2007－OS－106 (1)
2007／8／3

柔軟な負荷分散を可能にする分散型シングルIPクラスタ

藤田 肇[†] 松葉浩也[††] 石川 裕[†,††]

分散型シングル IP クラスタは負荷分散と耐故障性の両方を目標に設計されているが，従来の分散型シングル IP クラスタではクライアントの情報に基づいた静的な接続割り当てにしか対応しておらず，状況に応じた動的な負荷分散が行えない．そこで我々は分散型シングル IP クラスタを拡張した TCP-DSC (TCP for Distributed Servers with Centralized scheduler) を提案する．TCP-DSC は分散型シングル IP クラスタに TCP コネクションの割り当てを指示するマスタノードを導入し，柔軟な負荷分散を可能にするものである．HTTP サーバを用いた実験の結果，CGI ページへのアクセスにおいて TCP-DSC は従来の静的な割り当て手法と比較して約 50%の平均応答時間を達成することが示された．

Flexible Load Balancing on Distributed Single IP Clusers

HAJIME FUJITA,[†] HIROYA MATSUBA[††] and YUTAKA ISHIKAWA[†,††]

Distributed single IP clusters are designed toward both fault-tolerance and performance scalability. However, the static traffic assignment method, which employed by the existing designs and implementations, does not scale as centralized load balancing systems do. In this article, we propose a new load balancing method, named TCP-DSC (TCP for Distributed Servers with Centralized scheduler), to improve performance scalability. TCP-DSC introduces one dispatcher node which dispatches TCP connections to other nodes, so that it can balance workloads among nodes. Experimental results with HTTP server show that the method proposed has achieved 50% less response time than existing method in accessing CGI-generated pages.

1. はじめに

インターネットの普及に伴って，今日のサーバには非常に高い処理能力と信頼性が求められるようになっている[8]．サーバの性能と信頼性を向上させるためには物理的に複数のサーバを用いる必要があるが，ユーザの利便性のためにはこれら複数のサーバがあたかも1つのサーバであるかのように見えることが望ましい．

クラスタの IP アドレスを1つに見せる方法としては，クラスタを代表してクライアントとの通信を仲介するノードを用意する方法と，クラスタの各ノードが同一の IP アドレスをもって個別にクライアントと通信を行う方法とがある．このうち前者の方法では代表ノードが単一障害点となること，後者の方法ではクラスタの状態に応じた柔軟な負荷分散が実現できないことが問題である．

そこで本研究では，分散型シングル IP クラスタの負荷分散性能を向上させる手法 TCP-DSC (TCP for Distributed Servers with Centralized scheduler) を提案し実装する．このシステムでは，クラスタのノードのうちの1つがマスタノードとして選出され，マスタノードが新規 TCP 接続を受理するノードを決定することで負荷の分散をはかる．接続が確立した後は各ノードが個別にクライアントとの通信を行う．マスタノードは新規 TCP 接続の受付時のみに関わるため，マスタノードが故障しても他のノードが持っていたコネクションは失われること無くクライアントとの通信を継続できる．

† 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†† 東京大学情報基盤センター
Information Technology Center, The University of Tokyo

2. 背景

本稿では，クラスタによるシングル IP サーバを実現するための方法を大きく代表ノード型と分散型の2つに分けて整理する．

2.1 代表ノード型

代表ノード型シングル IP クラスタは，Linux Virtual Server[11),13)]，SAPS[9)] などのように，通常1つの代表ノードと複数のバックエンドノードから構成される (**図1**)．代表ノードがクラスタの外部に見せる IP アドレスを持ち，外部のクライアントから送られてきた

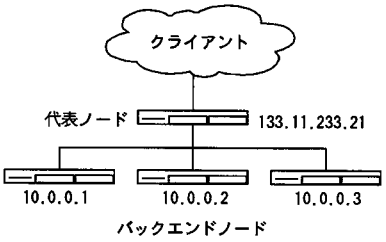

図 1　代表ノード型システムの構成例

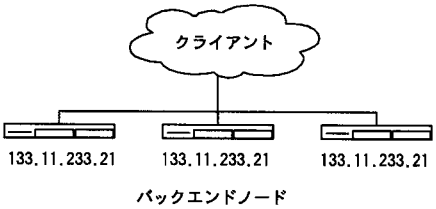

図 2　分散型システムの構成例

パケットを受け取る．代表ノードはバックエンドノードの 1 つを選び，このパケットを転送する．パケットの転送を受けたバックエンドノードは何らかの処理を行ってクライアントに応答を返す．返りのパケットが再度代表ノードで中継される構成と，中継されずに直接クライアントに対して送信される構成とがある．

