Vol. 11, No. 1, pp. 6—20, January 2000.

[3] D. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou. Process Migration. *ACM Computing Surveys*, Vol. 32, No. 3, pp. 241—299, September 2000

[4] V. Cardellini, E. Casalicchio, and M. Colajanni. The State of the Art in Locally Distributed Web-server Systems. *ACM Computing Surveys*, Vol. 34, No. 2, pp. 263—311, June 2002.

[5] S. Zhou. A Trace-driven Simulation Study of Dynamic Load Balancing. *IEEE Trans. on Software Engineering*, Vol. 14, No. 8, pp. 1327-1341, September 1988.

[6] D. Eager, E. Lazowska, and J. Zahorjan. A Comparison of Receiver-initiated and Sender-initiated Adaptive Load Sharing. *Performance Evaluation*, Vol. 6, No. 1, pp. 53—68, May, 1986.

[7] D. Ferrari and S. Zhou. A Load Index for Dynamic Load Balancing. Proceedings of *the Fall Joint Computer Conference*, pp. 684—690, November 1986.

[8] J. Stankovic, Simulations of Three Adaptive, Decentralized Controlled Job Scheduling Algorithms, *Computer Networks*, Vol. 8, pp. 199—217, August 1984.

[9] D. Eager, E. Lazowska, and J. Zahorjan. Adaptive Load Sharing in Homogeneous Distributed Systems. *IEEE Transactions on Software Engineering*, Vol. 12, No. 5, pp. 662—675, June 1986.

[10] M. Harchol-Balter and A. Downey, Exploiting Process Lifetime Distributions for Dynamic Load Balancing, *ACM Transactions on Computer Systems*, Vol. 15, No. 3, pp. 253—285, 1997.

[11] M. Dahlin. Interpreting Stale Load Information. *IEEE Transaction on Parallel and Distributed Systems*, Vol. 11, No. 10, October 2000.

[12] Kremien, J. Kramer, and J. Magee. Scalable, Adaptive Load Sharing for Distributed Systems. *IEEE Parallel and Distributed Technology*, Vol. 1, No. 3, pp. 62—70, August 1993.

[13] K. Shen, T. Yang, and L. Chu. Cluster Load Balancing for Fine-grain Network Services. In *Proceedings of the International Parallel and Distributed Processing Symposium*, pp. 51—59, April, 2002.

[14] P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation, In *Proceedings of Performance'98/SIGMETRICS'98*, pp. 151—161, July 1998.

[15] TPC Benchmark W Standard Specification Ver. 1.8, http://www.tpc.org/tpcw/spec/tpcw_V1.8.pdf, Feb 2002

[16] How Fast Does a Website Need To Be, http://www.perftestplus.com/resources/how_fast.pdf