代表ノードはバックエンドノードにパケットの転送を行う際に負荷分散もあわせて行う．代表ノードはクラスタ宛に到着する全てのパケットを受信するため，各バックエンドノードの負荷を把握することができる．このため，ラウンドロビンや最小コネクション数優先といった様々な負荷分散アルゴリズムを適用することができる[8]．

代表ノード型の欠点として，クライアントから入ってくるパケットが全て代表ノードによって中継されるため，代表ノードが単一障害点となることが挙げられる．すなわち代表ノードが故障した場合，クライアントとバックエンドノードとの通信は不可能になる．

2.2 分 散 型

Windows NLB (Network Load Balancing)[2] や Clone Cluster[12] などのような分散型のシングル IP クラスタでは，代表ノードを持たず，全てのノードが対等である．このタイプのクラスタにおいては，全てのノードが外部から見える同一の IP アドレスを持つ (**図 2**)．また，何らかの手段でこの IP アドレス宛に到着するパケットを全クラスタノードが受け取れるように設定しておく．

この状態ではクライアントからのパケットに対して全クラスタノードが応答してしまうため，1 つのパケットにつき 1 つのノードだけが応答するようなルールを決める必要がある．パケットが到着するたびに各ノード間で通信を行って受信するノードを決定する方式ではオーバーヘッドが大きすぎる．そのため分散型のシステムではクライアントの情報に対して静的な関数を適用しパケットを受信するかどうかを決定している．たとえば，クライアントの IP アドレスをもとに割り当てを決定することとし，クラスタノード数を N，あるノードのノード番号を i $(0 \leq i \leq N-1)$，クライアントの IP アドレスを a とすると，ノード i がそのパケットを受信するかどうかは

$$a \equiv i \ (mod N)$$

が成立するかどうかで判定できる．CloneCluster ではクライアントの IP アドレスの最下位 8 ビットを，Windows NLB ではクライアントの IP アドレスとポート番号の組み合わせをキーとして用いている．

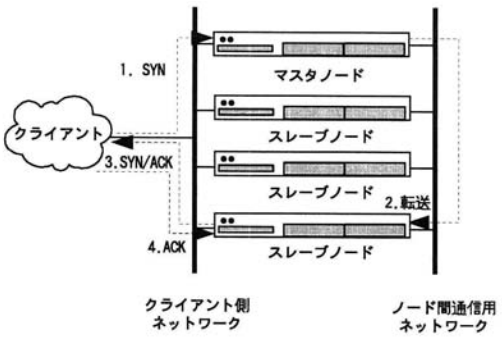

図 3　TCP-DSC の概要

分散型のシステムでは全ノードが対等であるため，代表ノードのような単一障害点を持たないという特徴がある．すなわちクラスタノードが N 台あったとき，$N-1$ 台までの故障に耐えることができる．ただし故障したノードがその時点で受け持っていたコネクションは失われる．

分散型のシステムにおいて負荷分散が有効に機能するかどうかは，クライアントの IP アドレスやポート番号といった特徴がランダムに，かつ一様に分布しているという仮定が成立するかどうかに依存している．このためアクセスするクライアントの特徴に偏りがあると特定のノードに負荷が集中する可能性がある．また長い期間について見れば各ノードへのアクセスが一様に分布している場合でも，短い期間についてみると特定のノードにアクセスが集中することが起こりうる．このため，分散型のシステムは代表ノード型に比べて負荷分散の効率が悪くなりがちであるという問題が指摘されている[12]．

3. 設　　計

本研究の提案する TCP-DSC の概要を**図 3** に示す．

TCP-DSC は分散型に分類されるシングル IP クラスタであり，分散型システムにおける TCP 接続の柔軟な振り分けと，それによる効率的な負荷の分散を目標としている．UDP をはじめとする TCP 以外の IP プロトコルについては，従来通りの分散型システムと同様に静的ハッシュ関数を用いることを前提とする．

クラスタノードのうちの 1 つは入力パケットの割り振りを指示するマスタノードを兼ねる．これ以降，便宜上クラスタ中のノードを 1 つのマスタノードとそれ以外のスレーブノードに分けて述べる．後述するように，マスタノードとスレーブノードの動作は TCP コネクション開始要求の受付を除いては同じである．

クラスタの各ノードは最低 2 つのネットワークインターフェースを持つ．片方はクライアントとの通信に使われるものであり，クラスタの全ノードで同一のパケットを受信するように設定される．もう一方のインターフェースはノード間での通信に使われ，パケットの転送や他ノードの状態監視などに使われる．

3.1 マスタノードの動作

マスタノードは SYN ビットの立った TCP セグメントを受け取ると (図 3 の 1)，スケジューリングポリシーに基づいてその接続要求をどのノードが処理すべきかを決定する．接続要求を処理しうるノードにはマスタノード自身も含まれる．もしスケジューリングの結果マスタノード以外のノードがコネクションを処理することが決まったら，ノード間通信用インターフェースを通じてそのノードにコネクション受付を指示するパケットを送る (図 3 の 2)．このパケットにはクライアントから送られてきた TCP ヘッダと，もしあればペイロードが書かれている．

マスタノードにおけるスケジューリングポリシーは置き換え可能となっており，サーバの目的に応じた様々なスケジューリングアルゴリズムが実装可能である．また，SYN 転送の際にはスケジューリングポリシーからの要請に応じて転送先ホストを記録することも行う．この機構については後に 3.5 で詳しく述べる．

3.2 スレーブノードの動作

スレーブノードでは，SYN ビットの立った TCP セグメントは全て破棄する．スレーブノードが新しい接続を受け付けるのは，マスタノードからの接続受付指示パケットを受け取ったときのみである．

接続の受付を指示するパケットを受け取ったスレーブノードは，そのパケットから TCP ヘッダとペイロードを取り出して TCP 層に渡す．これ以降は通常の TCP 通信と同じく，SYN ビットと ACK ビットが立ったセグメントをクライアントに送り返し (図 3 の 3)，再度クライアントから ACK ビットが立ったパケットを受信した時点で接続が確立される (図 3 の 4)．

3.3 接続確立後の通信

一度接続が確立した後は，SYN ビットの立っていないセグメントを受信するかどうかは各ノード自身が持つ TCP コネクション表を参照することで判定できる．すなわち，自分が持っている確立済み接続に属するセグメントは受信し，それ以外のセグメントは破棄する．この際通常の TCP の動作と異なり RST は送出しない．

3.4 故障への対応

TCP-DSC は分散型シングル IP クラスタの一種であるため，ノードの故障が起きた場合でもその故障に影響される範囲は限定される．スレーブノードが故障した場合，そのノードが受け持っていた接続は失われるが，それ以外のノードが受け持っていた通信は継続できる．また，マスタノードが故障した場合も同様に他のノードで確立済みの接続は維持される．

しかしながら故障したノードはそれ以降新しい接続を受け付けることができないため，ノードの故障に応じてクラスタを再構成する必要がある．特にマスタノードの故障時にはクライアントからの新しい接続が一切受け付けられなくなるため，新しいマスタノードを選出しなければならない．

故障に備えてノードの状態を監視するため，各ノード上で状態監視デーモンを走らせる．マスタノード上のデーモンは定期的に生存確認パケットをブロードキャストし，スレーブノードは生存確認パケットに対して応答を返す．

あるスレーブノードから一定時間生存確認に対する応答がない場合，マスタノードはそのスレーブノードが故障したと判断し，コネクション割り振りの対象から除外する．一方マスタノードから一定期間生存確認のブロードキャストを受け取らなかったスレーブノードは，マスタノードが故障したと判断し，新しいマスタノードを選ぶための選挙を開始する．TCP-DSC は分散型システムであるため，全てのスレーブノードが新しいマスタノードとなることができる．すなわち，最後の 1 ノードになるまでクラスタとしての機能を維持することができる．

耐障害性の観点から，マスタノードにはクラスタ全体の通信の続行に不可欠な状態を持たせないようにする．ただし，マスタノード上においてクラスタ全体の通信の維持にとって不可欠でない状態なら保持することを許可する．例えば，ラウンドロビンスケジューラは前回どのノードにパケットを配信したかという情報を保持しているが，もしこれが失われたとしても若干スケジューリング効率が悪化する可能性があるだけであって，通信の維持には影響がない．

3.5 SYN セグメント再送への対処

TCP コネクションの開始時においてサーバ側からの SYN/ACK セグメントが失われた場合 (**図 4**)，クライアント側は最初の SYN セグメントを再送してくるが，マスタノードがラウンドロビンのようなスケジューリングポリシーを使用している場合，この再送された SYN セグメントを別のノードに転送してしま

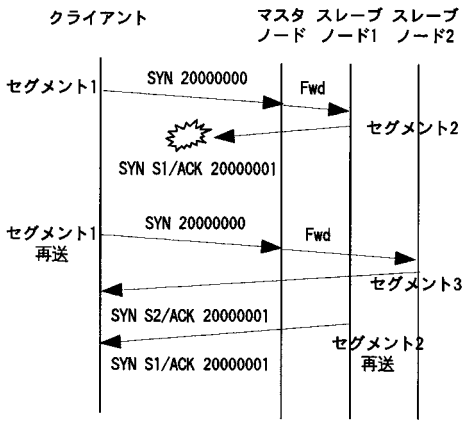

図 4 SYN/ACK セグメントが失われた場合

う可能性がある．

SYN セグメントの転送が複数回別々のスレーブノードに対して行われると，複数のノードが同一の SYN セグメントを受信した形になる．SYN セグメントを受信したノードは初期シーケンス番号 (ISN) を生成した後クライアントに SYN/ACK セグメントを送信し，SYN-RECEIVED 状態に移行しクライアントからの ACK を待つ．ISN はランダムに選ばれるため，通常は複数のサーバノードからの SYN/ACK セグメントは全く異なったシーケンス番号を持っている．図 4 の例では，スレーブノード 1 からの ISN を S1，スレーブノード 2 からの ISN を S2 としている．図 4 ではスレーブノード 2 からのセグメントが先にクライアントに到着するように描かれているが，スレーブノード 1 からのセグメントが先に受信される場合もある．

確率としては非常に低いものの，S1 と S2 が等しいか，S1 と S2 の差が小さい場合，複数のノードが同時に ESTABLISHED 状態に入りうるため，複数のサーバノードからのデータが混在してクライアントに受信される．そのため正しい通信を継続することが困難となる．

例えば S2=1000000，S1=1000100 のとき，スレーブノード 2 が先に ESTABLISHED 状態になった後何度かデータの送信を行って合計 100 オクテット送信した状況が生まれたとする．すると次にクライアントから返される ACK 番号は 1000101 となるので，2 つのサーバノードがどちらもその ACK を自分宛のものであると解釈して ESTABLISHED 状態になる．

このような状態に入ることを防ぐため，マスタノードにおいて SYN セグメントの転送が行われた際，そのパケットをどのノードに転送したかを一定期間保存しておく．記録する情報は，クライアントの IP アドレス，TCP ポート番号，転送先ノードのアドレスである．

最初の SYN セグメントや転送パケットが失われた場合は，スレーブノード上に状態変化が起こらないため問題ない．また，スケジューリングポリシーがクライアントの IP アドレスに基づくような静的なものであった場合，SYN の再送を受け取っても必ず同一のノードに SYN を転送するため，転送先の保存は必要ない．

マスタノードがクラッシュした後に新しいマスタノードが選出された場合も同様の注意が必要である．新しく選出されたマスタノードは自分自身および各スレーブノードが持っている SYN-RECEIVED 状態のソケットについて把握し，SYN 転送先リストを再構成した後に新しい SYN セグメントの転送を開始しなければならない．

4. 実　　装

提案手法の負荷分散性能を評価するため，Linux カーネルに対し TCP 接続の転送を行う部分についての実装を行った．この実装はカーネルの IP 層と TCP 層の間に新たな機構を加えるものであり，サーバ上で動くアプリケーションにとっては TCP-DSC カーネルはごく普通の Linux カーネルに見える．またクライアント側の TCP スタックにとっても TCP-DSC は普通の TCP 端末に見えるため，任意の TCP クライアントと通信可能である．

厳密には設計上の制限から，TCP-DSC はどのノードの接続済みソケットにも属さず，かつ SYN ビットの立っていない TCP セグメントを受信した場合に RST を返さない．これは，TCP-DSC が各ノードにおいて独立に TCP セグメントを受信するかどうか決定しているからであり，誰も受信しないという状況が判断できないためである．これは TCP 本来の挙動と異なるが，ファイアウォールやパケットフィルタを使用した場合には通常の TCP スタックを実装した端末においても同様の挙動を示す場合があるため，この制限は実際上問題にならないと考えられる．

4.1 ユーザ空間とのインターフェース

ユーザ空間からスケジューラの種類やパラメタを指示できるようにするため，setsockopt システムコールを用いたインターフェースを提供した．ユーザは設定プログラムを用いて，シングル IP クラスタとして受信する IP アドレス，スケジューラの種類，スケジューラへのオプション，スレーブノードのリストを登録する．

4.2 パケット操作の実装

カーネル内でパケットを操作する部分は netfilter[4] のモジュールとして実装した．netfilter は Linux カーネル内でネットワークのパケットの入力，出力，転送などが起きた際にそれらのイベントのフックを行い，パケットの破棄や書き換えを可能にする機構である．

4.2.1 SYN セグメントの受信

受信パケットに対するフィルタリングは，トランスポート層にパケットが渡される直前で行う．プロトコルタイプが TCP であり，かつ TCP ヘッダに SYN ビットが立っているパケットを受信すると，先に 4.1 で登録しておいた設定を検索し，そのパケットが TCP-DSC が扱う IP アドレス宛かどうかを調べる．

パケットが TCP-DSC のアドレス宛であれば，設定を参照して自分がマスタノードかどうかを判断し，マスタノードでなければその SYN セグメントを破棄する．マスタノードでは，設定に登録されているスケジューラを呼び出し，TCP 接続を受け付けるノードを決定する．決定されたノードが自分以外であれば，そのアドレス宛に SYN セグメントを送出する．ノード間で SYN セグメントを転送する際には，IP パケットタイプを 253 とし，受信側が TCP-DSC の転送パケットとわかるようにする．

4.2.2 転送パケットの受信

マスタノードによって選ばれたスレーブノードには SYN セグメントが転送されてくる．スレーブノードはこのパケットを受け取ると，パケットを TCP スタックの受信ルーチンへと渡す．すると TCP 層はあたかも自分のノードが直接 SYN セグメントを受け取ったかのように動きだし，SYN-REVEIVED 状態へと移行する．

4.2.3 接続確立後のセグメント

接続が確立した後のセグメント，すなわち SYN ビットの立っていない TCP パケットに関しては一切のフィルタリングを行わない．もしそのセグメントが自分のノードの持つコネクションに属するものであれば，通常の TCP 受信処理が行われる．そうでない場合には TCP 層は RST を送出しようとするが，クライアントが不要な RST を受信することで接続が中断されてしまうため，やはり netfilter を用いて送出されようとしている RST セグメントを破棄する．これによって自分に関係のないパケットの無視を実現している．

4.3 Ethernet フレームの分配

本来 Ethernet においては同一ブロードキャストドメインに存在するノードは物理的な通信媒体を共有し，他ノード宛フレームも含め全て受信可能であった．しかし現在ではスイッチの普及が進み，ほとんどの場合ユニキャストアドレス向けフレームは 1 対 1 通信で送られるため，そのままでは複数ノードが同一の入力フレームを共有することができない．これに対処するための手法がいくつか提案されている[2),12)]．

現在の TCP-DSC の実装ではスイッチのポートミラーリング機能を用い，全サーバノードのクライアント側インターフェースに同一のフレームを配送するように設定することでこの問題を解決している．また，全マシンのクライアント側インターフェースカードに全て同一の MAC アドレスを登録することで，プロミスキャスモードを用いることなくフレームの受信を可能にする．

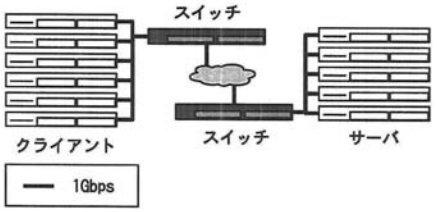

図 5 実験環境

表 1 実験に使用した計算機

	サーバ
CPU	AMD Opteron 175 2.2GHz × 2
メモリ	2GB
Ethernet カード	Intel Pro/1000 Server (クライアント側インターフェース) nForce Professional (オンボード) (クラスタノード間インターフェース)
カーネル	Linux 2.6.20 (x86_64)
	クライアント
CPU	Intel Xeon 2.80GHz × 2
メモリ	1GB
Ethernet カード	Intel Pro/1000 Server
カーネル	Linux 2.6.18 (i386)

5. 評　　価

5.1 実験環境

実験に用いた環境を**図 5** および**表 1** に示す．サーバ側に 5 台，クライアント側に 6 台の計算機を用意し，その間は 1Gbps の Ethernet で接続されている．実際の環境では合計 6 台の L2 スイッチがサーバ・クライアント間の経路上に存在しているが，図では省略している．

5.2 基本性能

5.2.1 レイテンシ

SYN セグメントがマスタノードからスレーブノードに転送された場合，本来の TCP 接続を開始するためのレイテンシに加えて 1-hop 分パケットを転送するための遅延が生じるはずである．この遅延がどの程度かを測定するため，SYN 転送を全く行わなかった場合と常に行った場合について，ユーザーレベルのアプリケーションにおいて TCP 接続要求を出してから完了するまでの時間 (connect システムコールに要する時間) を計測した．

測定結果を**表 2** に示す．測定はそれぞれ 10000 回行った．ARP 解決等の影響を排除するため，各測定の直前に 1 回の connect を試みており，その分の値は結果には含めていない．表から，SYN の転送により平均で約 12μs の遅延が生じることがわかる．

表 2　レイテンシ測定結果 (単位 μs)

SYN 転送	最小	平均	最大	標準偏差
転送なし	116	203.15	302	38.66
転送あり	124	215.78	511	38.90

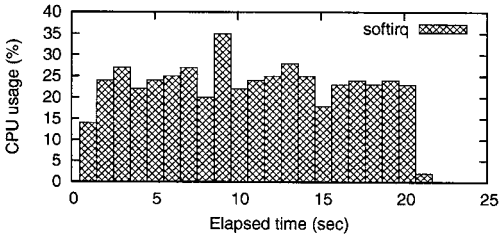

図 6　通信と無関係なノードにおける CPU 負荷 (上り方向トラフィック)

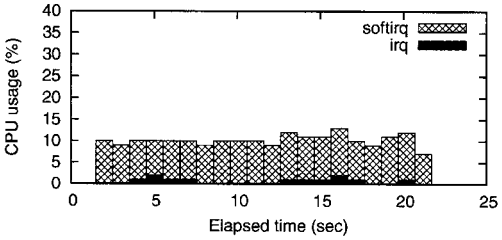

図 7　通信と無関係なノードにおける CPU 負荷 (下り方向トラフィック)

表 3　帯域幅とパケット受信頻度

トラフィック	帯域幅 (Mbits/sec)	サーバノードにおけるパケット受信頻度 (packets/sec)
上り方向	936	81504
下り方向	936	32066

5.2.2 CPU 負荷

一般に分散型シングル IP クラスタにおいては，各ノードが自分に関係のないパケットも一旦受信した後にソフトウェアでそのパケットが必要かどうかの判定を行う必要があるため，代表ノード型のシステムにおけるバックエンドノードに比べて各ノードの CPU 負荷が高くなることが指摘されている[2]．そこで，あるサーバノードがクライアントと通信を行っている時に他のサーバノードにおいて CPU 使用率を測定する実験を行った．

具体的には，iperf[3] を用いて 1Gbps の帯域をほぼ使い切る TCP バースト通信を行う．通信は片方向通信とし，クライアントからサーバにデータが向かう方向 (上り方向) とサーバからクライアントにデータが向かう方向 (下り方向) とに分けてそれぞれ 20 秒間の測定を行う．TCP 通信では確認応答のパケットが一定間隔ごとに逆方向に流れるため，上り方向トラフィックがあるときだけでなく，下り方向トラフィックのみが存在する場合でも，サーバノードが受け取るパケットは 0 にはならない．

本実験の結果を**図 6** および**図 7** に示す．図 6 は上り方向トラフィックが，図 7 は下り方向トラフィックが存在するときの CPU 使用率を示している．グラフは積み上げグラフである．グラフ中，irq はハードウェア IRQ コンテキストで消費された CPU 時間を，softirq は softirq コンテキストで消費された CPU 時間を表している．また，このとき観測された帯域幅およびパケット受信頻度は**表 3** に示す通りであった．

グラフから，上り方向のバースト通信が存在する状況では CPU 時間の約 25%がパケットの受信のために使われることがわかる．一方，下り方向のバースト通信が存在する際にはクライアントから返ってくる ACK の受信のために約 10%の CPU 時間を消費している．この比は表 3 に示したパケット受信頻度の比と一致する．

5.3 負荷分散性能

マスタノード上の集中的スケジューラの必要性を示すため，Apache HTTP サーバ[6] を用いて高負荷状態において負荷を分散する実験を行った．

本実験では，クライアント側から httperf[10] を用いて大量の HTTP リクエストをサーバに対して送り，応答が得られるまでの平均応答時間を計測する．また同時にサーバノード上で CPU 負荷を計測し，各ノードにどれだけの負荷がかかっているかを記録する．

クライアントからアクセスされるページは FSWiki[1] の作成する動的ページとした．FSWiki は Perl で書かれた Wiki エンジンの 1 つであり，CGI を経由して動作する．このページに対し，各クライアントから秒間 10 接続 (クライアント 6 台で合計秒間 60 接続) の頻度で 60 秒間アクセスを行う．

本実験では，サーバを TCP-DSC によるクラスタとし，マスタノードはラウンドロビンスケジューラによって TCP 接続の割り振りを行う．

5.3.1 比較対象

比較対象として，既存の分散型シングル IP クラスタと同等の動作をするパケットフィルタリングを実装し，全く同じ実験を行う．

この実装では，$I \equiv i \ (mod\ N)$ であるようなノード i が接続を受け付ける．ただし，N はクラスタ全体のノード数，$i\ (0 \leq i < N)$ はノード番号，I はクライアントからの SYN セグメントに記載された初期シーケンスナンバー (ISN) である．Windows NLB や CloneCluster といった既存実装と同じ動作とするため，マスタノードによる判断は行わず，各ノードが自律的に接続の受付を判断する．割り当てのためのキーとしてクライアントの IP アドレスやポート番号を用いなかったのは，クライアント台数が少ないため IP アドレス空間が偏ること，クライアントの Linux カーネルの性質として短い時間間隔の中では連続した一時

表 4　平均応答時間

方式	Connection Time (ms)	Reply Time (ms)
ラウンドロビン	542.65	214.50
ISN ハッシュ	731.52	435.33

ポート番号を割り当てる傾向があり，その結果ポート番号のランダム性が期待できないことによる．

5.3.2 結　　果

表 4 は CGI を用いたベンチマークにおいてサーバからの平均応答時間を測定したものである．Connection Time は HTTP サーバに対する TCP 接続を試みてから確立するまでの時間，Reply Time は HTTP サーバに対してリクエストラインを送出した後サーバから最初の応答が返ってくるまでの時間である．いずれもラウンドロビンの方が性能がよく，特に Reply Time においては半分以下の時間で応答が得られている．

図 8 および **図 9** はそれぞれ，CGI ベンチマークを行ったときのサーバノードにおける CPU 使用率をグラフ化したものである．ある時点における 5 台のサーバノードそれぞれの CPU 使用率の最小値，平均値，最大値をそれぞれプロットした．最大値と最小値が平均値に近いほど負荷が均等に分散されていることを表している．ISN に基づくハッシュ関数で接続を割り振った場合，負荷が最大になっているノードでは CPU 使用率がほぼ常に 100%に達している一方，負荷が最も低いノードでは 50% 程度しか CPU を消費していない．ラウンドロビン方式でスケジューリングした際には，ISN に基づくハッシュ関数を用いた場合よりも各ノードの負荷が均等に近づいていることがわかる．表 4 における応答時間の差はこの負荷の偏りによって生じたと考えられる．

5.4 議　　論

5.2.2 に示したように，TCP-DSC におけるパケット受信時の負荷は非常に高い．この CPU サイクルのうち，一部は削減することが可能であると考えられる．現在の実装では，SYN ビットの立っていない TCP セグメントについては何ら手を加えることなく TCP 層に渡して処理をまかせているが，より早い段階でパケットが必要かどうかを判断することで無駄に消費される CPU サイクルを削減できると考えられる．

6. 関 連 研 究

Hive システム[14] は本研究と同様に分散型シングル IP クラスタでの負荷分散を目標としたものである．Hive システムではクライアントの IP アドレスをもとにパケットを受信するかどうか決定するが，その際に静的ハッシュ関数ではなく全ノードで共有される表を用いている．この表が各ノードの負荷に応じて動的に更新されることで負荷の分散を行う．

Round Robin DNS[7] はインターネット上の複数の IP アドレスに対して複数のドメイン名を提供し，DNS クライアントからの問い合わせに対してラウンドロビン式に回答する IP アドレスの順番を変えていくものであるが，このような DNS ベースの機構と本研究で対象としているシングル IP 機構は相反するものではなく，組み合わせて用いられるべきものである．

Ultramonkey プロジェクトの Saru[5] は分散型シングル IP クラスタの機構を Linux Virtual Server の代表ノード (Director) に適用したものである．Saru では 2 台の Director を用意し，双方に同じパケットが配信されるよう設定する．それぞれの Director が着信したパケットを受け付けるかどうかは静的なハッシュ関数によって決定される．Saru では，ハッシュ関数の定義域 (例えば，IP アドレス空間) をいくつかの「ブロック」に分割し，それぞれの Linux Director が担当するブロックを定期的に交換しあうことで Linux Director 間の負荷分散をはかる．Saru は基本的に代表ノード型システムの亜種であるため，最初の SYN セグメント以降のパケットも必ず Director を通過することになり，もし片方の Director がクラッシュした際にはこの Director が受け持っていたコネクションは失われることになる．

7. おわりに

本研究では，分散型シングル IP クラスタに TCP 接続の集中的スケジューリングを行うノードを導入したシステム TCP-DSC を提案し，Linux 上に実装した．TCP-DSC は，従来の分散型システムの問題点であった静的な接続割り振りによる負荷の偏りを，集中型スケジューラの導入によって解決する．Apache 上の CGI プログラムに対して大量のアクセスを行う実験の結果，TCP-DSC 上でラウンドロビンスケジューラを用いることで，従来の静的な接続割り当て手法と比較して約 50% の平均応答時間を達成できることが示された．

今後の課題としては，不要なパケットを破棄する処理をより少ない CPU 負荷で行えるようにすること，障害からの回復に要する時間も含めた評価を行うこと，より現実に即した負荷に対して静的負荷分散手法と動的な集中スケジューリング手法との性能差を明らかにしていくこと，代表ノード型システムを含めた性能比較を行うこと，が挙げられる．

謝辞　本研究の一部は，科学技術振興機構 戦略的創造研究推進事業 (CREST) (領域名：実用化を目指した組込みシステム用ディペンダブル・オペレーティングシステム) 技術課題：「高信頼組込みシングルシステムイメージ OS」による．

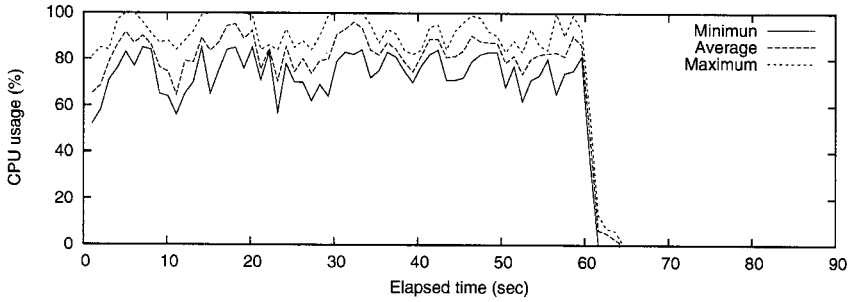

図 8　ラウンドロビンを用いた際の CPU 使用率の推移

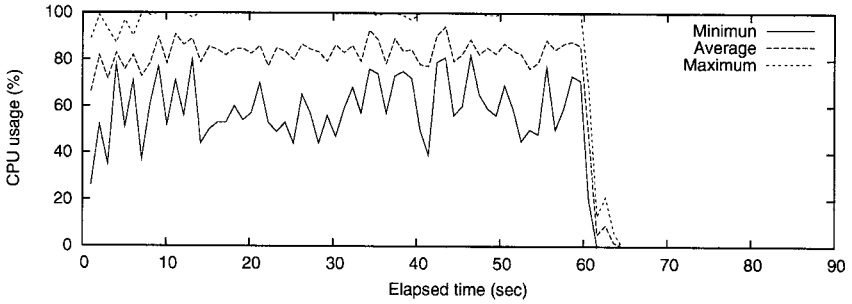

図 9　ISN ハッシュを用いた際の CPU 使用率の推移

参 考 文 献

1) FreeStyleWiki. `http://fswiki.poi.jp/`.
2) Network Load Balancing Technical Overview. `http://www.microsoft.com/technet/prodtechnol/windows2000serv/deploy/confeat/nlbovw.mspx`.
3) Nlanr/dast : Iperf - the tcp/udp bandwidth measurement tool. `http://dast.nlanr.net/Projects/Iperf/`.
4) The Netfilter.org project. `http://www.netfilter.org/`.
5) 猿 Saru: Active-Active. `http://www.ultramonkey.org/papers/active_active/`.
6) The Apache HTTP Server. `http://httpd.apache.org/`.
7) T. Brisco. DNS Support for Load Balancing. RFC 1794 (Informational), April 1995.
8) Valeria Cardellini, Emiliano Casalicchio, Michele Colajanni, and Philip S. Yu. The state of the art in locally distributed web-server systems. *ACM Computing Surveys*, Vol. 34, No. 2, pp. 263–311, 2002.
9) Hiroya Matsuba and Yutaka Ishikawa. Single IP address cluster for internet servers. In *Proceedings of 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS2007)*, 2007.
10) David Mosberger and Tai Jin. httperfa tool for measuring web server performance. *SIGMETRICS Performance Evaluation Review*, Vol. 26, No. 3, pp. 31–37, 1998.
11) Patrick O'Rourke and Mike Keefe. Performance Evaluation of Linux Virtual Server. *LISA 2001 15th Systems Administration Conference*, 2001.
12) Sujit Vaidya and Kenneth J. Christensen. A single system image server cluster using duplicated MAC and IP addresses. In *Proceedings of the 26th Annual IEEE Conference on Local Computer Networks*, pp. 206–214, 2001.
13) Wensong Zhang. Linux Virtual Servers for Scalable Network Services. *Linux Symposium*, 2000.
14) 瀧ヶ平健, Stanislav G. Sedukhin. マルチキャストによるクラスタ web サーバの構築. In *Proceedings of Internet Conference 2001*, 2001.